Representing and Reasoning with Large Games

by

Xin Jiang

B. Science, University of British Columbia, 2003

M. Science, University of British Columbia, 2006

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

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University Of British Columbia

(Vancouver)

December 2011

© Xin Jiang, 2011

Abstract

In the last decade, there has been much research at the interface of computer science and game theory. One important class of problems at this interface is the computation of solution concepts (such as Nash equilibrium or correlated equilibrium) of a finite game. In order to take advantage of the highly-structured utility functions in games of practical interest, it is important to design compact representations of games as well as efficient algorithms for computing solution concepts on such representations. In this thesis I present several novel contributions in this direction:

The design and analysis of Action-Graph Games (AGGs), a fully-expressive modeling language for representing simultaneous-move games. We propose a polynomial-time algorithm for computing expected utilities given arbitrary mixed strategy profiles, and leverage the algorithm to achieve exponential speedups of existing algorithms for computing Nash equilibria.

Designing efficient algorithms for computing pure-strategy Nash equilibria in AGGs. For symmetric AGGs with bounded treewidth our algorithm runs in polynomial time.

Extending the AGG framework beyond simultaneous-move games. We propose Temporal Action-Graph Games (TAGGs) for representing dynamic games and Bayesian Action-Graph Games (BAGGs) for representing Bayesian games. For certain subclasses of TAGGs and BAGGs we gave efficient algorithms for equilibria that achieve exponential speedups over existing approaches.

Efficient computation of correlated equilibria. In a landmark paper, Papadim-

itriou and Roughgarden described a polynomial-time algorithm (”Ellipsoid Against Hope”) for computing sample correlated equilibria of compactly-represented games. Recently, Stein, Parrilo and Ozdaglar showed that this algorithm can fail to find an exact correlated equilibrium. We present a variant of the Ellipsoid Against Hope algorithm that guarantees the polynomial-time identification of exact correlated equilibrium.

Efficient computation of optimal correlated equilibria. We show that the polynomial-time solvability of what we call the *deviation-adjusted social welfare problem* is a sufficient condition for the tractability of the optimal correlated equilibrium problem.

Preface

Certain chapters of this thesis are based on publications (or submissions to publications) by my collaborators and me (under the name Albert Xin Jiang). Per requirement of UBC Faculty of Graduate Studies, I describe here the relative contributions of all collaborators.

Chapter 3 is based on the article *Action-Graph Games* by Albert Xin Jiang, Kevin Leyton-Brown and Navin Bhat, published in Games and Economic Behavior, Volume 71, Issue 1, January 2011, Pages 141–173, Elsevier. Navin and Kevin first proposed Action-Graph Games without function nodes (called AGG-$\emptyset$s in this thesis), proposed an algorithm for computing expected utility for the symmetric case, and proposed an approach for computing sample Nash equilibria in symmetric AGG-$\emptyset$s, by adapting Blum et al. [2006]'s approach for speeding up Govindan and Wilson's [2003] global Newton method. My main contributions include: 1) extending the basic AGG-$\emptyset$ representation by introducing function nodes and additive structure, yielding the more general representations AGGs with Function Nodes (AGG-FNs) and AGG-FNs with Additive Structure (AGG-FNAs); 2) proposing and implementing an algorithm for computing expected utility for general AGGs, and proving that it runs in polynomial time; 3) implementing software packages for game-theoretic analysis using AGGs, including programs that speed up existing algorithms for sample Nash Equilibria [Govindan and Wilson, 2003, van der Laan et al., 1987] by leveraging the expected utility algorithm; 4) carrying out computational experiments; 5) preparation of the manuscript. Kevin has played a supervisory role throughout the project.

Chapter 4 is based on the paper *Computing Pure Nash Equilibria in Symmetric Action Graph Games* by Albert Xin Jiang and Kevin Leyton-Brown, published

in the Proceedings of AAAI, 2007, although the chapter contains a significant amount of new material. My main contributions include: 1) identification of the research problem and the design of the overall approach; 2) working out the details of our algorithm and proving its correctness and running time; 3) preparation of the manuscript. Kevin has played a supervisory role throughout the project.

Chapter 5 is based on the paper *Temporal Action-Graph Games: A New Representation for Dynamic Games* by Albert Xin Jiang, Kevin Leyton-Brown and Avi Pfeffer, published in the Proceedings of UAI, 2009. The identification and design of the overall research program is done via joint discussions by all three co-authors. My other contributions include: 1) working out the details of the Temporal Action-Graph Game representation and our algorithm for computing expected utility, and proving their properties; 2) implementing our algorithm and carrying out computational experiments; 3) preparation of a majority of the text in the manuscript. Kevin has played a supervisory role throughout the project.

Chapter 6 is based on the paper *Bayesian Action-Graph Games*, published in the Proceedings of NIPS, 2010. The identification and design of the overall research program is done via joint discussions by both co-authors. My other contributions include: 1) working out the details of the Bayesian Action-Graph Game representation, our algorithm for computing expected utility and our approach for computing Bayes-Nash equilibrium, and proving their properties; 2) implementing our algorithm and carrying out computational experiments; 3) preparation of the manuscript. Kevin has played a supervisory role throughout the project.

Chapter 7 is based on the paper *Polynomial-time Computation of Exact Correlated Equilibrium in Compact Games* by Albert Xin Jiang and Kevin Leyton-Brown, published in the Proceedings of ACM-EC, 2011. My main contributions include: 1) identification of the research program; 2) design of our algorithm and analysis of its properties; 3) preparation of the manuscript. Kevin has played a supervisory role throughout the project.

Chapter 8 is based on the manuscript *A General Framework for Computing Optimal Correlated Equilibria in Compact Games* by Albert Xin Jiang and Kevin Leyton-Brown, published in the Proceedings of the Seventh Workshop on Internet and Network Economics (WINE), 2011. My main contributions include: 1) identification of the research program; 2) design of our algorithm and analysis of its

properties; 3) preparation of the manuscript. Kevin has played a supervisory role throughout the project.

Table of Contents

List of Figures

Acknowledgments

First and foremost I would like to thank my parents, for their unconditional love and support, for their wisdom, and for encouraging me to pursue my interests. I am the person I am because of them, and I am very lucky to have them as parents.

Kevin Leyton-Brown has been my advisor since my MSc degree. He has introduced me to game theory, and mentored me through all the research projects described in this thesis. I am eternally grateful to him for being a great teacher and communicator, for showing me his research vision yet giving me the freedom to explore and find my research topics, for helping me refine my half-formed ideas, for giving me concrete advice and pushing me to be better at all aspects of being a researcher, and for the career opportunities he introduced me to. I can honestly say that I really enjoyed my Ph.D. experience.

I would like to thank fellow members of Kevin's game theory group and my office mates, David Thompson, James Wright and Baharak Rastegari, for stimulating discussions on research and otherwise, and for the camaraderie. I have also had many enjoyable discussions with Chris Ryan while he was doing his Ph.D. in Operations Research at UBC, during which he introduced me to quite a few interesting mathematical concepts including algebraic geometry and generating functions.

I would like to thank David Poole and Joel Friedman for serving on my supervisory committee, my university examiners Michael Friedlander and Sergei Severinov, and my external examiner David Parkes. They have given me very helpful feedbacks on my thesis. Many members of the algorithmic game theory research community have given me encouragement and help during my studies, I would like to especially mention Vince Conitzer, Christos Papadimitriou, Tim Roughgarden, Tuomas Sandholm, and Ted Turocy. I would also like to thank all my collaborators,

some of which I have mentioned above: Kevin Leyton-Brown, Navin Bhat, Avi Pfeffer, Mohammad Ali Safari, Chris Ryan, Nando de Freitas, Michael Buro, David Thompson, James Wright, and Damien Bargiacchi.

During my Ph.D. studies I was supported by UBC's University Graduate Fellowship for one year, the NSERC Canada Graduate Scholarship for three years, and partially by a Google Research Award "Advanced Computational Analysis of Position Auction Games". I would like to thank them for their financial support.

Chapter 1

Introduction

Game theory is a mathematical theory of *games*, interactions in which multiple autonomous agents, each with their own utility functions, act according to their own interests. Game theory has received a great deal of study, and is perhaps the dominant paradigm in microeconomics [e.g., Fudenberg and Tirole, 1991]. In the last decade, there has been much research at the interface of computer science and game theory [e.g., Nisan et al., 2007, Shoham and Leyton-Brown, 2009]. This interdisciplinary field has been named "algorithmic game theory", "computational economics", and "multiagent systems" by various researchers. This recent interest in game theory by the computer science community has been partially motivated by the explosion in the popularity of the Internet, which is essentially a network of computers controlled by selfish agents. There is thus much recent effort to apply game theory to various subdomains of the Internet such as TCP/IP routing, peer-to-peer sharing, auction environments including eBay and AdWords, and social networks.

One fundamental class of computational problems in game theory is the computation of *solution concepts* of a finite game. Examples of solution concepts include Nash equilibrium and correlated equilibrium. Intuitively, these solution concepts are answers to the following type of questions: what are the likely outcomes of the game, under certain models of rationality of the agents? Thus the task of computing these solution concepts can be understood in the language of AI as *reasoning* about the game. The goal is to be able to efficiently carry out such reasoning for

real-world multiagent systems. One application of such game-theoretic reasoning is the development of autonomous agents that can act intelligently by taking into account the strategic behavior of other agents. Another application is to help the designer of a system to predict its likely outcomes and to optimize the parameters of the system to achieve preferred outcomes. Furthermore, some computer scientists argue that the complexity of these computational problems have implications on whether equilibria can be reached in practice. A famous quote by Kamal Jain is "if your laptop cannot find the equilibrium, neither can the market."

The input to such computational problems is a description of the game. Most of the game theory literature presumes that simultaneous-action games will be represented in normal form. This is problematic because in many domains of interest the number of players and/or the number of actions per player is large. In the normal form representation, the game's payoff function is stored 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. A similar problem arises in dynamic games, for which the extensive form serves as the standard representation. For large games, it becomes infeasible to store the game in memory. Computations that require time polynomial in the input size are nevertheless impractical.

Fortunately, most large games of practical interest have highly-structured payoff functions, and thus it is possible to represent them *compactly*, by which we mean a representation that is exponentially smaller than its induced normal form. Intuitively, this helps to explain why people 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 lookup tables. Of course, there are any number of ways of representing games compactly. For example, games of interest could be assigned short ID numbers. But we ultimately want to be able to compute solution concepts of the games, and we would like the running time of our algorithms to depend on the size of the compact representation rather than the size of the corresponding normal form.

Can we design representations of games that are able to compactly encode a wide range of interesting games and are amenable to efficient computation? And how do we design efficient algorithms for computing solution concepts in these

compactly represented games? These are the central questions I tackle in this thesis.

Before discussing my contributions, I will first briefly summarize the relevant literature; I will give a more in-depth survey in Chapter 2. One thread of recent work in the literature has explored compact game representations (also called concise or succinct representations) that are able to succinctly describe games that exhibit certain types of structure. Examples of such representations for complete-information simultaneous-action games include anonymous games, graphical games [Kearns et al., 2001], and congestion games [Rosenthal, 1973]. Examples of structure include symmetry/anonymity, strict and action-specific independence, and additivity. However, the existing representations either only capture a subset of these types of structure, or are only able to represent a subset of games that exhibit a specific structure. There is a lack of a general modeling language that is fully expressive (able to express arbitrary games) while also able to compactly encode utility functions exhibiting commonly-encountered types of structure.

Nash equilibrium (NE) is perhaps the most well-known and well-studied game-theoretic solution concept. There is a line of recent results from the computational complexity theory community on the hardness of various computational problems regarding Nash equilibria, perhaps most prominently the series of papers [Chen and Deng, 2006, Daskalakis et al., 2006b, Goldberg and Papadimitriou, 2006] establishing the PPAD-completeness of the the problem of finding a sample mixed-strategy Nash equilibrium in normal-form games of two or more players. I take the view that although these hardness results are important for understanding the problems, they do not imply that practical algorithms cannot be built. For example, there has been great advances in the design and implementation of practical solvers for theoretically hard problems such as SAT and integer programming. In terms of algorithms for finding a Nash equilibrium, earlier literature from economics and operations research focused on algorithms for the normal form [e.g., Govindan and Wilson, 2003, van der Laan et al., 1987]. In the last decade, with more compact game representations being proposed, there has been more efforts from the computer science community on algorithms for compact representations. Such efforts can roughly be divided into two categories, "black-box" approaches and "special-purpose" approaches. A *black-box* algorithm requires certain subroutines

provided by the representation to work, but otherwise treats the representation as a black box. Examples include efforts to adapt algorithms designed for the normal form to compact representations [Bhat and Leyton-Brown, 2004, Blum et al., 2006]. The computation of *expected utility* has emerged as a key subtask required by many black-box algorithms. The ability to carry out this computation efficiently has become an important design criterion for compact representations. Fortunately, most existing representations admit polynomial-time algorithms for expected utility. The existing black-box approaches are for the problem of finding a sample Nash equilibrium; while this problem is very important, we are often interested in questions regarding the set of equilibria such as finding the optimal equilibrium. On the other hand, a *special-purpose* approach tries to exploit certain specific structure of the game, and is thus specific to the representation. Although not as general as the black-box approach, a special-purpose approach can often identify tractable subclasses of games while the general case is hard; furthermore it can sometimes compute a concise description of the set of equilibria, allowing us to e.g., compute the optimal equilibrium. Examples include algorithms for computing pure-strategy Nash equilibria for tree graphical games [Daskalakis and Papadimitriou, 2006, Gottlob et al., 2005] and singleton congestion games [Ieong et al., 2005], and for computing mixed-strategy Nash equilibria for symmetric games [Papadimitriou and Roughgarden, 2005] and anonymous games [Daskalakis and Papadimitriou, 2007]. In terms of software implementations, the GAMBIT [McKelvey et al., 2006] package contains many of the existing algorithms for the normal form and the extensive form. There is a relative lack of publicly-available implementations of algorithms for compact representations, except for the Gametracer [Blum et al., 2002] package which provides implementations of black-box adaptations of two of Govindan and Wilson's algorithms [Govindan and Wilson, 2003, 2004] for finding a sample Nash equilibrium.

In summary, although there have been many advances in the theoretical understanding of how certain types of structure in games can be exploited for efficient computation, the lack of a general representation and publicly available software implementations for structured games meant that the computational analysis of large games has not become practical. Much of this thesis can be understood as my efforts to address these problems. Below I give an outline of my contributions,

including the design of game representations that can capture a wide variety of computation-friendly structure, novel algorithms for computing sample equilibria as well as optimal equilibria in compact games, and software implementations of tools for modeling and reasoning about structured games.

In Chapter 3 I present work (joint with Kevin Leyton-Brown and Navin Bhat) regarding Action-graph games (AGGs), a compact representation of complete-information simultaneous-action games first proposed by Bhat and Leyton-Brown [2004]. We make several contributions that significantly extends Bhat and Leyton-Brown's [2004] original work. First, we extended the original definition of AGGs by introducing function nodes and additive utility functions, capturing a wider variety of utility structure. The resulting AGG representation is a fully-expressive modeling language that both extends and unifies previous approaches: it can compactly express games with structure such as strict or context-specific independence, anonymity, and additivity; it can be used to compactly encode all games that are compact when represented as graphical games, symmetric games, anonymous games, congestion games, and polymatrix games, as well as additional realistic games that would take exponential space to represent using these existing representations. Second, we gave a polynomial-time algorithm for the important task of computing expected utility for AGGs, which then allows us to speed up existing normal-form-based equilibrium-finding algorithms including Govindan and Wilson's [2003] Global Newton Method and the simplicial subdivision algorithm of van der Laan et al. [1987]. Third, we implemented and made available software tools for constructing, visualizing, and reasoning with AGGs. We present results of experiments showing that using AGGs leads to a dramatic increase in the size of games accessible to computational analysis.

Pure-strategy Nash equilibrium (PSNE) is a more restricted concept that Nash equilibrium, and has certain theoretically and practically attractive properties. In Chapter 4 I present work (joint with Kevin Leyton-Brown) on computing pure-strategy Nash equilibria for AGGs. Unlike our black-box approach in Chapter 3 for computing equilibria, here we use a special-purpose approach that exploits the graph-theoretical properties of the action graph. In particular, we propose a dynamic-programming algorithm that constructs equilibria of the game from equilibria of restricted games played on subgraphs of the action graph. If the game is

symmetric and the action graph has bounded treewidth, our algorithm determines the existence of pure-strategy Nash equilibrium in polynomial time. We also extend our approach to certain classes of asymmetric AGGs. Just as AGGs unify and extend existing representations, our approach can be understood as a generalization of existing special-purpose approaches for representations including singleton congestion games [Ieong et al., 2005] and graphical games [Daskalakis and Papadimitriou, 2006, Gottlob et al., 2005].

So far we have focused on representing and reasoning with simultaneous-action games. On the other hand, many multi-agent interactions involve decisions made sequentially over time; such situations are modeled as *dynamic games* in game theory. The standard representation for dynamic games, the extensive form, is inefficient for large, structured games, while the state-of-the-art compact representation, multi-agent influence diagrams (MAIDs), only capture strict utility independence structure. In Chapter 5 I present work (joint with Kevin Leyton-Brown and Avi Pfeffer), in which we propose temporal action-graph games (TAGGs), an extension of AGGs that can compactly represent dynamic games exhibiting a wide range of structure including anonymity or context-specific utility independencies. We also show that TAGGs can be understood as indirect MAID encodings in which many deterministic chance nodes are introduced. We provide an efficient algorithm for computing expected utility for TAGGs, and show both theoretically and empirically that our approach improves significantly on MAIDs.

Games of incomplete information, or Bayesian games, are an important game-theoretic model in which players are uncertain about the utilities of the game. Despite having many applications in economics, there are relatively fewer results on the computational aspects of Bayesian games, such as compact representations and practical algorithms for computing solution concepts like Bayes-Nash equilibria. In Chapter 6 we extend AGGs to the incomplete-information setting and present Bayesian action-graph games (BAGGs), a compact representation for Bayesian games. BAGGs can represent arbitrary Bayesian games, and furthermore can compactly express Bayesian games exhibiting commonly encountered types of structure including symmetry, action- and type-specific utility independence, and probabilistic independence of type distributions. We provide an algorithm for computing expected utility in BAGGs, and discuss conditions under which the algorithm runs

in polynomial time. Sample Bayes-Nash equilibria of BAGGs can be computed by adapting existing algorithms for complete-information normal form games and leveraging our expected utility algorithm.

First proposed by Aumann [1974, 1987], correlated equilibrium (CE) is another important solution concept. In a landmark paper, Papadimitriou and Roughgarden [2008] described a polynomial-time black-box algorithm ("Ellipsoid Against Hope") for computing sample correlated equilibria of concisely-represented simultaneous-move games. Recently, Stein, Parrilo and Ozdaglar [2010] showed that this algorithm can fail to find an exact correlated equilibrium, but can be easily modified to efficiently compute approximate correlated equilibria. Currently, it remains an open problem to determine whether the algorithm can be modified to compute an exact correlated equilibrium. In Chapter 7 we show that it can, presenting a variant of the Ellipsoid Against Hope algorithm that guarantees the polynomial-time identification of exact correlated equilibrium. Also, our algorithm is the first to tractably compute correlated equilibria with polynomial-sized supports; such correlated equilibria are more natural solutions than the mixtures of product distributions produced previously, and have several advantages including requiring fewer bits to represent, being easier to sample from, and being easier to verify.

However, since in general there can be an infinite number of correlated equilibria in a game, finding an arbitrary one is of limited value. In Chapter 8 we focus on the problem of computing a correlated equilibrium that optimizes some objective (e.g., social welfare). Papadimitriou and Roughgarden [2008] gave a sufficient condition for the tractability of the problem, however it only applies to a subset of existing representations. We propose a different algorithmic approach for the optimal CE problem that applies to *all* compact representations, and give a sufficient condition that generalizes Papadimitriou and Roughgarden's condition. In particular, we reduce the optimal CE problem to the *deviation-adjusted social welfare problem*, a combinatorial optimization problem closely related to the optimal social welfare outcome problem. Our algorithm can be understood as an instance of the black-box approach, with the computation of the deviated social welfare problem as the key subroutine provided by the game representation. This framework allows us to identify new classes of games for which the optimal CE problem is tractable, including graphical polymatrix games on tree graphs. We also study the problem

of computing the optimal *coarse correlated equilibrium*, a solution concept closely related to CE. Using a similar approach we derive a sufficient condition for this problem, and use it to prove that the problem is tractable for singleton congestion games.

In Appendix A I describe software packages we implemented and made available at http://agg.cs.ubc.ca.

Taken together, this thesis presents several basic components of an algorithmic framework for computational analysis of large games: compact representations for complete-information and incomplete-information simultaneous-action games as well as dynamic games, a collection of implemented algorithms for computing sample Nash and correlated equilibria given such games, and some theoretical foundations for computing PSNE and optimal correlated equilibria. These are parts of a larger ongoing effort by our research group, that aims to apply computational game-theoretic analysis to real-world systems, especially the design and analysis of market mechanisms such as auctions. Such *mechanism design* problems have traditionally been attacked via purely analytical means, but computational analysis allows us to tackle settings for which theoretical analysis is difficult or impossible. Position auctions for advertising slots, such as the Generalized Second-Price auction used by Google AdWords, have received much recent interest from computer scientists and economists. Thompson and Leyton-Brown [2009] were able to use AGGs to compactly represent complete-information position auctions and compute their Nash equilibria, which allows them to analyze the economic properties of such auctions such as revenue and efficiency. Building on their work, I am currently working with David and Kevin to extend this analysis to incomplete-information models of position auctions using BAGGs.

Finally, I mention a couple of papers on related topics that I co-authored but do not include in this thesis. In [Jiang and Safari, 2010], Mohammad Ali Safari and I analyzed the problem of deciding the existence of pure-strategy Nash equilibria for graphical games on restricted classes of graphs, and showed that the problem is in polynomial time if and only if the class of graphs has bounded treewidth (after iterated removal of sinks). We proved our result by applying Grohe's characterization of the complexity of homomorphism problems. This result illustrated a limitation of a class of graph-based special-purpose approaches that includes the algorithm

of Chapter 4, that it cannot be extended much beyond bounded-treewidth graphs. It influenced my later focus on more general approaches such as those in Chapters 7 and 8. In [Ryan et al., 2010], Chris Ryan, Kevin Leyton-Brown and I analyzed the problem of computing pure-strategy Nash equilibria in symmetric games whose utilities are compactly represented, such that the number of players can be exponential in the representation size. We showed that if the utility functions are represented as piecewise-linear functions, there exist polynomial-time algorithms for finding a pure-strategy Nash equilibria and count the number of equilibria. Our approach made use of the rational generating function method developed by Barvinok and Woods. I do not include these papers here because they do not fit in with the focus of the thesis.

Chapter 2

A Brief Survey on the Computation of Solution Concepts

In this chapter we give a brief survey on the economics and computer science literature on the computation of game-theoretic solution concepts, focusing on Nash equilibrium and correlated equilibrium. There have been several surveys on various aspects of this topic: von Stengel [2002] focused on two-player games; McKelvey and McLennan [1996] focused on algorithms for the normal form; Papadimitriou [2007] focused on complexity results. In this survey we give emphasis to topics most relevant to this thesis, i.e., results that are relevant to large, structured games. The goal of this chapter is to present a bird's-eye view of the state of the art. We will largely follow the narrative outlined in Chapter 1. In Section 2.1 we look at representations of games and the types of structure they capture. In Section 2.2 we look at algorithmic and complexity-theoretic results, with emphasis on algorithms for compact representations. In Section 2.3 we survey software packages for game-theoretic modeling and computation.

2.1 Representations of Games

A game is a mathematical model of interaction among self-interested agents. Informally, to specify a game we need to specify a set of agents (also known as players), a set of *strategies* for each agent, and a *utility function* for each agent that assigns a utility value (also known as payoff) to each outcome of the game. Such models can be further divided into complete-information static games, incomplete-information static games and dynamic games.

A *game representation* is a data structure that stores all information needed to specify a game. An *instance* of a representation is a game encoded in that representation. Thus it is often useful to think of a game representation as a *class* (or *type*) in the language of object-oriented software engineering, and an instance of a representation as an *object* in that class. Then the *size* of a representation is the amount of data required to specify a game instance (i.e., initialize an object) of that representation.

In this section we survey the existing literature on representing games. Section 2.1.1 focuses on representing complete-information static games; Section 2.1.2 focuses on representing dynamic games; Section 2.1.3 focuses on representing incomplete-information games.

2.1.1 Representing Complete-information Static Games

In static games, also known as simultaneous-move games, each agent chooses a strategy simultaneously (e.g., Rock-Paper-Scissors). By *complete-information* we mean that each agent knows the utility functions of all agents.

Definition 2.1.1. *A* complete-information static game *is a tuple* $(N, \{A_i\}_{i\in N}, \{u_i\}_{i\in N})$ *where*

- $N = \{1,\ldots,n\}$ *is the set of agents;*
- *for each agent i,* A_i *is the nonempty set of i's actions (or pure strategies). We denote by* $a_i \in A_i$ *one of agent i's actions. An action profile (or pure-strategy profile)* $a = (\alpha_1,\ldots,\alpha_n) \in \prod_{i\in N} A_i$ *is a tuple of actions of the n agents. We also denote by* a_{-i} *the* $(n-1)$*-tuple of actions by agents other than i under*

the action profile a.[1]

- $u_i : \prod_{j \in N} A_j \to \mathbb{R}$ *is i's utility function, which specifies i's utility given any action profile.*

A game representation is *fully expressive* if it can represent arbitrary games. We say a game representation has *polynomial type* [Daskalakis et al., 2006a] if the number of players and the number of actions for each player are bounded by polynomials of the representation size. For example, if the set of players and the sets of actions are encoded explicitly, then the representation has polynomial type. This is the case for all representations of static games discussed in this section.

Normal Form

A *normal form* representation of a game uses a multi-dimensional matrix $U_i \in \mathbb{R}^{\prod_{j \in N} A_j}$ to represent each utility function u_i. The size of this representation is approximately $n \prod_{j \in N} |A_j|$, which is $O(nm^n)$ where $m = \max_{i \in N} |A_i|$. Two-player normal-form games are also called bimatrix games, since the utility functions of such a game can be specified by two $A_1 \times A_2$ matrices.

Although these games are fully expressive, the size of the representation grows exponentially in the number of players. As a result, the normal form is unsuitable for representing large systems. Although several computational tasks such as finding pure Nash equilibria and computing expected payoff under mixed strategies are polynomial-time in the size of the normal form representation, they are intractable for large games because the representation size itself is exponential.

Graphical Games

Fortunately, most real-world large games have structure that allows them to be represented compactly. A popular compact representation of games is *graphical games*, proposed by Kearns et al. [2001]. A game is associated with a graph whose

[1] While in complete-information static games the concepts of actions and pure strategies coincide, we will see that this is no longer the case for incomplete-information games and dynamic games. For the cases when pure strategies are distinct from actions, we denote pure strategies by $s_i \in S_i$ and pure-strategy profiles by $s \in S$. For complete-information static games, both the a-based notation and the s-based notation are commonly used in the literature to denote pure strategies/actions [e.g., Fudenberg and Tirole, 1991, Shoham and Leyton-Brown, 2009].

nodes correspond to the players of the game and edges correspond to payoff influence between players. In other words, each player's payoffs depend only on his actions and those of his neighbors in the graph. We call this kind of structure *strict utility independence*.

Definition 2.1.2. *A* graphical game *is a tuple* $(G, \{U_i\}_{i \in N})$ *where*

- $G = (N, E)$ *is a directed graph,*[2] *with the set of vertices corresponding to the set of agents.* E *is a set of ordered tuples corresponding to the arcs of the graph, i.e.* $(i, j) \in E$ *means there is an arc from* i *to* j*. Vertex* j *is a* neighbor *of* i *if* $(j, i) \in E$*.*
- *for each* $i \in N$*,* a local utility function $U_i : \prod_{j \in \nu(i)} A_j \to \mathbb{R}$ *where* $\nu(i) = \{i\} \cup \{j \in N | (j, i) \in E\}$ *is the* neighborhood *of* i*.*

Each local utility function U_i is represented as a matrix of size $\prod_{j \in \nu(i)} |A_j|$. Since the size of the local utility functions dominates the size of the graph G, the total size of the representation is $O(nm^{(\mathscr{I}+1)})$ where $\mathscr{I}$ is the maximum in-degree of G.

A graphical game $(G, \{U_i\})$ specifies a game $(N, \{A_i\}, \{u_i\})$ where each A_i is specified by the domain of agent i in U_i, and for all $i \in N$ and all action profiles a we have $u_i(\mathbf{s}) \equiv U_i(a_{\nu(i)})$, where $a_{\nu(i)} = (a_j)_{j \in \nu(i)}$. Graphical games are fully expressive: an arbitrary game can be represented as a graphical game on a complete graph.

Symmetric Games and Anonymous Games

A game is *symmetric* when all players are identical and interchangeable. Formally, a game is symmetric if each player has an identical set of actions and for all permutation of players $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$,

$$u_i(a_1, \ldots, a_n) = u_{\pi(i)}(a_{\pi(1)}, \ldots, a_{\pi(n)}).$$

[2] Kearns et al. [2001] originally defined graphical games on undirected graphs, while some later authors [e.g., Daskalakis and Papadimitriou, 2006, Gottlob et al., 2005] used the directed graph version given here. A undirected graphical game is equivalent to a directed graphical game in which each edge $\{i, j\}$ from the undirected graph is replaced by two directed edges (i, j) and (j, i). Thus the directed graph version is more general.

Symmetric games have been studied since the beginning of noncooperative game theory. For example, Nash proved that symmetric games always have symmetric mixed Nash equilibria [Nash, 1951]. In a symmetric game, a player's utility depends only on the player's chosen action and the *configuration*, which is the vector of integers specifying the numbers of players choosing each of the actions. We say such a utility function exhibits *anonymity*. As a result, symmetric games can be represented more compactly than the normal form: we only need to specify a utility value for each action and each configuration. For a symmetric game with n players and m actions per player, the number of configurations is $\binom{n+m-1}{m-1}$. For fixed m, this grows like n^{m-1}, in which case $\Theta(n^{m-1})$ numbers are required to specify the game.

A straightforward generalization of symmetric games is k-symmetric games, in which there are k equivalence classes of players. Nash's [1951] result applies to a very general notion of symmetry: roughly, if a game is invariant under a permutation group, then there exists a Nash equilibrium strategy profile that is invariant under the same group. Specialized to k-symmetric games, it implies that they always have k-symmetric Nash equilibria, where strategies within each class are identical. Any game is a k-symmetric game with $k = n$. On the other hand, when k is small compared to n, k-symmetric games can be compactly represented by specifying utilities for each k-configuration, where a k-configuration is a tuple of k configurations, one for each equivalence class.

There has also been research [e.g., Brandt et al., 2009, Daskalakis and Papadimitriou, 2007] on a generalization of symmetric games called *anonymous games*, in which a given player's utility depends on his identity as well as the action chosen and the configuration. Anonymous games can be compactly represented in a similar manner, requiring $\Theta(n^m)$ numbers for fixed m.

Polymatrix Games

Polymatrix games are a class of games in which each player's utility is the sum of utilities resulting from her bilateral interactions with each of the $n-1$ other players. This can be represented by specifying for each pair of players i and j a bimatrix game (two-player normal form game) with sets of actions A_i and A_j.

When a utility function can be expressed as a sum of other functions, as in polymatrix games, we say it exhibits *additive* structure.

Congestion Games

A congestion game [Rosenthal, 1973] is a tuple $(N, M, (A_i)_{i\in N}, (K_{jk})_{j\in M, k\leq n})$, where $N = \{1,\ldots,n\}$ is the set of players, $M = \{1,\ldots,m\}$ is a set of facilities (or resources); A_i is player i's set of actions; each action $a_i \in A_i$ is a subset of the facilities: $a_i \subset M$. K_{jk} is the cost of using facility j when a total of k players have chosen actions that include facility j. For notational convenience we also define $K_j(k) \equiv K_{jk}$. Let $\#(j,a)$ be the number of players that chose facility j given the action profile a. The total cost (or disutility) of player i under pure strategy profile $a = (a_i, a_{-i})$ is the sum of the costs on each of the facilities in a_i,

$$Cost_i(a_i, a_{-i}) = -u_i(a_i, a_{-i}) = \sum_{j\in a_i} K_j(\#(j,a)). \tag{2.1.1}$$

Only nm numbers are needed to specify the costs $(K_{jk})_{j\in M,k\leq n}$. The representation also needs to specify the $\sum_{i\in N}|A_i|$ actions, each of which is a subset of M. If we use an m-bit binary string to represent each of these subsets, the total size of the congestion game representation is $O(mn + m\sum_{i\in N}|A_i|)$.

From the above definition we can see that congestion games exhibit a specific combination of anonymity and additive structure, plus a type of utility independence which we call *context-specific independence (CSI)*. This means that the independence structure of player i's utility function (i.e., which subset of players that affect player i's utility) changes depending on the *context*, which is a certain feature of the players' strategies (in this case the facilities included in i's chosen action). This is a more general type of independence structure than the strict independencies captured by graphical games. On the other hand, congestion games are not fully expressive.

Local Effect Games

Local Effect Games (LEGs), proposed by Leyton-Brown and Tennenholtz [2003], were the first graphical representation of games that focused on actions. In an LEG,

we have a graph whose nodes correspond to the actions of the game. Each player can choose any one of the nodes. Define configuration as in symmetric games, and let the configuration over node k, denoted $c(k)$, be the number of players choosing node k. There is a node function U_k associated with each node k which maps the configuration of node k to a real number. There is an edge function $U_{k,m}$ associated with each edge (k,m) of the graph, which maps the configuration over nodes k and m to a real number. The utility of a player i choosing node k is the sum of the node function U_k and all incoming edge functions, evaluated at the current configuration c:

$$U_k(c(k)) + \sum_{m \in v(k)} U_{m,k}(c(m), c(k)).$$

Like congestion games, LEGs also exhibit a combination of anonymity, additivity and context-specific independence structure. In this case the *context* for player i's utility independence is the *action* chosen by i. We call such structure action-specific independence. Unfortunately, like congestion games, LEGs are also not fully expressive.

Action-Graph Games

We have seen representations that capture various types of structure such as strict and context-specific independence, anonymity, and additivity. However, the existing representations either only capture a subset of these types of structure (graphical games, symmetric/anonymous games, polymatrix games), or are only able to represent a subset of games (symmetric/anonymous games, polymatrix games, congestion games, local-effect games).

Action-graph games (AGGs), proposed by Bhat and Leyton-Brown [2004] and extended by Jiang et al. [2011], are a compact representation of simultaneous-move games that extends and unifies these previous approaches. AGGs are fully expressive (able to represent arbitrary games), can compactly express games whose utility functions exhibit action-specific independence, anonymity or additivity, and furthermore have nice computational properties. Chapter 3 gives a detailed discussion of AGGs.

2.1.2 Representing Dynamic Games

In *dynamic games*, agents move sequentially. When agents are able to perfectly observe all moves, dynamic games are said to exhibit *perfect information*; otherwise, dynamic games exhibit *imperfect information*.

The standard representation for dynamic games is the *extensive form*, which is a tree whose edges represent moves of players. Thus each node of the tree corresponds to a unique sequence of moves. Utilities for all players are specified for each leaf of the tree. Each internal node is assigned to a player, who can choose among the edges below that node. Imperfect information is specified using *information sets*: each player's set of internal nodes is partitioned into information sets, and a player is unable to distinguish nodes in any of his information sets. Randomness in the environment can be represented as nodes for the Nature (also known as Chance) player, who randomizes over his actions according to some fixed distribution. See e.g., [Shoham and Leyton-Brown, 2009] for a formal definition of the extensive form.

Each extensive-form game can be transformed to an *induced normal form*, where each pure strategy of a player prescribes an action for each of her information sets. The number of pure strategies can be exponential in the size of the extensive form, so transforming to the induced normal form entails an exponential blowup in representation size. In this sense the extensive form can be seen as a compact representation of dynamic games. However, this representation requires us to specify utilities for every possible sequence of moves; when the game exhibits more structure than this, a more compact representation is needed.

For imperfect-information dynamic games, the most influential compact representation is multiagent influence diagrams (MAIDs) [Koller and Milch, 2003], which generalize single-agent influence diagrams to multiple agents. A MAID is represented as a directed graph, consisting of decision nodes, chance nodes and utility nodes. Each chance node corresponds to a random variable, with its domain and its probability distribution conditioned on its parents (nodes with incoming edges) specified by input. Each decision node represents a decision (over a finite number of choices) taken by some player, given her observations which are the instantiated values of the decision node's parents. Each utility node represents the payoff to

some player, as a function of the instantiated values of the node's parents. MAIDs are compact when players' utility functions exhibit strict independencies, but are unable to compactly represent utility functions with anonymity or action-specific independencies.

In Chapter 5 we discuss temporal action-graph games (TAGGs), which are a generalization of AGGs to the dynamic setting, and are able to compactly represent dynamic games with anonymity or context-specific utility independencies.

2.1.3 Representing Games of Incomplete Information

In many multi-agent situations, players are uncertain about the game being played. Harsanyi [1967] proposed games of incomplete information (or Bayesian games) as a mathematical model of such interactions.

Definition 2.1.3. *A Bayesian game is a tuple* $(N, \{A_i\}_{i\in N}, \Theta, P, \{u_i\}_{i\in N})$ *where* $N = \{1,\ldots,n\}$ *is the set of players; each* A_i *is player* i*'s action set, and* $A = \prod_i A_i$ *is the set of action profiles;* $\Theta = \prod_i \Theta_i$ *is the set of type profiles, where* Θ_i *is player* i*'s set of types;* $P : \Theta \to \mathbb{R}$ *is the type distribution and* $u_i : A \times \Theta \to \mathbb{R}$ *is the utility function for player* i*.*

As in the complete-information case, we denote by a_i an element of A_i, and $a = (a_1,\ldots,a_n)$ an action profile. Furthermore we denote by θ_i an element of Θ_i, and by θ a type profile.

The game is played as follows. A type profile $\theta = (\theta_1,\ldots,\theta_n) \in \Theta$ is drawn according to the distribution P. Each player i observes her type θ_i and, based on this observation, chooses from her set of actions A_i. Each player i's utility is then given by $u_i(a,\theta)$, where a is the resulting action profile. Intuitively player i's type represents her private information about the game.

Bayesian games can be encoded as dynamic games with an initial move by Nature. Thus dynamic game representations such as the extensive form can be used to represent Bayesian games. This is also why we do not discuss dynamic games of incomplete information here, as they can also be encoded using existing dynamic game representations. However, incomplete-information static games do have independent interest apart from their dynamic game interpretation, as they are more similar to complete-information static games than to dynamic games.

In specifying a Bayesian game, the space bottlenecks are the type distribution and the utility functions. Without additional structure, we cannot do better than representing each utility function $u_i : A \times \Theta \to R$ as a table and the type distribution as a table as well. We call this representation the *Bayesian normal form*. The size of this representation is $n \times \prod_{i=1}^{n}(|\Theta_i| \times |A_i|) + \prod_{i=1}^{n}|\Theta_i|$.

A Bayesian game can be converted to its *induced normal form*, which is a complete-information game with the same set of n players, in which each player's set of actions is her set of pure strategies in the Bayesian game. Each player's utility under an action profile is defined to be equal to the player's expected utility under the corresponding pure strategy profile in the Bayesian game. Alternatively, a Bayesian game can be transformed to its *agent form*, where each type of each player in the Bayesian game is turned into one player in a complete-information game. The sizes of the normal forms for the two complete-information interpretations are both exponential in the size of the Bayesian normal form.

Singh et al. [2004] proposed a incomplete information version of the graphical game representation. Gottlob et al. [2007] considered a similar extension of the graphical game representation. Like graphical games, such representations are limited in that they can only exploit *strict* utility independencies.

In Chapter 6 we discuss Bayesian Action-Graph Games (BAGGs), a fully-expressive compact representation for Bayesian games that can compactly express Bayesian games exhibiting commonly encountered types of structure including symmetry, action- and type-specific utility independence, and probabilistic independence of type distributions.

2.2 Computation of Game-theoretic Solution Concepts

Being able to compactly represent structured games is necessary, but often not sufficient for our purposes. We would like to efficiently reason about these games, by computing game-theoretic solution concepts such as Nash equilibrium and correlated equilibrium.

2.2.1 Computing Sample Nash Equilibria for Normal-Form Games

In this subsection, we survey the literature on computing Nash equilibria in games represented in normal form. We start with the definition of Nash equilibrium and some theoretical results on the complexity of finding a sample Nash equilibrium, then look at existing algorithms, focusing on approaches for games with more than two players. In summary, the problem of computing one Nash equilibrium is PPAD-complete: polynomial time algorithms are unlikely to exist. Unsurprisingly, existing approaches all require exponential time in the size of the normal form.

In a simultaneous-move game, a player i plays a *pure strategy* when she deterministically chooses an action from her action set A_i. She can also randomize over her actions, in which case we say that she plays a *mixed strategy*. Formally, let $\varphi(X)$ denote the set of all probability distributions over a set X. Define the set of mixed strategies for i as $\Sigma_i \equiv \varphi(A_i)$; then a mixed strategy $\sigma_i \in \Sigma_i$ is a probability distribution over A_i. Define the set of all mixed strategy profiles as $\Sigma \equiv \prod_{i \in N} \Sigma_i$; then a mixed strategy profile $\sigma \in \Sigma$ is a tuple of the n players' mixed strategies. The *expected utility* (also known as expected payoff) of player i under the mixed strategy profile σ, denote by $u_i(\sigma)$, is

$$u_i(\sigma) = \sum_{a \in A} u_i(a) \prod_{j \in N} \sigma_j(a_i), \tag{2.2.1}$$

where $\sigma_i(a_i)$ denotes the probability that i plays a_i. The *support* of a mixed strategy σ_j is the set of actions with positive probability under the distribution σ_j. A *support profile* is a tuple of all players' supports. Given σ_{-i}, a tuple of mixed strategies of players other than i, we define the best response set of i to be the set of i's mixed strategies that maximize her expected utility:

$$BR_i(\sigma_{-i}) = \arg\max_{\sigma_i} u_i(\sigma_i, \sigma_{-i})$$

Given σ_{-i}, the expected utility of i playing mixed strategy σ_i is a convex combination of the expected utilities of playing pure strategies in A_i, so at least one of the pure strategies must be a best response. Thus to check whether σ_i is a best response, we just need to compare its expected utility against the expected utilities

of playing each of i's pure strategies.

One of the central solution concepts in game theory is Nash equilibrium.

Definition 2.2.1 (Nash Equilibrium)**.** *A mixed strategy profile σ is a Nash equilibrium if for all $i \in N$, $\sigma_i \in BR_i(\sigma_{-i})$.*

Intuitively, a Nash equilibrium is strategically stable: no player can profit by unilaterally deviating from her current mixed strategy. From the above discussion on best response, an equivalent condition for Nash equilibrium is that for all $i \in N$, for all $a_i \in A_i$, $u_i(\sigma) \geq u_i(a_i, \sigma_{-i})$, where by a slight abuse of notation, we denote by (a_i, σ_{-i}) the mixed strategy profile where i plays pure strategy a_i and other players play according to σ.

One of the most famous results in game theory is Nash's proof that any finite game always has a Nash equilibrium [Nash, 1951]. For a tutorial on Nash's proof (as well as a derivation of Brouwer's fixed-point theorem, which is used by his proof), see [Jiang and Leyton-Brown, 2007b].

Although a Nash equilibrium always exists, the existence proofs do not give an efficient algorithm for finding one. The central computational problem we consider here is the problem of finding a *sample Nash equilibrium*:

Problem 2.2.2 (NASH)**.** *Given a game represented in normal form, find one Nash equilibrium.*

McKelvey and McLennan [1996] showed that this problem can be formulated as instances of other of computational problems, e.g.,

- finding a fixed point of a continuous function;
- finding a global minimum of a continuous function;
- solving a system of polynomial equations and inequalities.

A frequently-used notion of approximation for Nash equilibrium is the so-called ε-Nash equilibrium:

Definition 2.2.3 (ε-Nash Equilibrium)**.** *A mixed strategy profile σ is an ε-Nash equilibrium for some $\varepsilon \geq 0$ if for all $i \in N$, for all $a_i \in A_i$, $u_i(\sigma) + \varepsilon \geq u_i(a_i, \sigma_{-i})$.*

Intuitively, each player cannot gain more than ε by deviating from her mixed strategy. When $\varepsilon = 0$, we recover Nash equilibrium.

Complexity

The NASH problem is different from decision problems studied in complexity theory (e.g. SAT), which have a yes/no answer. Since a Nash equilibrium always exists, the decision problem asking about the existence of Nash equilibrium can be solved by a trivial algorithm that always returns "yes". Instead, we are interested in finding a Nash equilibrium. This is an example of a *function problem*, which requires more complex answers than yes/no. Because we can check whether a given mixed strategy profile is a Nash equilibrium by computing expected utilities, the NASH problem is in FNP, the function problem version of NP. In fact it belongs to TFNP, the class of FNP problems whose solutions are guaranteed to exist.

Another issue is that a Nash equilibrium for a game of more that two players may require irrational numbers in the probabilities, even if the game itself involves only rational payoffs. It is impossible to represent such a solution exactly using floating point numbers. Instead, in such cases we look for algorithms that given a game and an error tolerance ε represented in binary, computes an ε-Nash equilibrium. As always, we evaluate complexity as a function of the input size, which here includes ε.

A recent series of papers [Chen and Deng, 2006, Daskalakis et al., 2006b, Goldberg and Papadimitriou, 2006] established that the NASH problem is PPAD-complete for normal form games, even if the game has only two players. The complexity class PPAD, introduced by Papadimitriou [1994], stands for Polynomial Parity Argument (Directed version). It is the class of TFNP problems whose solutions are guaranteed by a parity argument. It is widely believed that PPAD-complete problems are unlikely to be in P [e.g., Papdimitriou, 2007].

Although any Nash equilibrium is close to an ε-Nash equilibrium (in the space of mixed strategy profiles), a given ε-Nash equilibrium may be arbitrarily far from any Nash equilibrium of the game. Etessami and Yannakakis [2007] studied the complexity of the problem of finding an ε-Nash equilibrium close to some exact Nash equilibrium. They showed that the problem is at least as hard as the square-root sum problem, which is not known even to belong to NP.

Algorithms for Two-Player Games

A two-player game is *zero-sum* if for all action profiles a, we have $u_1(a) + u_2(a) = 0$. For zero-sum games, Nash equilibria can be computed in polynomial time by linear programming (see, e.g., [Shoham and Leyton-Brown, 2009, von Neumann and Morgenstern, 1944]).

For general two-player games, the NASH problem can be formulated as a linear complementarity problem (LCP). The canonical method for solving such games is the Lemke-Howson Algorithm [Lemke and Howson, 1964]. Sets of labels are assigned to mixed-strategy profiles and Nash equilibria are characterized as "completely-labeled" mixed-strategy profiles. The algorithm uses pivoting techniques that are similar to the Simplex Algorithm to trace a path that ends at a completely-labeled point (i.e., Nash equilibrium). It is guaranteed to find a Nash equilibrium but in the worst case may require exponential time [Savani and von Stengel, 2004]. Lemke's algorithm [Lemke, 1965] is a related method that uses similar pivoting techniques.

Lipton et al. [2003] used the probabilistic method to show that for any two-player game, there always exists an ε-equilibrium with log-sized support. Their result implies a quasi-polynomial algorithm for finding an ε-equilibrium.

Another interesting property of two-player games is that if both of the payoff matrices have small rank (say k), then there exists a Nash equilibrium with small (size k) support. Such a Nash equilibrium can be found efficiently by going through the small-sized support profiles. This was discussed by Lipton et al. [2003], but they mentioned that the result was known earlier.

For bimatrix games whose entry-wise sum of the two matrices have small rank, Kannan and Theobald [2009] proposed a polynomial time algorithm for finding approximate Nash equilibria. More recently, Adsul et al. [2011] showed that if the rank of the sum of the two matrices is 1, a Nash equilibrium can be computed in polynomial time.

Fictitious Play

We now focus on algorithms for n-player games, where $n > 2$. We start with Fictitious Play [e.g., Brown, 1951, Shoham and Leyton-Brown, 2009], well-known

in the study of learning in games but can also be used as an algorithm for finding Nash equilibria. It is an iterative process; at each step, each player i plays a best response assuming each of the other players j chooses a mixed strategy corresponding to the empirical distribution of j's past actions. For certain classes of games (e.g., zero-sum games and potential games) the empirical distribution of this process converges to a Nash equilibrium. However it is not guaranteed to converge for all games, hence it is only a heuristic for general games.

Simplicial Subdivision

One influential class of algorithms for computing Nash equilibria in n-player games are *simplicial subdivision* algorithms, which are based on Scarf's algorithm [1967]. A modern version is due to van der Laan, Talman & van der Heyden [1987]. In a high level, the algorithm does the following:

1. The space of mixed strategy profiles $\Sigma = \prod_i \Sigma_i$ is partitioned into a set of subsimplexes.

2. We assign labels to vertices of the subsimplexes, in a way such that a "completely labeled" subsimplex corresponds to an approximate Nash equilibrium.

3. The algorithm follows a path of "almost completely labeled" subsimplexes, and eventually reaches a "completely labeled" subsimplex.

4. The approximate equilibrium is refined by restarting the algorithm near the approximate equilibrium, but using a finer grid.

It can be proven (using Sperner's Lemma) that the algorithm will always find an ε-equilibrium for any given ε. However the running time is exponential. In particular, the path could go through an exponential number of subsimplexes. Within each step of the path, one of the computational bottlenecks is computation of labels of the subsimplex. The computation of labels in turn depends on computation of expected utilities under mixed strategy profiles.

Function Minimization

McKelvey and McLennan [1996] discussed formulating Nash equilibria as solutions of a function minimization problem. Given mixed strategy profile σ, let $g_{ij}(\sigma)$ be the amount player i could gain by deviating to action j (and 0 if j is worse). A Nash equilibrium then corresponds to a global minimum of the function

$$v(\sigma) = \sum_i \sum_j [g_{ij}(\sigma)]^2,$$

subject to σ being a mixed strategy profile.

Note that the global minimum of $v(\sigma)$ is always 0, due to the existence of Nash equilibria. Standard function minimization techniques can then be applied. In order to find a global minimum, a good starting point is essential. According to McKelvey and McLennan [1996], this approach is "generally slower than other methods".

Homotopy Methods and the Global Newton Method

At a high level, a homotopy method starts with a game that has a simple solution, then continuously deforms the payoffs of the game, until it ends at the original game of interest. Meanwhile, the method traces the path of Nash equilibria for these games, starting at a Nash equilibrium of the simple game and ending at a Nash equilibrium of the game of interest. Several homotopy methods for computing Nash equilibria have been proposed (a recent survey is [Herings and Peeters, 2009]). One such approach is Govindan and Wilson's [2003] global Newton method (also known as continuation method [e.g., Blum et al., 2006]), which can be thought of as a generalization of the Lemke-Howson algorithm to the n-player case. It starts at a deformed game where one action per player is given a large bonus, such that there exists a unique equilibrium. At each iteration, it computes the direction of next step by following a gradient. Since the path is nonlinear, the algorithm needs to periodically correct accumulated error using a local Newton method.

One implementation of the algorithm is available in GameTracer [Blum et al., 2002]. The bottleneck of each iteration is the computation of the so-called payoff

Jacobian matrix given a mixed strategy profile. Entries of the Jacobian correspond to the expected utility of player i when i plays action a, player i' plays action a', and all other players play according to the given mixed strategy profile.

Iterated Polymatrix Approximation

Iterated Polymatrix Approximation is another algorithm proposed by Govindan and Wilson [2004]. At a high level, the algorithm can be summarized as follows.

1. Start at some strategy profile σ^0.
2. Consider the problem linearized at σ^0: we get a polymatrix game, which (as we will see in Section 2.2.2) can be solved using a variant of the the Lemke-Howson algorithm, to find equilibrium σ^1. The payoffs of the polymatrix game correspond to entries of the payoff Jacobian.
3. Repeat with starting point σ^1.

If this process converges, it converges to a Nash equilibrium. However the algorithm is not guaranteed to converge. Thus, like fictitious play, this belongs to the category of heuristics. In cases of non-convergence, the authors propose using the result of the algorithm as a starting point for the Govindan-Wilson global Newton method.

Support Enumeration

Porter et al. [2008] proposed an algorithm that finds Nash equilibria by searching through support profiles. The algorithm can be summarized as follows.

1. Enumerate all support profiles, starting with small support sizes
2. Given a support profile, determine whether there exists a Nash equilibrium having that support profile.
    - For 2-player games, this involves solving a linear feasibility program.

- For n-player games, this involves solving a system of polynomial equations and inequalities[3] of degree $n-1$.

3. Stop when one equilibrium is found.

Since the number of possible support profiles is exponential in the size of the normal form, and for n-player games step 2 requires exponential time, the above algorithm has exponential worst-case complexity. Nevertheless, the motivation behind the algorithm is the observation that many games have small-support Nash equilibria. When such equilibria exist, the algorithm can quickly find them.

Another effective speedup Porter et al.'s algorithm employs is to prune off support profiles by eliminating dominated strategies conditioned on the current support profile.

2.2.2 Computing Sample Nash Equilibria for Compact Representations of Static Games

So far we have focused on the NASH problem for normal form games. In this section we give an overview of literature on the computation of Nash equilibria under compact representations. Overall, we will see that (1) for many representations the NASH problem is in PPAD, and is PPAD-complete for fully-expressive representations, and (2) algorithms for the NASH problem can roughly be divided into two categories, "black-box" approaches which treat the representation as a black box, and "special-purpose" approaches which are representation-specific algorithms that exploit the structure of the representation such as symmetry and graph-theoretic properties.

[3] One may wonder why not just solve the system of polynomial equations and inequalities characterizing the Nash equilibria of the game (see Section 2.2.1). There are two reasons one might prefer to solve the support-profile-specific system here: (1) for small support profiles, the resulting systems are much smaller; (2) it is known that for generic games, the solution set of a support-profile-specific system minus all the inequality constraints has dimension zero, i.e., it consists of isolated points. This means one method for solving this system is to solve the system minus all the inequality constraints (which is a system of polynomial equations), then check the solutions against the inequality constraints. Compared to the problem of solving systems of polynomial equations and inequalities, a wider variety of algorithms are available for solving polynomial equations, including ones based on (complex) algebraic geometry such as Groebner basis methods and polynomial homotopy continuation methods.

Complexity

Fully-expressive game representations such as graphical games and AGGs can encode arbitrary normal form games. Therefore finding Nash equilibria for these representations is PPAD-hard. In other words, polynomial time algorithms are unlikely to exist.

On the other hand, Daskalakis et al. [2006a] proved the following result:

Theorem 2.2.4 ([Daskalakis et al., 2006a])**.** *If a game representation satisfies the following properties: (1) the representation has* polynomial type *(defined in Section 2.1.1), and (2) expected utility can be computed using an arithmetic binary circuit with polynomial length, with nodes evaluating to constant values or performing addition, substraction, or multiplication on their inputs, then the NASH problem for this representation can be polynomially reduced to the NASH problem for some two-player, normal-form game.*

Since the NASH problem is in PPAD for two-player, normal-form games, the theorem implies that if the above properties hold, the NASH problem for such a compact game representation is in PPAD. Many of the existing representations satisfy these conditions. This is a positive result: since the NASH problems for such a compact representation reduces to NASH for a two-player game with size polynomial in the size of the compact representation, solving such a two-player game can be much easier than solving the normal form of the original game.

The above result suggests that the computation of expected utility is of fundamental importance for the NASH problem. Another example of its importance is the observation that if we can compute expected utilities, we can verify a solution of the NASH problem. We will see more useful applications of expected utility computation throughout this survey.

Speeding up Existing Algorithms and the Black-box Approach

Quite a few of the existing algorithms for finding Nash equilibria of normal form games use computation of expected utility as a subroutine. Examples include Govindan and Wilson's Global Newton Method and Iterated Polymatrix Approximation, as well as the simplicial subdivision algorithm.

For many compact representations (including all compact representations introduced in Section 2.1.1), there exist efficient algorithms for computing expected utility that scale polynomially in the representation size [e.g., Papadimitriou and Roughgarden, 2008]. Using these methods instead of normal-form-based methods for the expected utility subroutine, we can achieve exponential speedup of these existing Nash equilibrium algorithms without introducing any change in the algorithms' behavior or output. Blum et al. [2006] were the first to propose such an approach, speeding up Govindan and Wilson's algorithms [2003, 2004] for graphical games and MAIDs. In Chapter 3 we discuss our work on speeding up Govindan and Wilson's Global Newton Method and the simplicial subdivision algorithm for AGGs.

From a software-engineering point of view, such algorithms have a nice modular structure: an algorithm calls certain subroutines provided by the representation that access information about the game, but is otherwise unaware of the internal structure of the representation. At the same time, the representation-specific subroutines do not need to know about the details of the calling algorithm. We call such algorithms *black-box* algorithms.

Another example of the black-box approach is the very recent adaptation of the support-enumeration approach to AGGs and graphical games [Thompson et al., 2011]. Here there are several required subroutines; one is the formulation of the polynomial system given a support profile. The polynomial system contains expressions for expected utilities, the construction of which can be thought of as symbolic computation of expected utilities. Many techniques for the expected utility problem in compact games translate to the symbolic problem. Another subroutine is the elimination of dominated strategies conditioned on a support profile.

The black-box approach is not limited to the problem of computing a sample Nash equilibrium. For example, in Section 2.2.7 we look at Papadimitriou and Roughgarden's [2008] algorithm for the problem of computing a correlated equilibrium, which requires a polynomial-time expected utility subroutine. This is also an example of a black-box algorithm that isn't a direct adaptation of an existing algorithm for the normal form.

On the other hand, specific representations may exhibit certain structure that can be exploited for efficient computation. We call these representation-specific al-

gorithms *special-purpose* algorithms. Intuitively, black-box algorithms and special-purpose algorithms both exploit the compact representation's structure, albeit at different levels: a black-box algorithm exploits structure to speed up a subroutine of the algorithm, keeping the rest of the algorithm intact across different representations, while in a special-purpose approach the entire algorithm is designed with a specific representation in mind. We now go through several representations and their corresponding special-purpose algorithms.

Polymatrix Games

Yanovskaya [1968] showed that Nash equilibria of a polymatrix game are solutions of an LCP. Such equilibria can be computed using a variant of the Lemke Howson algorithm [Howson Jr, 1972].

Symmetric Games

As mentioned in Section 2.1.1, Nash [1951] proved that any symmetric game always has a symmetric Nash equilibrium. The space of symmetric strategy profiles has lower dimension than the space of mixed strategy profiles, so one might expect the problem of finding symmetric Nash equilibria to be easier than NASH in the general case.

Gale et al. [1950] showed that NASH for bimatrix games can be reduced to finding a symmetric Nash equilibrium for symmetric bimatrix games. Therefore, the recent PPAD-completeness result for bimatrix games implies that finding symmetric Nash is also PPAD-complete.

On the other hand, for symmetric games with a large number of players but a small number of actions, Papadimitriou and Roughgarden [2005] proposed a polynomial-time algorithm for finding a symmetric Nash equilibrium. The algorithm is based on the enumeration of all symmetric support profiles and the solution of a polynomial system for each support profile.

Anonymous Games

For anonymous games, the existence of symmetric equilibria is no longer guaranteed. Thus the above algorithm for symmetric games with a small number of

actions does not apply. Nevertheless, in a series of papers Daskalakis and Papadimitriou [2007, 2008, 2009] proposed polynomial-time algorithms for finding approximate Nash equilibria for anonymous games having a constant number of actions per player.

Graphical Games

Kearns et al. [2001] presented a polynomial-time algorithm for finding approximate Nash equilibrium in graphical games on tree graphs. The algorithm is based on a discretization of the mixed strategy space and a message-passing approach similar to probabilistic inference algorithms for Bayesian networks. For computing approximate Nash equilibria in graphical games on general graphs, Ortiz and Kearns [2003] and Vickrey and Koller [2002] proposed several approaches based on similar ideas.

Elkind et al. [2006] presented a polynomial-time algorithm for finding exact Nash equilibria for graphical games on path graphs. The problem of finding exact Nash for tree graphs is still open.

Symmetric AGGs

Besides the black-box algorithms that we discuss in Chapter 3, Daskalakis et al. [2009] presented a polynomial-time *special-purpose* algorithm for finding an approximate symmetric Nash equilibrium in symmetric AGGs on tree graphs. Their algorithm is based on a discretization of the space of symmetric mixed strategies and a message-passing/dynamic programming approach.

2.2.3 Computing Sample Bayes-Nash Equilibria for Incomplete-information Static Games

Bayes-Nash equilibrium is a solution concept for Bayesian games that is analogous to Nash equilibrium for complete-information games. Before we give its definition we first need to define strategies in Bayesian games. In a Bayesian game, player i can deterministically choose a *pure strategy* s_i, in which given each $\theta_i \in \Theta_i$ she deterministically chooses an action $s_i(\theta_i)$. Player i can also randomize and play a *mixed strategy* σ_i, in which her probability of choosing a_i given θ_i is $\sigma_i(a_i|\theta_i)$.

That is, given a type $\theta_i \in \Theta_i$, she plays according to distribution $\sigma_i(\cdot|\theta_i)$ over her set of actions A_i. A mixed strategy profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a tuple of the players' mixed strategies.

The *expected utility* of i given θ_i under a mixed strategy profile σ is the expected value of i's utility under the resulting joint distribution of a and θ, conditioned on i receiving type θ_i:

$$u_i(\sigma|\theta_i) = \sum_{\theta_{-i}} P(\theta_{-i}|\theta_i) \sum_a u_i(a,\theta) \prod_j \sigma_j(a_j|\theta_j). \tag{2.2.2}$$

A mixed strategy profile σ is a *Bayes-Nash equilibrium* if for all i, for all θ_i, for all $a_i \in A_i$, $u_i(\sigma|\theta_i) \geq u_i(\sigma^{\theta_i \to a_i}|\theta_i)$, where $\sigma^{\theta_i \to a_i}$ is the mixed strategy profile that is identical to σ except that i plays a_i with probability 1 given θ_i.

Computing Bayes-Nash Equilibria via Complete-information Interpretations

Harsanyi [1967] showed that a Bayesian game can be interpreted as one of two equivalent complete-information games via both "induced normal form" and "agent form" interpretations. Specifically, the Nash equilibria of these complete-information games correspond to Bayes-Nash equilibria of the Bayesian game. (A detailed description of these correspondences is given in Chapter 6.) Thus one approach is to interpret a Bayesian game as a complete-information game, enabling the use of existing Nash-equilibrium-finding algorithms. However, as mentioned in Section 2.1.3, generating the normal form representations under both of these complete-information interpretations leads to an exponential blowup in representation size.

Howson and Rosenthal [1974] applied the agent form transformation to 2-player Bayesian games, resulting in a complete-information polymatrix game which (recall from Section 2.2.2) can be solved using a variant of the Lemke-Howson algorithm. Their approach was able to avoid the aforementioned exponential blowup because in this case the agent forms admit a more compact representation (as polymatrix games). However, for n-player Bayesian games the corresponding agent forms do not correspond to polymatrix games or any other known representation. Nevertheless, in Chapter 6 we propose a general approach for computing sample Bayes-Nash equilibria in n-player Bayesian games (and BAGGs in particular).

Specifically, our approach solves the agent form of the BAGG using black-box versions of the Global Newton Method [Govindan and Wilson, 2003] and the simplicial subdivision algorithm [van der Laan et al., 1987], and instead of explicitly constructing the normal form of the agent form we use the BAGG as a compact representation of its agent form.

Special-purpose Approaches

Singh et al. [2004] proposed an incomplete information version of the graphical game representation, and presented efficient algorithms for computing approximate Bayes-Nash equilibria in the case of tree games. Gottlob et al. [2007] considered a similar extension of the graphical game representation and analyzed the problem of finding a pure-strategy Bayes-Nash equilibrium. Oliehoek et al. [2010] proposed a heuristic search algorithm for common-payoff Bayesian games, which has applications to cooperative multi-agent problems.

2.2.4 Computing Sample Nash Equilibria for Dynamic Games

In perfect-information extensive-form games, all information sets contain a single node. As a result, each subtree of the extensive-form game tree form a *subgame* which can be solved independently of the rest of the tree. The *backward induction* algorithm computes a Nash equilibrium of the game by solving subgames from the leaves to the root. The running time is linear in the size of the extensive form. Furthermore, when the game is zero sum, it is possible to prune parts of the game tree that are not optimal. The canonical algorithm, Alpha-Beta pruning, has been influential in the design of high-performance game-playing systems for perfect-information games such as chess and checkers.

For extensive-form games with imperfect information, transforming to the induced normal form entails an exponential blowup in representation size. This is the main difficulty of the Nash equilibrium problem for dynamic games compared to the simultaneous-move case, and avoiding this exponential blowup is the focus of considerable existing literature.

One common assumption is *perfect recall*: roughly, that each player remembers all her decisions and observations. For dynamic games with perfect recall,

there always exists a Nash equilibrium in *behavior strategies*, where a player independently chooses a distribution over actions at each of her information sets [Kuhn, 1953]. Computationally, behavior strategies are easier to work with, since representing a behavior strategy requires space linear in the extensive form, while representing a mixed strategy (i.e. a distribution over pure strategies) requires exponential space. For MAIDs, a behavior strategy for a player entails choosing, at each of her decision nodes and for each possible instantiation of the node's parents, a probability distribution over her choices.

The *sequence form* formulation of Koller, Meggido and von Stengel [1996] encodes a behavior strategy as a vector of "realization probabilities". Using this formulation, the Nash equilibrium problem for zero-sum dynamic games can be formulated as a linear program of size polynomial in the extensive form representation. For two-player general-sum dynamic games, using the sequence form the Nash equilibrium problem can be formulated as a linear complementarity program (LCP) and solved using Lemke's algorithm.

For n-player games, Govindan and Wilson [2002] proposed an extension of their Global Newton Method to perfect-recall extensive-form games. As with the sequence form, strategies are encoded as realization probabilities. Daskalakis et al. [2006a] showed that the problem of finding a Nash equilibrium in behavior strategies for perfect-recall extensive-form games is in PPAD.

For compact representations, existing approaches can again be divided into black-box and special-purpose ones. Koller and Milch [2001] proposed a special-purpose approach for decomposing a MAID into subgraphs, each of which can be solved independently. As in the simultaneous-move case, the computation of expected utility is again an important subtask used by many game-theoretic computations. For example, such a subroutine can be used to run fictitious play, although (like in the simultaneous-move case) it is not guaranteed to converge. Blum et al. [2006] proposed a black-box approach for adapting Govindan and Wilson's Global Newton Method for extensive-form games to MAIDs, by speeding up the subtask of computing the Jacobian matrix using a MAID-specific subroutine. In Chapter 5 we show that this algorithm can also be adapted to TAGGs.

2.2.5 Questions about the Set of All Nash Equilibria of a Game

So far we have focused on finding one arbitrary Nash equilibrium. Since in general there can be more than one Nash equilibrium in a game, we are sometimes more interested in questions about the set of all Nash equilibria. Such problems include finding all Nash equilibria, counting the number of equilibria, and finding optimal Nash equilibria according to some objective such as *social welfare*, which is defined to be the sum of the players' utilities. Unsurprisingly, such problems are usually intractable in the worst case (see e.g. [Conitzer and Sandholm, 2008]).

For the problem of finding all Nash equilibria, Mangasarian [1964] proposed an algorithm for bimatrix games. More recently, Avis et al. [2010] described and implemented two algorithms for bimatrix games. Herings and Peeters [2005] proposed an algorithm that computes all Nash equilibria in an *n*-player normal form game by enumerating all support profiles. Compared to the support-enumeration method for finding a sample Nash equilibrium as discussed in Section 2.2.1, here the algorithm does not stop at a single Nash equilibrium and keeps going until all support profiles have been visited. At each support profile, the corresponding polynomial system is solved by either polynomial homotopy continuation or Groebner basis methods.

For the problem of computing optimal Nash equilibria, Sandholm et al. [2005] proposed and evaluated a practical approach for bimatrix games using mixed-integer programming.

2.2.6 Computing Pure-Strategy Nash Equilibria

A *pure-strategy Nash equilibrium (PSNE)*, also known as *pure Nash equilibrium* or *pure equilibrium*, is a pure strategy profile that is a Nash equilibrium. Equivalently:

Definition 2.2.5. *An action profile $a \in A$ is a* pure-strategy Nash equilibrium (PSNE) *of the game Γ if for all $i \in N$, for all $a'_i \in A_i$, $u_i(a_i, a_{-i}) \geq u_i(a'_i, a_{-i})$.*

Unlike mixed strategy Nash equilibria, PSNEs do not always exist in a game. Nevertheless, in many ways PSNE is a more attractive solution concept than mixed-strategy Nash equilibrium. First, PSNE can be easier to justify because it does not require the players to randomize. Second, it can be easier to analyze because of

its discrete nature (see, e.g., [Brandt et al., 2009]). There are several versions of the problem of computing PSNEs: deciding if a PSNE exists, finding one, counting the number of PSNEs, enumerating them, and finding the optimal equilibrium according to some objective (e.g., social welfare). Unlike the NASH problem, for games in normal form these problems can be solved in polynomial time in the input size, by enumerating all pure strategy profiles. Of course, since the size of the normal form representation grows exponentially in the number of players, this is problematic in practice. We thus focus on the problem for compact representations. The problem is hard in the most general case, when utility functions are arbitrary, efficiently-computable functions represented as circuits [Schoenebeck and Vadhan, 2006] or Turing Machines [Alvarez et al., 2005]. This is in contrast to the NASH case, where the Nash problems for both the normal form and fully-expressive compact representations are PPAD-complete.

Iterated Best Response

Iterated best response is a well-known both as a learning dynamics and as a heuristic algorithm for PSNE [e.g., Shoham and Leyton-Brown, 2009]. It is an iterative process starting at some arbitrary pure strategy profile. At each step, if there exists a player that is not playing a best response to the current pure strategy profile, that player changes her strategy to a best response. The process stops when all are playing best responses, in which case we have reached a PSNE. A related process is *iterated better response*, in which a deviating player only has to pick a pure strategy that is better than the current one. These processes can be carried out for all representations that provide efficient evaluation of utilities under arbitrary pure-strategy profiles. However, like fictitious play, these are not guaranteed to converge for games in general.

Graphical Games

Gottlob et al. [2005] were the first to analyze the existence problem of pure-strategy Nash equilibria in graphical games. They proved that while the problem is NP-complete in general, on games with graphs of bounded hypertree-width there exist a dynamic-programming algorithm that determines the existence of PSNE (and

finds one if it exists) in time polynomial in the size of the representation. Daskalakis and Papadimitriou [2006] reduced the problem to a Markov Random Field (MRF), and then applied the standard clique tree algorithm to the resulting MRF. Among their results they showed that for graphical games on graphs with log-sized treewidth, and bounded neighborhood size and bounded number of actions per player, the existence of pure Nash equilibria can be decided in polynomial time.

Jiang and Safari [2010] analyzed the problem of deciding the existence of pure-strategy Nash equilibria for graphical games on restricted classes of graphs, and gave a complete characterization of hard and easy classes of graphical games with bounded indegree, showing that the only tractable classes of graphs are those with bounded treewidth (after iterated removal of sinks).

Daskalakis and Papadimitriou [2005] analyzed the complexity of finding pure and mixed Nash equilibria of graphical games on highly regular graphs (specifically, the d-dimensional grid) with identical local payoff functions for every player. Such games can be represented very compactly, as only the local payoff function at one neighborhood needs to be stored. They showed that finding pure-strategy Nash equilibria is tractable if $d = 1$ and NEXP-complete otherwise.

Symmetric Games

For symmetric games, questions about PSNE can be computed straightforwardly by checking all configurations, which requires polynomial time in the size of the representation, and polynomial time in n when the number of actions is fixed. Indeed, Brandt et al. [2009] proved that the existence problem for PSNE of symmetric games with constant number of actions is in the complexity class AC^0, which is the set of problems that can be solved by polynomial-sized constant-depth circuits with unlimited-fanin AND- and OR-gates. For anonymous games, efficient algorithms for PSNE have also been proposed [Brandt et al., 2009, Daskalakis and Papadimitriou, 2007].

Ryan et al. [2010] considered the problem of finding pure-strategy Nash equilibria in symmetric games whose utilities are very compactly represented, such that the number of players can be exponential in the representation size, and showed that if the utility functions are represented as piecewise-linear functions, there exist

polynomial-time algorithms for finding a pure-strategy Nash equilibria and count the number of equilibria.

Congestion Games

For congestion games, a PSNE always exists [Rosenthal, 1973]. Furthermore, iterated best-response dynamics always converge to a PSNE [Monderer and Shapley, 1996]. However, Fabrikant et al. [2004] showed that such dynamics may require an exponential number of steps to converge, and furthermore the problem of finding a PSNE for congestion games is complete for the complexity class PLS (which stands for Polynomial Local Search), which implies that a polynomial-time algorithm is unlikely to exist.

For *singleton congestion games*, where the game is symmetric and each action consists of choosing only a single resource, Ieong et al. [2005] presented a polynomial-time algorithm for finding an optimal PSNE.

AGGs

Since AGGs can compactly encode arbitrary graphical games, the existence problem is NP-complete for AGGs. Conitzer [pers. comm., 2004] and Daskalakis et al. [2009] showed that the problem is NP-complete even for symmetric AGGs.

In Chapter 4 we present a dynamic programming approach for computing PSNE in AGGs. For symmetric AGGs with bounded treewidth, our algorithm determines the existence of PSNE (and returns one if any exists) in polynomial time. We also show that our approach can be extended to certain classes of asymmetric AGGs.

2.2.7 Computing Correlated Equilibrium

First proposed by Aumann [1974, 1987], correlated equilibrium (CE) is another important solution concept. Whereas in a mixed strategy Nash equilibrium players randomize independently, in a correlated equilibrium the players are allowed to coordinate their behavior based on signals from an intermediary. CE has interesting connections to the theory of online learning: the empirical distribution of no-internal-regret learning dynamics converge to the set of CE [e.g., Hart and

Mas-Colell, 2000, Nisan et al., 2007].

A correlated equilibrium is defined as a distribution x over action profiles, such that when a trusted intermediary draws a strategy profile a from this distribution, privately announcing to each player i her own component a_i, i will have no incentive to choose another strategy, assuming others follow the suggestions. This requirement can be written as a set of linear *incentive constraints* on x. Combining these with the constraints that x is a distribution, the set of correlated equilibria can be formulated as a linear feasibility program with size polynomial in the size of the normal form. (A detailed description of this formulation is given in Chapter 7.) Thus it takes polynomial time in the size of the normal form to compute one CE, and indeed to compute an optimal CE according to some linear objective function.

For compact representations, the same LP can have an exponential number of variables, due to the fact that the input size can be exponentially smaller. Thus, the above approach is not efficient for compact representations. Another challenge is that even explicitly representing a solution vector x can take exponential space. Thus, a compact representation for the distribution x is required. Furthermore, in order for the intermediary to be able to tractably implement such a correlated equilibrium, we also need an efficient algorithm for sampling from the distribution.

In a landmark paper, Papadimitriou and Roughgarden [2008] proposed a black-box algorithm for computing a sample CE, which runs in polynomial time when the game representation has polynomial type and when there is a polynomial-time algorithm for computing expected utility given mixed strategy profiles. The solutions are represented as mixtures of product distributions. Recently, Stein, Parrilo and Ozdaglar [2010] showed that this algorithm can fail to find an exact correlated equilibrium, but can be (easily) modified to efficiently compute approximate correlated equilibria. In Chapter 7 we present a variant of the Ellipsoid Against Hope algorithm that guarantees the polynomial-time identification of *exact* correlated equilibrium.

For the problem of computing the optimal CE, Papadimitriou and Roughgarden [2008] showed that the problem is NP-hard for many existing representations, and gave a sufficient condition for the problem to be tractable. They showed that symmetric games, anonymous games and graphical games on tree graphs satisfy such a condition. In Chapter 8 we give a sufficient condition that generalizes Pa-

padimitriou and Roughgarden's condition. In particular, we reduce the optimal CE problem to the *deviation-adjusted social welfare problem*, a combinatorial optimization problem closely related to the optimal social welfare outcome problem. This framework allows us to identify new classes of games for which the optimal CE problem is tractable, including graphical polymatrix games on tree graphs. Our algorithm can be understood as a black-box algorithm, with *deviation-adjusted social welfare problem* as the required subroutine.

A couple of special-purpose approaches have been proposed for graphical games. Kakade et al. [2003] proposed an algorithm for computing a CE with maximum entropy in tree graphical games in polynomial time. More recently, Kamisetty et al. [2011] proposed a practical approach for approximating the optimal CE in graphical games.

Computing Coarse Correlated Equilibria

Coarse correlated equilibrium (CCE) [Hannan, 1957] is a solution concept closely related to CE. The difference between the two is the class of deviations they consider. Whereas CE requires that each player have no profitable deviation even if she takes into account the signal she receives from the intermediary, CCE only requires that each player have no profitable *unconditional deviation*. CCE is also related to online learning: the empirical distribution of a no-external-regret learning dynamics converge to the set of CCE.

As in the case of CE, the set of CCE can also be formulated as an LP. A formal description is given in Chapter 8. A CE is also a CCE, and hence results for the polynomial-time computation of a sample CE also apply to the computation of a sample CCE.

On the other hand, since the optimal CE problem is not always tractable, the optimal CCE problem could be easier than the optimal CE problem for some representations. In Chapter 8 we show that for singleton congestion games, the optimal CCE problem can be solved in polynomial time, while the complexity of the optimal CE problem for this class of games is unknown.

Computing Extensive-form Correlated Equilibria

Recently, von Stengel and Forges [2008] proposed extensive-form correlated equilibrium (EFCE), a solution concept for perfect-recall extensive-form games that is closely related to correlated equilibrium. Recall that in an extensive-form game, each pure strategy of a player prescribes a move for each of her information sets. Like correlated equilibria, an EFCE is a distribution over pure-strategy profiles. Whereas in a CE of the induced normal form of the game the intermediary recommends a pure strategy to each player at the start of the game, in an EFCE the intermediary recommends a move to the player only when the corresponding information set is reached.

Huang and Von Stengel [2008] described a polynomial-time algorithm for computing sample extensive-form correlated equilibria. Their algorithm follows a very similar structure as Papadimitriou and Roughgarden's Ellipsoid Against Hope algorithm, and the flaws of the Ellipsoid Against Hope algorithm pointed out by Stein et al. [2010] also carry over. As a result, the algorithm can fail to find an exact EFCE. In Chapter 7 we extend our fix for Papadimitriou and Roughgarden's Ellipsoid Against Hope algorithm to Huang and Von Stengel's algorithm, allowing it to compute an exact EFCE.

2.2.8 Computing Other Solution Concepts

Other solution concepts have been proposed in the economics literature to represent different notions of rational behavior. Computer scientists have studied the corresponding computational problems, including the computation of (iterated) elimination of dominated strategies [Conitzer and Sandholm, 2005], Stackelberg equilibrium [Conitzer and Sandholm, 2006, Paruchuri et al., 2008], closed under rational behavior (CURB) sets [M. Benisch and Sandholm, 2010], and sink equilibrium [Goemans et al., 2005].

While these are interesting problems, they are not directly related to this thesis and we refer interested readers to the papers referenced above.

2.3 Software

GAMBIT [McKelvey et al., 2006] is a collection of software tools for game theoretic analysis. It includes implementations of many of the existing algorithms for the normal form and the extensive form. It also provides a graphical user interface for creating normal form and extensive form games, running algorithms for computing Nash equilibria, and visualizing the resulting profiles. It is available at http://www.gambit-project.org.

Gametracer [Blum et al., 2002] provides black-box adaptations of two of Govindan and Wilson's algorithms for finding a sample Nash equilibrium: Global Newton Method [Govindan and Wilson, 2003] and Iterated Polymatrix Approximation [Govindan and Wilson, 2004]. The algorithms are written as C++ functions that takes an instance of "`gnmgame`", an abstract class with an abstract method[4] for computing expected utilities.[5] As a result, in order to apply these algorithms to a specific game representation, one merely has to implement the representation as a subclass of `gnmgame`. The package itself only provides a subclass for the normal form representation. Gametracer's source code is available for download at http://dags.stanford.edu/Games/gametracer.html. It has also been adapted and incorporated into GAMBIT.

GAMUT [Nudelman et al., 2004] is a suite of game instance generators. It includes many classes of games studied in the economics and computer science literature, and parameterization options for the dimensions of the game, the types of utility functions and randomization. The stated purpose of GAMUT is for evaluating game-theoretic algorithms. The main output format for GAMUT is normal form. GAMUT is available at http://gamut.stanford.edu.

In Appendix A we describe the software tools we implemented and make available at http://agg.cs.ubc.ca. They include command-line programs for finding sample Nash equilibria in AGGs and BAGGs, a graphical user interface for creating, editing and visualizing AGGs, and extensions of GAMUT that generate AGG instances.

[4] An abstract method in C++ means that only the interface of method is given; any subclass that is not also abstract needs to provide an implementation of the method.

[5] Another abstract method is for computing payoff Jacobians (see Chapter 3 for the definition), which usually requires similar types of computations as expected utilities.

Chapter 3

Action-Graph Games

3.1 Introduction

In this chapter we focus on complete-information simultaneous-action games. An overview of the literature on compact representations and computation of solution concepts for such games is given in Chapter 2, specifically Sections 2.1.1, 2.2.2 and 2.2.7. As we summarized in Chapter 1, the existing representations either only capture a subset of the known types of structure (anonymity, strict and action-specific independence, and additivity), or are only able to represent a subset of games. Meanwhile, the computation of *expected utility* has emerged as a key subtask required by many black-box algorithms for computing solution concepts.

3.1.1 Our Contributions

Action-graph games (AGGs) are a general game representation that can be understood as offering the advantages of—and, indeed, unifying—existing representations including graphical games and congestion games. Like graphical games, AGGs can represent any game, and important game-theoretic computations can be performed efficiently when the AGG representation is compact. Hence, AGGs offer a general representational framework for game-theoretic computation. Like congestion games, AGGs compactly represent context-specific independence, anonymity, and additivity, though unlike congestion games they do not require any of these. Finally, AGGs can also compactly represent many games that are not compact as

either graphical games or as congestion games.

We begin this chapter in Section 3.2 by defining action-graph games, including the basic representation and extensions with function nodes and additive utility functions, and characterizing their representation sizes. In Section 3.3 we provide several more examples of structured games which can be compactly represented as AGGs. Then we turn from representational to computational issues. In Section 3.4 we present a dynamic programming algorithm for computing an agent's expected utility under an arbitrary mixed-strategy profile, prove its complexity, and explore several elaborations. In Section 3.5 we show that (as a corollary of the polynomial complexity of our expected utility algorithm) the problem of finding an ε-Nash equilibrium of an AGG is in PPAD: this is a positive result, as AGGs can be exponentially smaller than normal-form games. We also show how to use our dynamic programming algorithm to speed up existing methods for computing sample ε-Nash and ε-correlated equilibria. Finally, in Section 3.6 we present the results of extensive experiments with some of these algorithms, demonstrating that AGGs can feasibly be used to reason about interesting games that were inaccessible to any previous techniques. The largest game that we tackled in our experiments had 20 agents and 13 actions per agent; we found its Nash equilibrium in 14.3 minutes. A normal form representation of this game would have involved 9.4×10^{134} numbers, requiring an outrageous 7.5×10^{126} gigabytes even to store.

Finally, let us describe the relationship between this chapter and past work on AGGs. Leyton-Brown and Tennenholtz [2003] introduced local-effect games, which can be understood as symmetric AGGs in which utility functions are required to satisfy a particular linearity property. Bhat and Leyton-Brown [2004] introduced the basic AGG representation and some of the computational ideas for reasoning with them. The dynamic programming algorithm was first proposed in Jiang and Leyton-Brown [2006], as was the idea of function nodes. An extended version of that paper appeared as Chapter 2 of the MSc thesis [Jiang, 2006]. The current chapter is based on the journal publication [Jiang et al., 2011], which substantially elaborates upon and extends the representations and methods from these earlier papers. Specifically, [Jiang et al., 2011] introduced the additive structure model and the encoding of congestion games, several of the examples, our computational methods for *k*-symmetric games and for additive structure, our speedup of

the simplicial subdivision algorithm, and all experiments presented in this chapter (Section 3.6).

3.2 Action Graph Games

This section has three parts, each of which defines a different AGG variant. In Section 3.2.1 we define the basic AGG representation (which we dub AGG-$\emptyset$), characterize its representation size, and show how it can be used to represent normal-form, graphical, and symmetric games. In Section 3.2.2 we introduce the idea of *function nodes*, show how AGGs with function nodes (AGG-FNs) can capture additional structure in several example games, and show how to represent anonymous games as AGG-FNs. In Section 3.2.3 we introduce AGG-FNs with additive structure (AGG-FNA), which compactly represent additive structure in the utility functions of AGGs, and show how congestion games can be succinctly written as AGG-FNAs.

3.2.1 Basic Action Graph Games

We begin with an intuitive description of basic action-graph games. Consider a directed graph with nodes $\mathscr{A}$ and edges E, and a set of agents $N = \{1,\ldots,n\}$. Identical tokens are given to each agent $i \in N$. To play the game, each agent i simultaneously places her token on a node $a_i \in A_i$, where $A_i \subseteq \mathscr{A}$. Each node in the graph thus corresponds to an action choice that is available to one or more of the agents; this is where action-graph games get their name. Each agent's utility is calculated according to an arbitrary function of the node she chose and the *numbers* of tokens placed on the nodes that neighbor that chosen node in the graph. We will argue below that any simultaneous-move game can be represented in this way, and that action-graph games are often much more compact than games represented in other ways.

We now turn to a formal definition of basic action-graph games. Let $N = \{1,\ldots,n\}$ be the set of agents. Central to our model is the *action graph*.

Definition 3.2.1 (Action graph)**.** *An* action graph $G = (\mathscr{A}, E)$ *is a directed graph where:*

- *$\mathscr{A}$ is the set of nodes. We call each node $\alpha \in \mathscr{A}$ an* action*, and $\mathscr{A}$ the* set of distinct actions*. For each agent $i \in N$, let A_i be the set of actions available to i, with $\mathscr{A} = \bigcup_{i \in N} A_i$.*[1] *We denote by $a_i \in A_i$ one of agent i's actions. An* action profile *(or* pure strategy profile*) is a tuple $a = (a_1, \ldots, a_n)$. Denote by A the set of action profiles. Then $A = \prod_{i \in N} A_i$ where $\prod$ is the Cartesian product.*
- *E is a set of directed edges, where self edges are allowed. We say α' is a* neighbor *of α if there is an edge from α' to α, i.e., $(\alpha', \alpha) \in E$. Let the* neighborhood *of α, denoted $\nu(\alpha)$, be the set of neighbors of α, i.e., $\nu(\alpha) \equiv \{\alpha' \in \mathscr{A} | (\alpha', \alpha) \in E\}$.*

Given an action graph and a set of agents, we can further define a *configuration*, which is a feasible arrangement of agents across nodes in an action graph.

Definition 3.2.2 (Configuration)**.** *Given an action graph $(\mathscr{A}, E)$ and a set of action profiles A, a* configuration *c is a tuple of $|\mathscr{A}|$ non-negative integers $(c(\alpha))_{\alpha \in \mathscr{A}}$, where $c(\alpha)$ is interpreted as the number of agents who chose action $\alpha \in \mathscr{A}$, and where there exists some $a \in A$ that would give rise to c. Denote the set of all configurations as C. Let $\mathscr{C} : A \to C$ be the function that maps from an action profile a to the corresponding configuration c. Formally, if $c = \mathscr{C}(a)$ then $c(\alpha) = |\{i \in N : a_i = \alpha\}|$ for all $\alpha \in \mathscr{A}$.*

We can also restrict a configuration to a given node's neighborhood.

Definition 3.2.3 (Configuration over a neighborhood)**.** *Given a configuration $c \in C$ and a node $\alpha \in \mathscr{A}$, let the* configuration over the neighborhood *of α, denoted $c^{(\alpha)}$, be the restriction of c to $\nu(\alpha)$, i.e., $c^{(\alpha)} = (c(\alpha'))_{\alpha' \in \nu(\alpha)}$. Similarly, let $C^{(\alpha)}$ denote the set of configurations over $\nu(\alpha)$ in which at least one player plays α.*[2] *Let $\mathscr{C}^{(\alpha)} : A \to C^{(\alpha)}$ be the function which maps from an action profile to the corresponding configuration over $\nu(\alpha)$.*

[1] Different agents' action sets A_i, A_j may (partially or completely) overlap. The implications of this will become clear once we define the utility functions.

[2] If action α 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 α (denoted $C^{(s,i)}$) is different from the set of configurations given that j played α. $C^{(\alpha)}$ is the union of these sets of configurations.

Now we can state the formal definition of basic action-graph games as follows.

Definition 3.2.4 (Basic action-graph game)**.** *A basic action-graph game (AGG-∅) is a tuple* (N, A, G, u) *where*

- N *is the set of agents;*
- $A = \prod_{i \in N} A_i$ *is the set of action profiles;*
- $G = (\mathscr{A}, E)$ *is an action graph, where* $\mathscr{A} = \bigcup_{i \in N} A_i$ *is the set of distinct actions;*
- $u = (u^\alpha)_{\alpha \in \mathscr{A}}$ *is a tuple of* $|\mathscr{A}|$ *functions, where each* $u^\alpha : C^{(\alpha)} \to \mathbb{R}$ *is the* utility function *for action* α*. Semantically,* $u^\alpha(c^{(\alpha)})$ *is the utility of an agent who chose* α*, when the configuration over* $\nu(\alpha)$ *is* $c^{(\alpha)}$*.*

For notational convenience, we define $u(\alpha, c^{(\alpha)}) \equiv u^\alpha(c^{(\alpha)})$ and $u_i(a) \equiv u(a_i, \mathscr{C}^{(a_i)}(a))$. We also define $A_{-i} \equiv \prod_{j \neq i} A_j$ as the set of action profiles of agents other than i, and denote an element of A_{-i} by a_{-i}.

Example: Ice Cream Vendors

The following example helps to illustrate the elements of the AGG-∅ representation, and also exhibits context-specificity and anonymity in utility functions. This example would not be compact under the existing game representations discussed in the introduction. It was inspired by Hotelling [1929], and elaborates an example used in Leyton-Brown and Tennenholtz [2003].

Example 3.2.5 (Ice Cream Vendor game)**.** *Consider a setting in which n vendors sell ice cream or strawberries, and must choose one of four locations along a beach. There are three kinds of vendors:* n_I *ice cream vendors,* n_S *strawberry vendors, and* n_W *vendors who can sell both ice cream and strawberry, but only on the west side. Ice cream (strawberry) vendors are negatively affected by the presence of other ice cream (strawberry) vendors in the same or neighboring locations, and are simultaneously positively affected by the presence of nearby strawberry (ice cream) vendors.*

The AGG-∅ representation of this game is illustrated in Figure 3.1. As always, nodes represent actions and directed edges represent membership in a node's

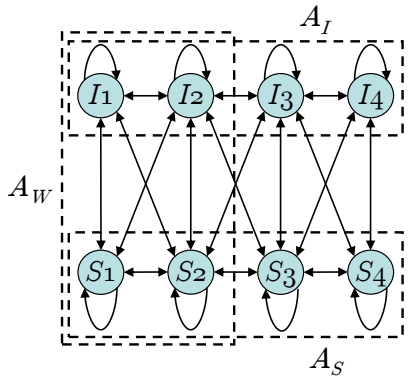

Figure 3.1: AGG-$\emptyset$ representation of the Ice Cream Vendor game.

neighborhood. The dotted boxes represent the action sets for each group of players; for example, the ice cream vendors have action set A_I. Note that this game exhibits context-specific independence without any strict independence, and that the graph structure is independent of n.

Size of an AGG-$\emptyset$ Representation

Intuitively, AGG-$\emptyset$s capture two types of structure in games:

1. Shared actions capture the game's *anonymity* structure: agent i's utility depends only on her action a_i and the configuration. Thus, agent i cares about the *number* of players that play each action, but not the identities of those players.
2. The (lack of) edges between nodes in the action graph expresses *context-specific independencies* of utilities of the game: for all $i \in N$, if i chose action $\alpha \in \mathscr{A}$, then i's utility depends only on the configuration over the neighborhood of α. In other words, the configuration over actions not in $\nu(\alpha)$ does not affect i's utility.

We have claimed informally that action graph games provide a way of representing games compactly. But what exactly is the size of an AGG-$\emptyset$ representation, and how does it grow with the number of agents n? In this subsection we give a

bound on the size of an AGG-$\emptyset$, and show that asymptotically it is never worse than the size of the equivalent normal form.

From Definition 3.2.4 we observe that to completely specify an AGG-$\emptyset$ we need to specify (1) the set of agents, (2) each agent's set of actions, (3) the action graph, and (4) the utility functions. The first three can easily be compactly represented:

1. The set of agents $N = \{1, \ldots, n\}$ can be specified by the integer n.
2. The set of actions $\mathscr{A}$ can be specified by the integer $|\mathscr{A}|$. Each agent's action set $A_i \subseteq \mathscr{A}$ can be specified in $O(|\mathscr{A}|)$ space.
3. The action graph $G = (\mathscr{A}, E)$ can be straightforwardly represented as neighbor lists: for each node $\alpha \in \mathscr{A}$ we specify its list of neighbors $\nu(\alpha) \subseteq \mathscr{A}$. The space required is $\sum_{\alpha \in \mathscr{A}} |\nu(\alpha)|$, which is bounded by $|\mathscr{A}|\mathscr{I}$, where $\mathscr{I} = \max_\alpha |\nu(\alpha)|$, i.e., the maximum in-degree of G.

We observe that whereas the first three components of an AGG-$\emptyset$ (N, A, G, u) can always be represented in space polynomial in n and $|A_i|$, the size of the utility functions is worst-case exponential. So the size of the utility functions determines whether an AGG-$\emptyset$ can be tractably represented. Indeed, for the rest of the paper we will refer to the number of payoff values stored as the representation size of the AGG-$\emptyset$. The following proposition gives an upper bound on the number of payoff values stored.

Proposition 3.2.6. *Given an AGG-$\emptyset$, the number of payoff values stored by its utility functions is at most $|\mathscr{A}| \frac{(n-1+\mathscr{I})!}{(n-1)!\mathscr{I}!}$. If $\mathscr{I}$ is bounded by a constant as n grows, the number of payoff values is $O(|\mathscr{A}| n^{\mathscr{I}})$, i.e. polynomial with respect to n.*

Proof. For each utility function $u^\alpha : C^{(\alpha)} \to \mathbb{R}$, we need to specify a utility value for each distinct configuration $c^{(\alpha)} \in C^{(\alpha)}$. The set of configurations $C^{(\alpha)}$ can be derived from the action graph, and can be sorted in lexicographical order. Thus, we can just specify a list of $|C^{(\alpha)}|$ utility values that correspond to the (ordered) set of configurations.[3] In general there is no closed form expression for $|C^{(\alpha)}|$, the number of distinct configurations over $\nu(\alpha)$. Instead, we consider the operation of extending all agents' action sets via $\forall i : A_i \mapsto \mathscr{A}$. The number of configurations over

$\nu(\alpha)$ under the new action sets is an upper bound on $|C^{(\alpha)}|$. This is the number of (ordered) combinatorial compositions of $n-1$ (since one player has already chosen α) into $|\nu(\alpha)|+1$ nonnegative integers, which is $\binom{n-1+|\nu(\alpha)|}{|\nu(\alpha)|} = \frac{(n-1+|\nu(\alpha)|)!}{(n-1)!|\nu(\alpha)|!}$. Then the total space required for the utilities is bounded from above by $|\mathscr{A}|\frac{(n-1+\mathscr{I})!}{(n-1)!\mathscr{I}!}$. If $\mathscr{I}$ is bounded by a constant as n grows, this grows like $O(|\mathscr{A}|n^{\mathscr{I}})$. □

For each AGG-$\emptyset$, there exists a unique *induced normal form* representation with the same set of players and $|A_i|$ actions for each i; its utility function is a matrix that specifies each player i's payoff for each possible action profile $a \in A$. This implies a space complexity of $n\prod_{i=1}^{n}|A_i|$. When $A_i \geq 2$ for all i, the size of the induced normal form representation grows exponentially with respect to n. On the other hand, we observe that the number of payoff values stored in an AGG-$\emptyset$ representation is always less than or equal to the number of payoff values in the induced normal form representation. Of course, the AGG-$\emptyset$ representation has the extra overhead of representing the action graph, which is bounded by $|\mathscr{A}|\mathscr{I}$. But this overhead is dominated by the size of the induced normal form, $n\prod_j |A_j|$. Thus, an AGG-$\emptyset$'s asymptotic space complexity is never worse than that of its induced normal form game.

It is also possible to describe a reverse transformation that encodes any arbitrary game in normal form as an AGG-$\emptyset$. Specifically, a unique node a_i must be created for each action available to each agent i. Thus $\forall\alpha \in \mathscr{A}$, $c(\alpha) \in \{0,1\}$, and $\forall i$, $\sum_{\alpha\in A_i} c(\alpha)$ must equal 1. The configuration simply indicates each agent's action choice, and expresses no anonymity or context-specific independence structure.

This representation is no more or less compact than the normal form. More precisely, the number of distinct configurations over $\nu(a_i)$ is the number of action profiles of the other players, which is $\prod_{j\neq i}|A_j|$. Since i has $|A_i|$ actions, $\prod_j |A_j|$ payoff values are needed to represent i's payoffs. So in total $n\prod_j |A_j|$ payoff values are stored, exactly the number in the normal form.

[3]This is the most compact way of representing the utility functions, but does not provide easy random access to the utilities. Therefore, when we want to do computation using AGGs, we may convert each utility function u^{α} 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 $O(\mathscr{I}|C^{(\alpha)}|)$.

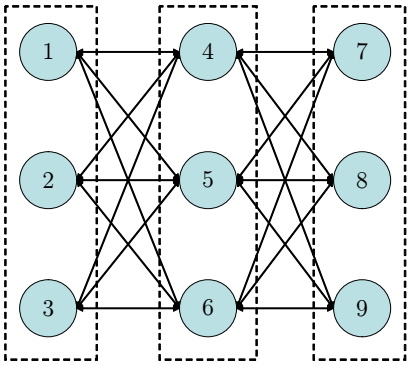

Figure 3.2: AGG-∅ representation of a 3-player, 3-action graphical game.

One might ask whether AGG-∅s can compactly represent known classes of structured games. Consider the graphical game representation as defined in Definition 2.1.2. Graphical games can be represented as AGG-∅s by replacing each node i in the graphical game by a distinct cluster of nodes A_i representing the action set of agent i. If the graphical game has an edge from i to j, edges must be created in the AGG-∅ so that $\forall a_i \in A_i, \forall a_j \in A_j,\ a_i \in \nu(a_j)$. The resulting AGG-∅s are as compact as the original graphical games. Figure 3.2 shows the AGG-∅ representation of a graphical game having three nodes and two edges (i.e., player 1 and player 3 do not directly affect each others' payoffs).

Another important class of structured games are symmetric games as defined in Section 2.1.1. An arbitrary symmetric game can be encoded as an AGG-∅ without an increase in asymptotic size. Specifically, let $A_i = \mathscr{A}$ for all $i \in N$. The resulting action graph is a clique, i.e., $\nu(\alpha) = \mathscr{A}$ for all $\alpha \in \mathscr{A}$.

3.2.2 AGGs with Function Nodes

There are games with certain kinds of context-specific independence structures that AGG-∅s are not able to exploit (see, e.g., Example 3.2.7 below). In this section we extend the AGG-∅ representation by introducing *function nodes*, allowing us to exploit a much wider variety of utility structures. Of course, as always, compact representation is not interesting as an end in itself. In Section 3.4.2 we identify broad subclasses of AGG-FNs—indeed, rich enough to encompass all AGG-FN examples presented in this chapter —which are amenable to efficient computation.

Examples: Coffee Shops and Parity

Example 3.2.7 (Coffee Shop game)**.** *Consider a game involving n players; each player plans to open a coffee shop in a downtown area, represented by a $r \times k$ grid. Each player can choose to open a shop located within any of the $B \equiv rk$ blocks or decide not to enter the market. Conditioned on player i choosing some location α, her utility depends on the numbers of players who chose (i) the same block; (ii) any of the surrounding blocks; and (iii) any other location.*

The normal form representation of this game has size $n|\mathscr{A}|^n = n(B+1)^n$. Since there are no strict independencies in the utility function, the asymptotic size of the graphical game representation is the same. Let us now represent the game as an AGG-$\emptyset$. We observe that if agent i chooses an action α corresponding to one of the B locations, then her payoff is affected by the configuration over all B locations. Hence, $v(\alpha)$ must consist of B action nodes corresponding to the B locations, and so the action graph has in-degree $\mathscr{I} = B$. Since the action sets completely overlap, the representation size is $\Theta(|\mathscr{A}||C^{(\alpha)}|) = \Theta\left(B\frac{(n-1+B)!}{(n-1)!B!}\right)$. If we hold B constant, this becomes $\Theta(Bn^B)$, 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 $\Theta(B^n)$, which is only slightly better than the normal form and graphical game representations.

Intuitively, the AGG-$\emptyset$ representation is able to exploit anonymity structure in this game. However, this game's payoff function also has context-specific structure that the AGG-$\emptyset$ does not capture. Observe that u^α depends only on three quantities: the number of players who chose the same block, the number of players who chose an adjacent block, and the number of players who chose another location. In other words, u^α can be written as a function g of only three integers: $u^\alpha(c^{(\alpha)}) = g(c(\alpha), \sum_{\alpha' \in \mathscr{A}'} c(\alpha'), \sum_{\alpha'' \in \mathscr{A}''} c(\alpha''))$ where $\mathscr{A}'$ is the set of actions surrounding α and $\mathscr{A}''$ the set of actions corresponding to other locations. The AGG-$\emptyset$ representation is not able to exploit this context-specific information, and so duplicates some utility values.

There exist many similar examples in which the utility functions u^α can be expressed as functions of a small number of intermediate parameters. Here we give one more.

Example 3.2.8 (Parity game)**.** *In a "parity game", each u^α depends only on whether the number of agents at neighboring nodes is even or odd, as follows:*

$$u^\alpha = \begin{cases} 1 & \text{if } \sum_{\alpha' \in \nu(\alpha)} c(\alpha') \mod 2 = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Observe that in the Parity game u^α can take just two distinct values; however, the AGG-$\emptyset$ representation must specify a value for every configuration $c^{(\alpha)}$.

Definition of AGG-FNs

Structure such as that in Examples 3.2.7 and 3.2.8 can be exploited within the AGG framework by introducing *function nodes* to the action graph G; intuitively, we use them to describe intermediate parameters upon which players' utilities depend. Now G's vertices consist of both the set of action nodes $\mathscr{A}$ and the set of function nodes $\mathscr{P}$, i.e. $G = (\mathscr{A} \cup \mathscr{P}, E)$. We require that no function node $p \in \mathscr{P}$ can be in any player's action set: $\mathscr{A} \cap \mathscr{P} = \{\}$. Thus, the total number of nodes in G is $|\mathscr{A}| + |\mathscr{P}|$. Each node in G can have action nodes and/or function nodes as neighbors. We associate a function $f^p : C^{(p)} \to \mathbb{R}$ with each $p \in \mathscr{P}$, where $c^{(p)} \in C^{(p)}$ denotes configurations over p's neighbors. The configurations c are extended to include the function nodes by the definition $c(p) \equiv f^p(c^{(p)})$. If $p \in \mathscr{P}$ has no neighbors, f^p is a constant function. To ensure that the AGG is meaningful, the graph G restricted to nodes in $\mathscr{P}$ is required to be a directed acyclic graph (DAG). This condition ensures that for all α and p, $c(\alpha)$ and $c(p)$ are well defined. To ensure that every $p \in \mathscr{P}$ is "useful", we also require that p has at least one outgoing edge. As before, for each action node α we define a utility function $u^\alpha : C^{(\alpha)} \to \mathbb{R}$. We call this extended representation an Action Graph Game with Function Nodes (AGG-FN), and define it formally as follows.

Definition 3.2.9 (AGG-FN)**.** *An Action Graph Game with Function Nodes (AGG-FN) is a tuple $(N, A, \mathscr{P}, G, f, u)$, where:*

- *N is the set of agents;*
- *$A = \prod_{i \in N} A_i$ is the set of action profiles;*
- *$\mathscr{P}$ is a finite set of function nodes;*

- $G = (\mathscr{A} \cup \mathscr{P}, E)$ *is an action graph, where* $\mathscr{A} = \bigcup_{i \in N} A_i$ *is the set of distinct actions. We require that the restriction of G to the nodes* $\mathscr{P}$ *is acyclic and that for every* $p \in \mathscr{P}$ *there exists an* $m \in \mathscr{A} \cup \mathscr{P}$ *such that* $(p, m) \in E$*;*
- f *is a tuple* $(f^p)_{p \in \mathscr{P}}$*, where each* $f^p : C^{(p)} \to \mathbb{R}$ *is an arbitrary mapping from neighbors of p to real numbers;*
- u *is a tuple* $(u^\alpha)_{\alpha \in \mathscr{A}}$*, where each* $u^\alpha : C^{(\alpha)} \to \mathbb{R}$ *is the* utility function *for action* α*.*

Given an AGG-FN, we can construct an equivalent AGG-$\emptyset$ with the same players N and actions $\mathscr{A}$ and equivalent utility functions, but without any function nodes. We call this the *induced AGG-*$\emptyset$ of the AGG-FN. There is an edge from α' to α in the induced AGG-$\emptyset$ either if there is an edge from α' to α in the AGG-FN, or if there is a path from α' to α through a chain consisting entirely of function nodes. From the definition of AGG-FNs, the utility of playing action α is uniquely determined by the configuration $c^{(\alpha)}$, which is uniquely determined by the configuration over the actions that are neighbors of α in the induced AGG-$\emptyset$. As a result, the utility tables of the induced AGG-$\emptyset$ can be filled in unambiguously. We observe that the number of utility values stored in an AGG-FN is no greater than the number of utility values in the induced AGG-$\emptyset$. On the other hand, AGG-FNs have to represent the functions f^p for each $p \in \mathscr{P}$. In the worst case, these functions can be represented as explicit mappings similar to the utility functions u^α. However, it is often possible to define these functions algebraically by combining elementary operations, as we do in most of the examples given in this chapter . In this case the functions' representations require a negligible amount of space.

Representation Size

What is the size of an AGG-FN $(N, A, \mathscr{P}, G, f, u)$? The following proposition gives a sufficient condition for the representation size to be polynomial. Here we speak about a *class* of AGG-FNs because our statement is about the asymptotic behavior of the representation size. This is in contrast to Proposition 3.2.6, where we gave an exact bound on the size of an individual AGG-$\emptyset$.

Proposition 3.2.10. *A class of AGG-FNs has representation size bounded by a function polynomial in n,* $|\mathscr{A}|$ *and* $|\mathscr{P}|$ *if the following conditions hold:*

1. *for all function nodes $p \in \mathscr{P}$, the size of p's range $|\mathscr{R}(f^p)|$ is bounded by a function polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$; and*
2. $\max_{m \in A \cup \mathscr{P}} \nu(m)$ *(the maximum in-degree in the action graph) is bounded by a constant.*

Proof. Given an AGG-FN $(N, A, \mathscr{P}, G, f, u)$, it is straightforward to check that all components except u and f are polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$.

First, consider an action node $\alpha \in \mathscr{A}$. Recall that the size of the utility function u^α is $C^{(\alpha)}$. Partition $\nu(\alpha)$, the set of α's neighbors, into $\nu_{\mathscr{A}}(\alpha) = \nu(\alpha) \cap \mathscr{A}$ and $\nu_{\mathscr{P}}(\alpha) = \nu(\alpha) \cap \mathscr{P}$ (neighboring action nodes and function nodes respectively). Since for each action $\alpha' \in \nu_{\mathscr{A}}(\alpha)$, $c(\alpha') \in \{0, \ldots, n\}$, and for each $p' \in \nu_{\mathscr{P}}(\alpha)$, $c(p) \in \mathscr{R}(f^p)$, then $C^{(\alpha)} \leq (n+1)^{|\nu_{\mathscr{A}}(\alpha)|} \prod_{p \in \nu_{\mathscr{P}}(\alpha)} |\mathscr{R}(f^p)|$. This is polynomial because all action node in-degrees are bounded by a constant.

Now consider a function node $p \in \mathscr{P}$. Without loss of generality, assume that its function f^p is represented explicitly as a mapping. (Any other representation of f^p can be transformed into this explicit representation.) The representation size of f^p is then $C^{(p)}$. Using the same reasoning as above, we have $C^{(p)} \leq (n+1)^{|\nu_{\mathscr{A}}(p)|} \prod_{q \in \nu_{\mathscr{P}}(p)} |\mathscr{R}(f^q)|$, which is polynomial since all function node in-degrees are bounded by a constant. □

When the functions f^p do not have to be represented explicitly, we can drop the requirement on the in-degree of function nodes.

Corollary 3.2.11. *A class of AGG-FNs has representation size bounded by a function polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$ if the following conditions hold:*

1. *for all function nodes $p \in \mathscr{P}$, the function f^p has a representation whose size is polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$;*
2. *for each function node $p \in \mathscr{P}$ that is a neighbor of some action node α, the size of p's range $|\mathscr{R}(f^p)|$ is bounded by a function polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$; and*
3. $\max_{\alpha \in A} \nu(\alpha)$ *(the maximum in-degree among action nodes) is bounded by a constant.*

A very useful type of function node is the *simple aggregator*.

Definition 3.2.12 (Simple aggregator)**.** *A function node $p \in \mathscr{P}$ is a* simple aggregator *if each of its neighbors $\nu(p)$ are action nodes and f^p is the summation function: $f^p(c^{(p)}) = \sum_{m \in \nu(p)} c(m)$.*

Simple aggregator function nodes take the value of the total number of players who chose any of the node's neighbors. Since these functions can be specified in constant space, and since $\mathscr{R}(f^p) = \{0, \ldots, n\}$ for all p, Corollary 3.2.11 applies. That is, the representation sizes of AGG-FNs whose function nodes are all simple aggregators are polynomial whenever the in-degrees of action nodes are bounded by a constant. In fact, under certain assumptions we can prove an even tighter bound on the representation size, analogous to Proposition 3.2.6 for AGG-$\emptyset$s. Intuitively, this works because both configurations on action nodes and configurations on simple aggregators count the numbers of players who behave in certain ways.

Proposition 3.2.13. *Consider a class of AGG-FNs whose function nodes are all simple aggregators. For each $m \in \mathscr{A} \cup \mathscr{P}$, define the function*

$$\beta(m) = \begin{cases} m & m \in A; \\ \nu(m) & \textit{otherwise.} \end{cases}$$

Intuitively, $\beta(m)$ is the set of nodes whose counts are aggregated by node m. If for each $\alpha \in \mathscr{A}$ and for each $m, m' \in \nu(\alpha)$, $\beta(m) \cap \beta(m') = \{\}$ unless $m = m'$ (i.e., no action node affects α in more than one way), then the AGG-FNs' representation sizes are bounded by $|\mathscr{A}|\binom{n-1+\mathscr{I}}{\mathscr{I}}$ where $\mathscr{I} = \max_{\alpha \in A} |\nu(\alpha)|$ is the maximum in-degree of action nodes.

Proof. Consider the utility function u^α for an arbitrary action α. Each neighbor $m \in \nu(\alpha)$ is either an action or a simple aggregator. Observe that a configuration $c^{(\alpha)} \in C^{(\alpha)}$ is a tuple of integers specifying the numbers of players choosing each action in the set $\beta(m)$ for each $m \in \nu(\alpha)$. As in the proof of Proposition 3.2.6, we extend each player's set of actions to $|\mathscr{A}|$, making the game symmetric. This weakly increases the number of configurations. Since the sets $\beta(m)$ are non-overlapping, the number of configurations possible in the extended action space is equal to the number of (ordered) combinatorial compositions of $n-1$ into $|\nu(\alpha)|+1$ nonnegative integers, which is $\binom{n-1+|\nu(\alpha)|}{|\nu(\alpha)|}$. This includes one bin for

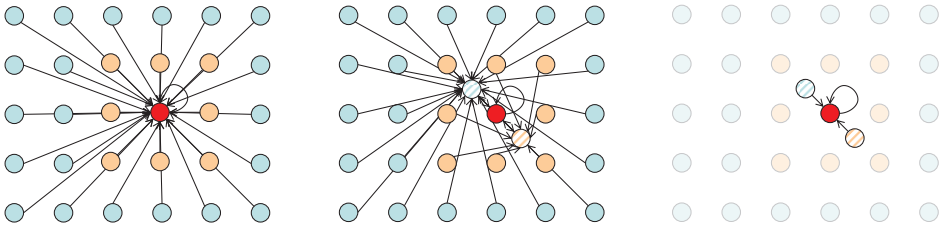

Figure 3.3: A 5×6 Coffee Shop game: Left: the AGG-$\emptyset$ representation without function nodes (looking at only the neighborhood of α). Middle: we introduce two function nodes, p' (bottom) and p'' (top). Right: α now has only 3 neighbors.

each action or simple aggregator in $\nu(\alpha)$, plus one bin for agents that take an action that is neither in $\nu(\alpha)$ nor in the neighborhood of any simple aggregator in $\nu(\alpha)$. Then the total space required for representing u is bounded by $|\mathscr{A}|\binom{n-1+\mathscr{I}}{\mathscr{I}}$ where $\mathscr{I} = \max_{\alpha \in A} |\nu(\alpha)|$. □

Consider the Coffee Shop game from Example 3.2.7. For each action node α corresponding to a location, we introduce two simple aggregator function nodes, p'_α and p''_α. Let $\nu(p'_\alpha)$ be the set of actions surrounding α, and $\nu(p''_\alpha)$ be the set of actions corresponding to other locations. Then we set $\nu(\alpha) = \{\alpha, p'_\alpha, p''_\alpha\}$, as shown in Figure 3.3. Now each $c^{(\alpha)}$ is a configuration over only three nodes. Since each f^p is a simple aggregator, Corollary 3.2.11 applies and the size of this AGG-FN is polynomial in n and $\mathscr{A}$. In fact since the game is symmetric and the $\beta()$'s as defined in Proposition 3.2.13 are non-overlapping, we can calculate the exact value of $|C^{(\alpha)}|$ as the number of compositions of $n-1$ into four nonnegative integers, $\frac{(n+2)!}{(n-1)!3!} = n(n+1)(n+2)/6 = O(n^3)$. We must therefore store $Bn(n+1)(n+2)/6 = O(Bn^3)$ utility values. This is significantly more compact than the AGG-$\emptyset$ representation, which has a representation size of $O(B\frac{(n-1+B)!}{(n-1)!B!})$.

We can represent the parity game from Example 3.2.8 in a similar way. For each action α we create a function node p_α, and let $\nu(p_\alpha) = \nu(\alpha)$. We then modify $\nu(\alpha)$ so that it has only one member, p_α. For each function node p we define f^p as $f^p(c^{(p)}) = \sum_{\alpha \in \nu(p)} c(\alpha) \mod 2$. Since $\mathscr{R}(f^p) = \{0,1\}$, Corollary 3.2.11 applies. In fact, each utility function just needs to store two values, and so the representation size is $O(|\mathscr{A}|)$ plus the size of the action graph.

3.2.3 AGG-FNs with Additive Structure

So far we have assumed that the utility functions $u^\alpha : C^{(\alpha)} \to \mathbb{R}$ are represented explicitly, i.e., by specifying the payoffs for all $c^{(\alpha)} \in C^{(\alpha)}$. This is not the only way to represent a mapping; the utility functions could be defined as analytical functions, decision trees, logic programs, circuits, or even arbitrary algorithms. These alternative representations might be more natural for humans to specify, and in many cases are more compact than the explicit representation. However, this extra compactness does not always allow us to reason more efficiently with the games. In this section, we look at utility functions with *additive structure*. These functions can be represented compactly and do allow more efficient computation.

Definition of AGG-FNs with Additive Structure

We say that a multivariate function has *additive structure* if it can be written as a (weighted) sum of functions of subsets of the variables. This form is more compact because we only need to represent the summands, which have lower dimensionality than the entire function.

We extend the AGG-FN representation by allowing u^α to be represented as a weighted sum of the configuration of the neighbors of α.[4]

Definition 3.2.14. *A utility function u^α of an AGG-FN is* additive *if for all $m \in \nu(\alpha)$ there exist $\lambda_m \in \mathbb{R}$, such that*

$$u^\alpha(c^{(\alpha)}) \equiv \sum_{m \in \nu(\alpha)} \lambda_m c(m). \tag{3.2.1}$$

Such an additive utility function can be represented as the tuple $(\lambda_m)_{m \in \nu(\alpha)}$. This is a very versatile representation of additivity, because the neighbors of α can be function nodes. Thus additive utility functions can represent weighted sums of arbitrary functions of configurations over action nodes. We now formally define an AGG-FN representation where some of the utility functions are additive.

[4] Such a utility function could also be represented using standard function nodes representing summation. However, we treat the common case of additivity separately because it is amenable to special-purpose computational methods (intuitively, leveraging the linearity of expectation; see Section 3.4.3).

Definition 3.2.15. *An* AGG-FN with additive structure (AGG-FNA) *is a tuple* $(N, A, \mathscr{P}, G, f, \mathscr{A}_+, \Lambda, u)$ *where* $N, A, \mathscr{P}, G, f$ *are as defined in Definition 3.2.9, and*

- $\mathscr{A}_+ \subseteq \mathscr{A}$ *is the set of actions whose utility functions are additive;*
- $\Lambda = (\lambda^{\alpha_+})_{\alpha_+ \in \mathscr{A}_+}$*, where each* $\lambda^{\alpha_+} = (\lambda_m^{\alpha_+})_{m \in \nu(\alpha)}$ *is the tuple of coefficients representing the additive utility function* u^{α_+}*;*
- $u = (u^\alpha)_{\alpha \in \mathscr{A} \setminus \mathscr{A}_+}$*, where each* u^α *is as defined in Definition 3.2.9. These are the non-additive utility functions of the game, which are represented explicitly.*

Representation Size

We only need $|\nu(\alpha)|$ numbers to represent the coefficients of an additive utility function u^α, whereas the explicit representation requires $|C^{(\alpha)}|$ numbers. Of course we also need to take into account the sizes of the neighboring function nodes $p \in \nu(\alpha)$ and their corresponding functions f^p, which represent the summands of the additive functions. Each f^p either has a simple description requiring negligible space, or is represented explicitly as a mapping. In the latter case its size can be analyzed the same way as utility functions on action nodes. That is, when the neighbors of p are all actions then Proposition 3.2.6 applies; otherwise the discussion in Section 3.2.2 applies.

Representing Congestion Games as AGG-FNAs

An arbitrary congestion game can be encoded as an AGG-FNA with no loss of compactness, where all u^α are represented as additive utility functions. Given a congestion game $(N, M, (A_i)_{i \in N}, (K_{jk})_{j \in M, k \leq n})$ as defined in Definition 2.1.1, we construct an AGG-FNA with the same number of players and same number of actions for each player as follows.

- Create $\sum_{i \in N} |A_i|$ action nodes, corresponding to the actions in the congestion game. In other words, the action sets do not overlap.
- Create $2m$ function nodes, labeled $(p_1, \ldots, p_m, q_1, \ldots, q_m)$. For each $j \in M$, there is an edge from p_j to q_j. For all $j \in M$ and for all $\alpha \in \mathscr{A}$, if facility j

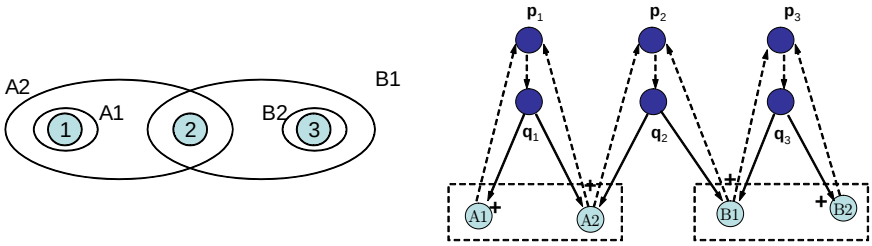

Figure 3.4: Left: a two-player congestion game with three facilities. The actions are shown as ovals containing their respective facilities. Right: the AGG-FNA representation of the same congestion game.

is included in action α in the congestion game, then in the action graph there is an edge from the action node α to p_j, and also an edge from q_j to α.

- For each p_j, define $c(p_j) \equiv \sum_{\alpha \in \nu(j)} c(\alpha)$, i.e., p_j is a simple aggregator. Since its neighbors are the actions that includes facility j, thus $c(p_j)$ is the number of players that chose facility j, which is $\#(j,a)$.
- Assign each q_j only one neighbor, namely p_j, and define $c(q_j) \equiv f^{q_j}(c(p_j)) \equiv K_j(c(p_j))$. In other words, $c(q_j)$ is exactly $K_j(\#(j,a))$, the cost on facility j.
- For each action node α, represent the utility function u^α as an additive function with weight -1 for each of its neighbors,

$$u^\alpha(c^{(\alpha)}) = \sum_{j \in \nu(\alpha)} -c(j) = -\sum_{j \in \nu(\alpha)} K_j(\#(j,a)). \tag{3.2.2}$$

Example 3.2.16 (Congestion game)**.** *Consider the AGG-FNA representation of a two-player congestion game (see Figure 3.4). The congestion game has three facilities labeled {1, 2, 3}. Player A has actions A1={1} and A2={1, 2}; Player B has actions B1={2, 3} and B2={3}.*

Now let us consider the representation size of this AGG-FNA. The action graph has $|\mathscr{A}| + 2m$ nodes and $O(m|\mathscr{A}|)$ edges; the function nodes $p_1, \ldots, p_m$ are simple aggregators and each only requires constant space; each f^{q_j} requires n numbers to specify so the total size of the AGG-FNA is $\Theta(mn + m|\mathscr{A}|) = \Theta(mn + m\sum_{i \in N} |A_i|)$.

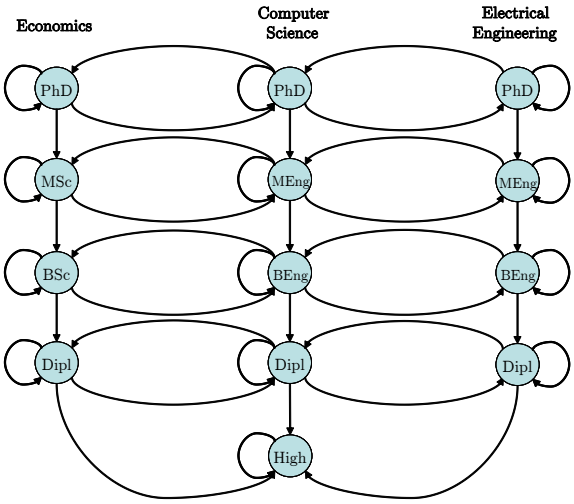

Figure 3.5: AGG-∅ representation of the Job Market game.

Thus this AGG-FNA representation has the same space complexity as the original congestion game representation.

One extension of congestion games is *player-specific congestion games* [Milchtaich, 1996, Monderer, 2007]. Instead of all players having the same costs K_{jk}, in these games each player has a different set of costs. This can be easily represented as an AGG-FNA by following the construction above, but using a different set of function nodes $q_{i1}, \ldots, q_{im}$ for each player i.

3.3 Further Examples

In this section we provide several more examples of structured games that can be compactly represented as AGGs.

3.3.1 A Job Market

Here we describe a class of example games that can be compactly represented as AGG-∅s. Unlike the Ice Cream Vendor game, the following example does not involve choosing among actions that correspond to geographical locations.

Example 3.3.1 (Job Market game)**.** *Consider the individuals competing in a job market. Each player chooses a field of study and a level of education to achieve.*

The utility of player i is the sum of two terms: (a) a constant cost depending only on the chosen field and education level, capturing the difficulty of studies and the cost of tuition and forgone wages; and (b) a variable reward, depending on (i) the number of players who chose the same field and education level as i, (ii) the number of players who chose a related field at the same education level, and (iii) the number of players who chose the same field at one level above or below i.

Figure 3.5 gives an action graph modeling one such job market scenario, in which there are three fields, Economics, Computer Science and Electrical Engineering . For each field there are four levels of postsecondary study: Diploma, Bachelor, Master and PhD. Economics and Computer Science are considered related fields, and so are Computer Science and Electrical Engineering. There is another action representing high school education, which does not require a choice of field. The maximum in-degree of the action graph is five, whereas a naive representation of the game as a symmetric game (see Section 3.2.1) would correspond to a complete action graph with in-degree 13. Thus this AGG-$\emptyset$ representation is able to take advantage of anonymity as well as context-specific independence structure.

3.3.2 Representing Anonymous Games as AGG-FNs

One property of the AGG-$\emptyset$ representation as defined in Section 3.2.1 is that utility function u^α is shared by all players who have α in their action sets. What if we want to represent games with *agent-specific* utility functions, where utilities depend not only on α and $c^{(\alpha)}$, but also on the *identity* of the player playing α?

As mentioned in Section 2.1.1, researchers have studied *anonymous games*, which deviate from symmetric games by allowing agent-specific utility functions [Daskalakis and Papadimitriou, 2007, Kalai, 2004, 2005]. To represent games of this type as AGGs, we cannot just let multiple players share action α, because that would force those players to have the same utility function u^α. It does work to give agents non-overlapping action sets, replicating each action once for each agent. However, the resulting AGG-$\emptyset$ is not compact; it does not take advantage of the fact that each of the replicated actions affects other players' utilities in the same way. Using function nodes, it is possible to compactly represent this kind of structure. We again split α into separate action nodes α_i for each player i able

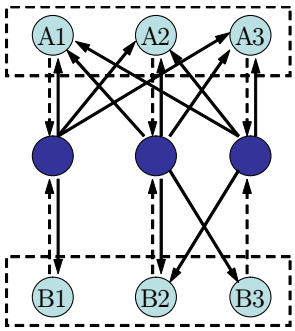

Figure 3.6: AGG-FN representation of a game with agent-specific utility functions.

to take the action. Now we also introduce a function node p with every α_i as a neighbor, and define f^p to be a simple aggregator. Now p gives the total number of agents who chose action α, expressing anonymity, and action nodes include p as a neighbor instead of each α_i. This allows agents to have different utility functions without sacrificing representational compactness.

Example 3.3.2 (Anonymous game)**.** *Consider an anonymous game with two classes of players, each class sharing the same utility functions. The AGG-FN representation of the game is shown in Figure 3.6. Players from the first class have action set* $\{A1, A2, A3\}$*, and players from the second class have action set* $\{B1, B2, B3\}$*. Furthermore, the utility functions of the second class of players exhibit certain context-specific independence structure, which are expressed by the absence of some of the possible edges from function nodes to action nodes B1, B2, B3.*

3.3.3 Representing Polymatrix Games as AGG-FNAs

A polymatrix game (defined in Section 2.1.1) can be compactly represented as an AGG-FNA. The encoding is as follows. The AGG-FNA has non-overlapping action sets. For each pair of players (i, j), we create two function nodes to represent i and j's payoffs under the bimatrix game between them. Each of these function nodes has incoming edges from all of i's and j's actions. For each player i and each of his actions a_i, there are incoming edges from the $n-1$ function nodes representing i's payoffs in his bimatrix games against each of the other players.

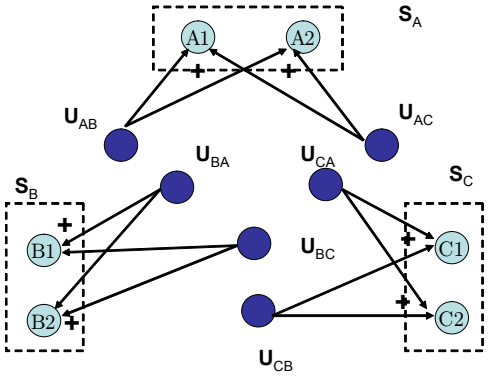

Figure 3.7: AGG-FNA representation of a 3-player polymatrix game. Function node U_{AB} represents player A's payoffs in his bimatrix game against B, U_{BA} represents player B's payoffs in his bimatrix game against A, and so on. To avoid clutter we do not show the edges from the action nodes to the function nodes in this graph. Such edges exist from A and B's actions to U_{AB} and U_{BA}, from A and C's actions to U_{AC} and U_{CA}, and from B and C's actions to U_{BC} and U_{CB}.

u^{a_i} is an additive utility function with weights equal to 1. Based on arguments similar to those in Section 3.2.1, this AGG-FNA representation has the same space complexity as the total size of the bimatrix games.

Example 3.3.3 (Polymatrix game)**.** *Consider the AGG-FNA representation of a three-player polymatrix game, given in Figure 3.7. Each player's payoff is the sum of her payoffs in* 2×2 *game with played with each of the other players; she is only able to choose her action once. This additive utility function can be captured by introducing a function node* U_{ij} *to represent each player* i*'s utility in the bimatrix game played with player* j*.*

3.3.4 Congestion Games with Action-Specific Rewards

So far the only use we have shown for AGG-FNAs is bringing existing game representations into the AGG framework. Of course, another key advantage of our approach is the ability to compactly represent games that would not have been compact under these existing game representations. We now give such an example.

Example 3.3.4 (Congestion game with action-specific rewards)**.** *Consider the following game with n players. As in a congestion game, there is a set of facilities M, each action involves choosing a subset of the facilities, and the cost for facility j depends only on the number of players that chose facility j. Now further assume that, in addition to the cost of using the facilities, each player i also derives some utility R_i depending only on her own action a_i, i.e., the set of facilities she chose. This utility is not necessarily additive across facilities. That is, in general if $A, B \subset M$ and $A \cap B = \emptyset$, $R_i(A \cup B) \neq R_i(A) + R_i(B)$. So i's total utility is*

$$u_i(a) = R_i(a_i) - \sum_{j \in a_i} K_j(\#(j,a)). \tag{3.3.1}$$

This game can model a situation in which the players use the facilities to complete a task, and the utility of the task depends on the facilities chosen. Another interpretation is given by Ben-Sasson et al. [2006], in their analysis of "congestion games with strategy costs," which also have exactly this type of utility function. This work interpreted (the negative of) $R_i(a_i)$ as the computational cost of choosing the pure strategy a_i in a congestion game.

Due to the extra $R_i(a_i)$ term in the utility expression (3.3.1)*, this game cannot be directly represented as a congestion game or a player-specific congestion game,*[5] *but it can be compactly represented as an AGG-FNA. We create $\sum_i |A_i|$ action nodes, giving the agents nonoverlapping action sets. We have shown in Section 3.2.3 that we can use function nodes and additive utility functions to represent the congestion-game-like costs. Beyond this construction, we just need to create a function node r_i for each player i and define $c(r_i)$ to be equal to $R_i(a_i)$. The neighbors of r_i are i's entire action set: $v(r_i) = A_i$. Since the action sets do not overlap, there are only $|A_i|$ distinct configurations over A_i. In other words, $|C^{(r_i)}| = |A_i|$ and we need only $O(|A_i|)$ space to represent each R_i. The total size of the representation is $O(mn + m\sum_{i \in N} |A_i|)$.*

[5] Interestingly, Ben-Sasson et al. [2006] showed that this game belongs to the set of potential games, which implies that there exists an equivalent congestion game. However, building such a congestion game from the potential function following Monderer and Shapley's [1996] construction yields an exponential number of facilities, meaning that this congestion game representation is exponentially larger than the AGG-FNA representation presented here.

3.4 Computing Expected Payoff with AGGs

Up to this point, we have concentrated on how AGGs may be used to compactly represent games of interest. But compact representation is only half the story, and indeed by itself is relatively easy to achieve. Our goal is to identify a compact representation that can be used directly (e.g., without conversion to its induced normal form) for the computation of game-theoretic quantities of interest. We now turn to this computational perspective, and show that we can indeed leverage AGG's representational compactness in the computation of game-theoretic quantities. In this section we focus on the computational task of computing an agent's expected payoff under a mixed strategy profile. As we discussed in Section 2.2, this task is important as an inner-loop problem in the computation of many game-theoretic quantities, including Govindan and Wilson's [2003, 2004] algorithms for finding Nash equilibria, the simplicial subdivision algorithm for finding Nash equilibria [van der Laan et al., 1987], and Papadimitriou and Roughgarden's [2008] algorithm for finding correlated equilibria. We discuss some of these applications in Section 3.5.

Our main result of this section is an algorithm that efficiently computes expected payoffs of AGGs by exploiting their context-specific independence, anonymity and additivity structure. In Section 3.4.1 we introduce our expected payoff algorithm for AGG-$\emptyset$s, and show (in Theorem 3.4.1) that the algorithm runs in time polynomial in the size of the input AGG-$\emptyset$. For the special case of symmetric strategies in symmetric AGG-$\emptyset$s, we present a different algorithm in Section 3.4.1 which runs asymptotically faster than our general algorithm for AGG-$\emptyset$s; in Section 3.4.1 we extend this approach to the broader class of *k-symmetric* AGG-$\emptyset$s. Finally, in Sections 3.4.2 and 3.4.3 we extend our expected payoff algorithm to AGG-FNs and AGG-FNAs respectively, and identify (in Theorems 3.4.5 and 3.4.6) conditions under which these extended algorithms run in polynomial time.

3.4.1 Computing Expected Payoff for AGG-$\emptyset$s

Following the notation of Section 2.2, we denote a mixed strategy of i by $\sigma_i \in \Sigma_i$, a mixed-strategy profile by $\sigma \in \Sigma$, and the probability that i plays action α as $\sigma_i(\alpha)$.

Now we can write the expected utility to agent i for playing pure strategy a_i,

given that all other agents play the mixed strategy profile σ_{-i}, as

$$V^i_{a_i}(\sigma_{-i}) \equiv \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \Pr(a_{-i}|\sigma_{-i}), \tag{3.4.1}$$

$$\Pr(a_{-i}|\sigma_{-i}) \equiv \prod_{j \neq i} \sigma_j(a_j). \tag{3.4.2}$$

Note that Equation 3.4.2 gives the probability of a_{-i} under the mixed strategy σ_{-i}. In the rest of this section we focus on the problem of computing $V^i_{a_i}(\sigma_{-i})$ given i, a_i and σ_{-i}. Having established the machinery to compute $V^i_{a_i}(\sigma_{-i})$, we can then compute the expected utility of player i under a mixed strategy profile σ as $\sum_{a_i \in A_i} \sigma_i(a_i) V^i_{a_i}(\sigma_{-i})$.

One might wonder why Equations (3.4.1) and (3.4.2) are not the end of the story. Notice that Equation (3.4.1) is a sum over the set A_{-i} of action profiles of players other than i. The number of terms is $\prod_{j \neq i} |A_j|$, which grows exponentially in n. If we were to use the normal form representation, there really would be $|A_{-i}|$ different outcomes to consider, each with potentially distinct payoff values. Thus, using normal form the evaluation of Equation (3.4.1) would be the best possible algorithm for computing $V^i_{a_i}$. Since AGGs are fully expressive, the same is true for games without any structure represented as AGGs. However, what about games that are exponentially more compact when represented as AGGs than when represented in the normal form? For these games, evaluating Equation (3.4.1) amounts to an exponential-time algorithm.

In this section we present an algorithm that given any i, a_i and σ_{-i}, computes the expected payoff $V^i_{a_i}(\sigma_{-i})$ in time polynomial in the size of the AGG-$\emptyset$ representation. In other words, our algorithm is efficient if the AGG-$\emptyset$ is compact, and requires time exponential in n if it is not. In particular, recall from Proposition 3.2.6 any AGG-$\emptyset$ with maximum in-degree bounded by a constant has a representation size that is polynomial in n. As a result our algorithm is polynomial in n for such games.

Exploiting Context-Specific Independence: Projection

First, we consider how to take advantage of the context-specific independence structure of an AGG-$\emptyset$: the fact that i's payoff when playing a_i only depends on

configurations over the neighborhood of i. The key idea is that we can *project* other players' strategies onto a smaller action space that is strategically the same from the point of view of an agent who chose action a_i. That is, we construct a graph from the point of view of a given agent, expressing his sense that actions that do not affect his chosen action are in a sense the "same action." This can be seen as inducing a context-specific graphical game. Formally, for every action $\alpha \in \mathscr{A}$ define a reduced graph $G^{(\alpha)}$ by including only the nodes $\nu(\alpha)$ and a new node denoted $\emptyset$. The only edges included in $G^{(\alpha)}$ are the directed edges from each of the nodes $\nu(\alpha)$ to the node α. Player j's action a_j is projected to a node $a_j^{(\alpha)}$ in the reduced graph $G^{(\alpha)}$ by the mapping

$$a_j^{(\alpha)} \equiv \begin{cases} a_j & a_j \in \nu(\alpha) \\ \emptyset & a_j \notin \nu(\alpha) \end{cases} . \tag{3.4.3}$$

In other words, actions that are not in $\nu(\alpha)$ (and therefore do not affect the payoffs of agents playing α) are projected onto a new action, $\emptyset$. The resulting *projected* action set $A_j^{(\alpha)}$ has cardinality at most $\min(|A_j|, |\nu(\alpha)|+1)$. This is illustrated in Figure 3.8, using the Ice Cream Vendor game described in Example 3.2.5.

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

$$\sigma_j^{(\alpha)}(a_j^{(\alpha)}) \equiv \begin{cases} \sigma_j(a_j) & a_j \in \nu(\alpha) \\ \sum_{\alpha' \in A_j \setminus \nu(\alpha)} \sigma_j(\alpha') & a_j^{(\alpha)} = \emptyset \end{cases} . \tag{3.4.4}$$

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

$$V_{a_i}^i(\sigma_{-i}) = \sum_{a_{-i}^{(a_i)} \in A_{-i}^{(a_i)}} u\left(a_i, \mathscr{C}^{(a_i)}(a_i, a_{-i})\right) \Pr\left(a_{-i}^{(a_i)} | \sigma_{-i}^{(a_i)}\right),$$
$$\Pr\left(a_{-i}^{(a_i)} | \sigma_{-i}^{(a_i)}\right) = \prod_{j \neq i} \sigma_j^{(a_i)}\left(a_j^{(a_i)}\right).$$

The summation is over $A_{-i}^{(a_i)}$, which in the worst case has $(|\nu(a_i)|+1)^{(n-1)}$ terms.

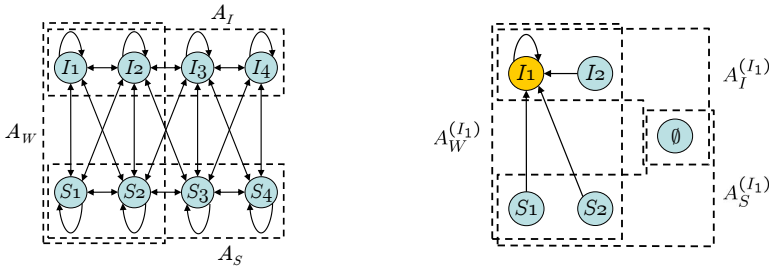

Figure 3.8: Projection of the action graph. Left: action graph of the Ice Cream Vendor game. Right: projected action graph and action sets with respect to the action C1.

So for AGG-$\emptyset$s with strict or context-specific independence structure, computing $V_{a_i}^i(\sigma_{-i})$ in this way is exponentially faster than doing the summation in (3.4.1) directly. However, the time complexity of this approach is still exponential in n.

Exploiting Anonymity: Summing over Configurations

Next, we want to take advantage of the anonymity structure of the AGG-$\emptyset$. 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 $V_{a_i}^i(\sigma_{-i})$ as a sum over the possible configurations, weighted by their probabilities:

$$V_{a_i}^i(\sigma_{-i}) = \sum_{c^{(a_i)} \in C^{(a_i,i)}} u_i\left(a_i, c^{(a_i)}\right) \Pr\left(c^{(a_i)} | \sigma^{(a_i)}\right), \tag{3.4.5}$$

$$\Pr\left(c^{(a_i)} | \sigma^{(a_i)}\right) = \sum_{\substack{a\,: \\ \mathscr{C}^{(a_i)}(a) = c^{(a_i)}}} \prod_{j=1}^{n} \sigma_j(a_j). \tag{3.4.6}$$

where $\sigma^{(a_i)} \equiv (a_i, \sigma_{-i}^{(a_i)})$ and $\Pr(c^{(a_i)}|\sigma^{(a_i)})$ is the probability of $c^{(a_i)}$ given the mixed strategy profile $\sigma^{(a_i)}$. Recall that $C^{(a_i,i)}$ is the set of configurations over $\nu(a_i)$ given that i played a_i. So Equation (3.4.5) is a summation of size $|C^{(a_i,i)}|$, the number of configurations given that i played a_i, which is polynomial in n if $|\nu(a_i)|$ is bounded by a constant. The difficult task is to compute $\Pr(c^{(a_i)}|\sigma^{(a_i)})$ for all $c^{(a_i)} \in C^{(a_i,i)}$,

i.e., the probability distribution over $C^{(a_i,i)}$ induced by $\sigma^{(a_i)}$. We observe that the sum in Equation (3.4.6) is over the set of all action profiles corresponding to the configuration $c^{(a_i)}$. The size of this set is exponential in the number of players. Therefore directly computing the probability distribution using Equation (3.4.6) would take time exponential in n.

Can we do better? We observe that the players' mixed strategies are independent, i.e., σ is a product probability distribution $\sigma(a) = \prod_i \sigma_i(a_i)$. Also, each player affects the configuration c independently. This structure allows us to use dynamic programming (DP) to efficiently compute the probability distribution $\Pr(c^{(a_i)}|\sigma^{(a_i)})$. The intuition behind our algorithm is to apply one agent's mixed strategy at a time, effectively adding one agent at a time to the action graph. Let $\sigma^{(a_i)}_{1\ldots k}$ denote the projected strategy profile of agents $\{1,\ldots,k\}$. Denote by $C^{(a_i)}_k$ the set of configurations induced by actions of agents $\{1,\ldots,k\}$. Similarly, write $c^{(a_i)}_k \in C^{(a_i)}_k$. Denote by P_k the probability distribution on $C^{(a_i)}_k$ induced by $\sigma^{(a_i)}_{1\ldots k}$, and by $P_k[c]$ the probability of configuration c. At iteration k of the algorithm, we compute P_k from P_{k-1} and $\sigma^{(a_i)}_k$. After iteration n, the algorithm stops and returns P_n. The pseudocode of our DP algorithm is shown as Algorithm 1, and our full algorithm for computing $V^i_{a_i}(\sigma_{-i})$ is summarized in Algorithm 2.

Each $c^{(a_i)}_k$ 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* [Fredkin, 1962]. 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 $|\nu(a_i)|$. To achieve efficient enumeration of all elements of a trie, we store the elements in a list, in the order of their insertion. We omit the proof of correctness of our algorithm, which is relatively straightforward.

Complexity

Let $C^{(a_i,i)}(\sigma_{-i})$ denote the set of configurations over $\nu(a_i)$ that have positive probability of occurring under the mixed strategy (a_i, σ_{-i}). In other words, this is the

Algorithm 1: Computing the induced probability distribution $\Pr(c^{(a_i)}|\sigma^{(a_i)})$.

Input: a_i, $\sigma^{(a_i)}$
Output: P_n, which is the distribution $\Pr(c^{(a_i)}|\sigma^{(a_i)})$ represented as a trie.
$c_0^{(a_i)} = (0,\ldots,0)$;
$P_0[c_0^{(a_i)}] = 1.0$; `// Initialization:` $C_0^{(a_i)} = \{c_0^{(a_i)}\}$
for $k = 1$ *to* n **do**
    Initialize P_k to be an empty trie;
    foreach $c_{k-1}^{(a_i)}$ *from* P_{k-1} **do**
        foreach $a_k^{(a_i)} \in A_k^{(a_i)}$ *such that* $\sigma_k^{(a_i)}(a_k^{(a_i)}) > 0$ **do**
            $c_k^{(a_i)} = c_{k-1}^{(a_i)}$;
            if $a_k^{(a_i)} \neq \emptyset$ **then**
                $c_k^{(a_i)}(a_k^{(a_i)})$ += 1 ; `// Apply action` $a_k^{(a_i)}$
            if $P_k[c_k^{(a_i)}]$ *does not exist yet* **then**
                $P_k[c_k^{(a_i)}] = 0.0$;
            $P_k[c_k^{(a_i)}]$ += $P_{k-1}[c_{k-1}^{(a_i)}] \times \sigma_k^{(a_i)}(a_k^{(a_i)})$;
return P_n

number of terms we need to add together when doing the weighted sum in Equation (3.4.5). When σ_{-i} has full support, $C^{(a_i,i)}(\sigma_{-i}) = C^{(a_i,i)}$.

Theorem 3.4.1. *Given an AGG-$\emptyset$ representation of a game, i's expected payoff $V_{a_i}^i(\sigma_{-i})$ can be computed in $\Theta(n|\mathscr{A}| + n|\nu(a_i)|^2|C^{(a_i,i)}(\sigma_{-i})|)$ time, which is polynomial in the size of the representation. If $\mathscr{I}$, the in-degree of the action graph, is bounded by a constant, $V_{a_i}^i(\sigma_{-i})$ can be computed in time polynomial in n.*

Proof. Since looking up an entry in a trie takes time linear in the size of the key, which is $|\nu(a_i)|$ in our case, the complexity of doing the weighted sum in Equation (3.4.5) is $O(|\nu(a_i)||C^{(a_i,i)}(\sigma_{-i})|)$.

Algorithm 1 requires n iterations; in iteration k, we look at all possible combinations of $c_{k-1}^{(a_i)}$ and $\alpha_k^{(a_i)}$, and in each case do a trie look-up which costs $\Theta(|\nu(a_i)|)$. Since $|\mathscr{A}_k^{(a_i)}| \leq |\nu(a_i)| + 1$, and $|C_{k-1}^{(a_i)}| \leq |C^{(a_i,i)}|$, the complexity of Algorithm 1 is $\Theta(n|\nu(a_i)|^2|C^{(a_i,i)}(\sigma_{-i})|)$. This dominates the complexity of summing up Equation (3.4.5). Adding the cost of computing $\sigma_{-i}^{(\alpha)}$, we get the overall complexity of

Algorithm 2 Computing expected utility $V_{a_i}^i(\sigma_{-i})$, given a_i and σ_{-i}.

1. for each $j \neq i$, compute the projected mixed strategy $\sigma_j^{(a_i)}$ using Equation (3.4.4):

$$\sigma_j^{(a_i)}(a_j^{(a_i)}) \equiv \begin{cases} \sigma_j(a_j) & a_j \in \nu(a_i) \\ \sum_{\alpha' \in A_j \setminus \nu(a_i)} \sigma_j(\alpha') & a_j^{(a_i)} = \emptyset \end{cases}.$$

2. compute the probability distribution $\Pr(c^{(a_i)}|a_i, \sigma_{-i}^{(a_i)})$ by following Algorithm 1.
3. calculate the expected utility using the following weighted sum (Equation (3.4.5)):

$$V_{a_i}^i(\sigma_{-i}) = \sum_{c^{(a_i)} \in C^{(a_i,i)}} u_i\left(a_i, c^{(a_i)}\right) \Pr\left(c^{(a_i)}|\sigma^{(a_i)}\right).$$

expected payoff computation $\Theta(n|\mathscr{A}| + n|\nu(a_i)|^2|C^{(a_i,i)}(\sigma_{-i})|)$.

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

The proof of Theorem 3.4.1 shows that besides exploiting the compactness of the AGG-$\emptyset$ representation, our algorithm is also able to exploit the cases where the mixed strategy profiles given have small support sizes, because the time complexity depends on $|C^{(a_i,i)}(\sigma_{-i})|$ which is small when support sizes are small. This is important in practice, since we will often need to carry out expected utility computations for strategy profiles with small supports. Porter et al. [2008] observed that quite often games have Nash equilibria with small support, and proposed algorithms that explicitly search for such equilibria. In other algorithms for computing Nash equilibria such as Govindan-Wilson and simplicial subdivision, it is also quite often necessary to compute expected payoffs for mixed strategy profiles with small support.

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 $|C^{(a_i,i)}(\sigma_{-i})|$ remains the same, the ordering does affect the intermediate configurations $C_k^{(a_i)}$. We can use the following heuristic to try to minimize the number of intermediate configurations: sort the players in ascending order of the sizes of their projected action sets. This reduces the amount of work we do in earlier iterations of Algorithm 1, but does not change its overall complexity.

The Case of Symmetric Strategies in Symmetric AGG-$\emptyset$s

As described in Section 3.2.1, if a game is symmetric it can be represented as an AGG-$\emptyset$ with $A_i = \mathscr{A}$ for all $i \in N$. Given a symmetric game, we are often interested in computing expected utilities under *symmetric* mixed strategy profiles, where a mixed strategy profile σ is symmetric if $\sigma_i = \sigma_j \equiv \sigma_*$ for all $i, j \in N$. In Section 3.5.2 we will discuss algorithms that make use of expected utility computation under symmetric strategy profiles to compute a symmetric Nash equilibrium of symmetric games.

To compute the expected utility $V_{a_i}^i(\sigma_*)$, we could use the algorithm we proposed for general AGG-$\emptyset$s under arbitrary mixed strategies, which requires time polynomial in the size of the AGG-$\emptyset$. But we can gain additional computational speedup by exploiting the symmetry in the game and the strategy profile.

As before, we want to use Equation (3.4.5) to compute the expected utility, so the crucial task is again computing the probability distribution over projected configurations, $\Pr(c^{(a_i)}|\sigma^{(a_i)})$. Recall that $\sigma^{(a_i)} \equiv (a_i, \sigma_{-i}^{(a_i)})$. Define $\Pr(c^{(a_i)}|\sigma_*^{(a_i)})$ to be the distribution induced by $\sigma_{-i}^{(a_i)}$, the partial mixed strategy profile of players other than i, each playing the symmetric strategy $\sigma_*^{(a_i)}$. Once we have the distribution $\Pr(c^{(a_i)}|\sigma_*^{(a_i)})$, we can then compute the distribution $\Pr(c^{(a_i)}|\sigma^{(a_i)})$ straightforwardly by applying player i's strategy a_i. In the rest of this section we focus on computing $\Pr(c^{(a_i)}|\sigma_*^{(a_i)})$.

Define $\mathscr{S}(c^{(a_i)})$ to be the set containing all action profiles $a^{(a_i)}$ such that $\mathscr{C}(a^{(a_i)}) = c^{(a_i)}$. Since all agents have the same mixed strategies, each pure action profile in

$\mathscr{S}(c^{(a_i)})$ is equally likely, so for any $a^{(a_i)} \in \mathscr{S}(c^{(a_i)})$

$$\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right) = \left|\mathscr{S}(c^{(a_i)})\right| \Pr\left(a^{(a_i)}|\sigma_*^{(a_i)}\right), \tag{3.4.7}$$

$$\Pr\left(a^{(a_i)}|\sigma_*^{(a_i)}\right) = \prod_{\alpha \in \mathscr{A}^{(a_i)}} (\sigma_*^{(a_i)}(\alpha))^{c^{(a_i)}(\alpha)}. \tag{3.4.8}$$

The sizes of $\mathscr{S}(c^{(a_i)})$ are given by the multinomial coefficient

$$\left|\mathscr{S}\left(c^{(a_i)}\right)\right| = \frac{(n-1)!}{\prod_{\alpha \in \mathscr{A}^{(a_i)}} \left(c^{(a_i)}(\alpha)\right)!}. \tag{3.4.9}$$

Better still, using a Gray code technique we can avoid reevaluating these equations for every $c^{(a_i)} \in C^{(a_i)}$. Denote the configuration obtained from $c^{(a_i)}$ by decrementing by one the number of agents taking action $\alpha \in \mathscr{A}^{(a_i)}$ and incrementing by one the number of agents taking action $\alpha' \in \mathscr{A}^{(a_i)}$ as $c^{(a_i)'} \equiv c^{(a_i)}_{(\alpha \to \alpha')}$. Then consider the graph $H_{C^{(a_i)}}$ whose nodes are the elements of the set $C^{(a_i)}$, and whose directed edges indicate the effect of the operation $(\alpha \to \alpha')$. This graph is a regular triangular lattice inscribed within a $(|\mathscr{A}^{(a_i)}|-1)$-dimensional simplex. Having computed $Pr(c^{(a_i)}|\sigma_*^{(a_i)})$ for one node of $H_{C^{(a_i)}}$ corresponding to configuration $c^{(a_i)}$, we can compute the result for an adjacent node in $O(1)$ time,

$$\Pr\left(c^{(a_i)}_{(\alpha \to \alpha')}|\sigma_*^{(a_i)}\right) = \frac{\sigma_*^{(a_i)}(\alpha')c^{(a_i)}(\alpha)}{\sigma_*^{(a_i)}(\alpha)\left(c^{(a_i)}(\alpha')+1\right)} \Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right). \tag{3.4.10}$$

$H_{C^{(a_i)}}$ always has a Hamiltonian path (attributed to an unpublished result of Knuth by Klingsberg [1982]), so having computed $\Pr(c^{(a_i)}|\sigma_*^{(a_i)})$ for an initial $c^{(a_i)}$ using Equation (3.4.8), the results for all other projected configurations (nodes in $H_{C^{(a_i)}}$) can be computed by using Equation (3.4.10) at each subsequent step on the path. Generating the Hamiltonian path corresponds to finding a combinatorial Gray code for compositions; an algorithm with constant amortized running time is given by Klingsberg [1982]. Intuitively, it is easy to see that a simple, "lawnmower" Hamiltonian path exists for any lower-dimensional projection of $H_{C^{(a_i)}}$, with the only state required to compute the next node in the path being a direction value for each dimension.

Our algorithm for computing the distribution $\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right)$ is summarized in

Algorithm 3 Computing distribution $\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right)$ in a symmetric AGG-$\emptyset$

1. let $c^{(a_i)} = c_0^{(a_i)}$, where $c_0^{(a_i)}$ is the initial node of a Hamiltonian path of $H_{C^{(a_i)}}$.
2. compute $\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right)$ using Equation (3.4.7):

$$\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right) = \frac{(n-1)!}{\prod_{\alpha\in\mathscr{A}^{(a_i)}}\left(c^{(a_i)}(\alpha)\right)!}\prod_{\alpha\in\mathscr{A}^{(a_i)}}(\sigma_*^{(a_i)}(\alpha))^{c^{(a_i)}(\alpha)}.$$

3. While there are more configurations in $C^{(a_i)}$:
   (a) get the next configuration $c^{(a_i)}_{(\alpha\to\alpha')}$ in the Hamiltonian path, using Klingsberg's algorithm [Klingsberg, 1982].
   (b) compute $\Pr\left(c^{(a_i)}_{(\alpha\to\alpha')}|\sigma_*^{(a_i)}\right)$ using Equation (3.4.10):

$$\Pr\left(c^{(a_i)}_{(\alpha\to\alpha')}|\sigma_*^{(a_i)}\right) = \frac{\sigma_*^{(a_i)}(\alpha')c^{(a_i)}(\alpha)}{\sigma_*^{(a_i)}(\alpha)\left(c^{(a_i)}(\alpha')+1\right)}\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right).$$

   (c) let $c^{(a_i)} = c^{(a_i)}_{(\alpha\to\alpha')}$.
4. output $\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right)$ for all $c^{(a_i)} \in C^{(a_i)}$.

Algorithm 3. For computing expected utility, we again use Algorithm 2, except with Algorithm 3 replacing Algorithm 1 as the subroutine for computing the distribution $\Pr\left(c^{(a_i)}|\sigma_*^{(a_i)}\right)$.

Theorem 3.4.2. *Computation of the expected utility $V_{a_i}^i(\sigma_*)$ under a symmetric strategy profile for symmetric action-graph games using Equations* (3.4.5), (3.4.7), (3.4.8) *and* (3.4.10) *takes time $O(|\mathscr{A}| + |\nu(a_i)|\left|C^{(a_i)}(\sigma^{(a_i)})\right|)$.*

Proof. Projection to $\sigma_*^{(a_i)}$ takes $O(|\mathscr{A}|)$ time since the strategies are symmetric. Equation (3.4.5) has $\left|C^{(a_i)}(\sigma^{(a_i)})\right|$ summands. The probability for the initial configuration requires $O(n)$ time. Using Gray codes the computation of subsequent probabilities can be done in constant amortized time for each configuration. Since each look-up of the utility function takes $O(|\nu(a_i)|)$ time, the total complexity of the algorithm is $O(|\mathscr{A}| + |\nu(a_i)|\left|C^{(a_i)}(\sigma^{(a_i)})\right|)$. □

Algorithm 4 Computing the probability distribution $\Pr(c^{(a_i)}|\sigma^{(a_i)})$ in a k-symmetric AGG-$\emptyset$ under a k-symmetric mixed strategy profile $\sigma^{(a_i)}$.

1. Partition the players according to $\{N_1,\ldots,N_k\}$.
2. For each $l \in \{1,\ldots,k\}$, compute $\Pr(c^{(a_i)}|\sigma_{N_l}^{(a_i)})$, the probability distribution induced by $\sigma_{N_l}^{(a_i)}$, the partial strategy profile of players in N_l. Since $\sigma_{N_l}^{(a_i)}$ is symmetric, this can be computed efficiently using Algorithm 3 as discussed in Section 3.4.1.
3. Combine the k probability distributions together using Algorithm 1, resulting in the distribution $\Pr(c^{(a_i)}|\sigma^{(a_i)})$.

Note that this is faster than our dynamic programming algorithm for general AGG-$\emptyset$s under arbitrary strategies, whose complexity is $\Theta(n|\mathscr{A}|+n|\nu(a_i)|^2\left|C^{(a_i)}(\sigma^{(a_i)})\right|)$ by Theorem 3.4.1. In the usual case where the second term dominates the first, the algorithm for symmetric strategies is faster by a factor of $n|\nu(a_i)|$.

k-symmetric Games

We now move to a generalization of symmetry in games that we call k-symmetry.

Definition 3.4.3. *An AGG-$\emptyset$ is k-symmetric if there exists a partition $\{N_1,\ldots,N_k\}$ of N such that for all $l \in \{1,\ldots,k\}$, for all $i,j \in N_l$, $A_i = A_j$.*

Intuitively, k-symmetric AGG-$\emptyset$s represent games with k classes of identical agents, where agents within each class are identical. Note that all games are trivially n-symmetric. The Ice Cream Vendor game of Example 3.2.5 is a nontrivial k-symmetric AGG-$\emptyset$ with $k=3$.

Given a k-symmetric AGG-$\emptyset$ with partition $\{N_1,\ldots,N_k\}$, a mixed strategy profile σ is k-symmetric if for all $l \in \{1,\ldots,k\}$, for all $i,j \in N_l$, $\sigma_i = \sigma_j$. We are often interested in computing expected utility under k-symmetric strategy profiles. For example in Section 3.5.2 we will discuss algorithms that make use of such expected utility computations to find k-symmetric Nash equilibria in k-symmetric games. To compute expected utility under a k-symmetric mixed strategy profile, we can use a hybrid approach when computing the probability distribution over configurations, shown in Algorithm 4. Observe that this algorithm combines our specialized Algorithm 3 for handling symmetric games from Section 3.4.1 with the idea of running

Algorithm 1 on the joint mixed strategies of subgroups of agents discussed at the end of Section 3.4.1.

3.4.2 Computing Expected Payoff with AGG-FNs

Algorithm 1 cannot be directly applied to AGG-FNs with arbitrary f^p. First of all, projection of strategies does not work directly, because a player j playing an action $a_j \notin \nu(\alpha)$ could still affect $c^{(\alpha)}$ via function nodes. Furthermore, the general idea of using dynamic programming to build up the probability distribution by adding one player at a time does not work because for an arbitrary function node $p \in \nu(\alpha)$, each player would not be guaranteed to affect $c(p)$ independently. We could convert the AGG-FN to an AGG-$\emptyset$ in order to apply our algorithm, but then we would not be able to translate the extra compactness of AGG-FNs over AGG-$\emptyset$s into more efficient computation. In this section we identify two subclasses of AGG-FN for which expected utility can be efficiently computed. In Section 3.4.2 we show that when all function nodes belong to a restricted class of contribution-independent function nodes, expected utility can be computed in polynomial time. In Section 3.4.2 we reinterpret the expected utility problem as a Bayesian network inference problem, which can be computed in polynomial time if the resulting Bayesian network has bounded treewidth.

Contribution-Independent Function Nodes

Definition 3.4.4. *A function node p in an AGG-FN is* contribution-independent (CI) *if*

- $\nu(p) \subseteq \mathscr{A}$*, i.e., the neighbors of p are action nodes.*
- *There exists a commutative and associative operator $*$, and for each $\alpha \in \nu(p)$ an integer w_α, such that given an action profile $a = (a_1, \ldots, a_n)$, $c(p) = *_{i \in N : a_i \in \nu(p)} w_{a_i}$.*
- *The running time of each $*$ operation is bounded by a polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$. Furthermore, $*$ can be represented in space polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$.*

An AGG-FN is contribution-independent if all its function nodes are contribution-independent.

Note that it follows from this definition that $c(p)$ can be written as a function of $c^{(p)}$ by collecting terms: $c(p) \equiv f^p(c^{(p)}) = *_{\alpha \in \nu(p)}(*_{k=1}^{c(\alpha)} w_\alpha)$.

Simple aggregators can be represented as contribution-independent function nodes, with the $+$ operator serving as $*$, and $w_\alpha = 1$ for all α. The Coffee Shop game is thus an example of a contribution-independent AGG-FN. For the parity game in Example 3.2.8, $*$ is instead addition mod 2. An example of a non-additive CI function node arises in a perfect-information model of an (advertising) auction in which actions correspond to bid amounts [Thompson and Leyton-Brown, 2009]. Here we want $c(p)$ to represent the amount of the winning bid, and so we let w_α be the bid amount corresponding to action α, and $*$ be the max operator.

The advantage of contribution-independent AGG-FNs is that for all function nodes p, each player's strategy affects $c(p)$ independently. This fact allows us to adapt our algorithm to efficiently compute the expected utility $V_{a_i}^i(\sigma_{-i})$. For simplicity we present the algorithm for the case where we have one operator $*$ for all $p \in \mathscr{P}$, but our approach can be directly applied to games with different operators and w_α associated with different function nodes.

We define the *contribution* of action α to node $m \in \mathscr{A} \cup \mathscr{P}$, denoted $\delta_\alpha(m)$, as 1 if $m = \alpha$, 0 if $m \in \mathscr{A} \setminus \{\alpha\}$, and $*_{m' \in \nu(m)}(*_{k=1}^{\delta_\alpha(m')} w_\alpha)$ if $m \in \mathscr{P}$. Then it is easy to verify that given an action profile $a = (a_1, \ldots, a_n)$, $c(\alpha) = \sum_{j=1}^n \delta_{a_j}(\alpha)$ for all $\alpha \in \mathscr{A}$ and $c(p) = *_{j=1}^n \delta_{a_j}(p)$ for all $p \in \mathscr{P}$. Given that player i played a_i, and for all $\alpha \in \mathscr{A}$, we define the *projected contribution* of action α under a_i, denoted $\delta_\alpha^{(a_i)}$, as the tuple $(\delta_\alpha(m))_{m \in \nu(a_i)}$. Note that different actions α may have identical projected contributions under a_i. Player j's mixed strategy σ_j induces a probability distribution over j's projected contributions, $\Pr(\delta^{(a_i)} | \sigma_j) = \sum_{a_j : \delta_{a_j}^{(a_i)} = \delta^{(a_i)}} \sigma_j(a_j)$. 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 $\sigma_j^{(a_i)}$ in our algorithm for AGG-$\emptyset$s.

Algorithm 1 for computing the distribution $\Pr(c^{(a_i)} | \sigma)$ can be straightforwardly adopted to work with contribution-independent AGG-FNs. Whenever we apply player k's contribution $\delta_{a_k}^{(a_i)}$ to $c_{k-1}^{(a_i)}$, the resulting configuration $c_k^{(a_i)}$ is computed

componentwise as follows: $c_k^{(a_i)}(m) = \delta_{a_k}^{(a_i)}(m) + c_{k-1}^{(a_i)}(m)$ if $m \in \mathscr{A}$, and $c_k^{(a_i)}(m) = \delta_{a_k}^{(a_i)}(m) * c_{k-1}^{(a_i)}(m)$ if $m \in \mathscr{P}$.

To analyze the complexity of computing expected utility, it is necessary to know the representation size of a contribution-independent AGG-FN. For each function node p we need to specify $*$ and $(w_\alpha)_{\alpha \in \nu(p)}$ instead of f^p directly. Let $\|*\|$ denote the representation size of $*$. Then the total size of a contribution-independent AGG-FN is $O(\sum_{\alpha \in \mathscr{A}} |C^{(\alpha)}| + \|*\|)$. As discussed in Section 3.2.2, this size is not necessarily polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$; although when the conditions in Corollary 3.2.11 are satisfied, the representation size is polynomial.

Theorem 3.4.5. *Expected utility can be computed in time polynomial in the size of a contribution-independent AGG-FN. Furthermore, if the in-degrees of the action nodes are bounded by a constant and the sizes of ranges $|\mathscr{R}(f^p)|$ for all $p \in \mathscr{P}$ are bounded by a polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$, then expected utility can be computed in time polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$.*

Proof Sketch. Following similar complexity analysis as Theorem 3.4.1, if an AGG-FN is contribution-independent, expected utility $V_{a_i}^i(\sigma_{-i})$ can be computed in $O(n|\mathscr{A}||C^{(a_i)}|(T_* + |\nu(a_i)|))$ time, where T_* denotes the maximum running time of an $*$ operation. Since T_* is polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$ by Definition 3.4.4, the running time for computing expected utility is polynomial in the size of the AGG-FN representation. The second part of the theorem follows from a direct application of Corollary 3.2.11. □

For AGG-FNs whose function nodes are all simple aggregators, each player's set of projected contributions has size at most $|\nu(a_i) + 1|$, as opposed to $|\mathscr{A}|$ in the general case. This leads to a run time complexity of $O(n|\mathscr{A}| + n|\nu(a_i)|^2|C^{(a_i)}|)$, which is better than the complexity of the general case proved in Theorem 3.4.5. Applied to the Coffee Shop game, since $|C^{(\alpha)}| = O(n^3)$ and all function nodes are simple aggregators, our algorithm takes $O(n|\mathscr{A}| + n^4)$ time, which grows *linearly* in $|\mathscr{A}|$.

Beyond Contribution Independence

What about the case where not all function nodes are contribution-independent—is there anything we can do besides converting the AGG-FN into its induced AGG-$\emptyset$? It turns out that by reducing the problem of computing expected utility to a Bayesian network inference problem, we can still efficiently compute expected utilities for certain additional classes of AGG-FNs.

Bayesian networks compactly represent probability distributions exhibiting conditional independence structure (see, e.g., [Pearl, 1988, Russell and Norvig, 2003]). A Bayesian network is a DAG in which nodes represent random variables and edges represent direct probabilistic dependence. Each node X is associated with a conditional probability distribution (CPD) specifying the probability of each realization of random variable X conditional on the realizations of its parent random variables.

A key step in our approach for computing expected utility in AGG-FNs is computing the probability distribution over configurations $\Pr(c^{(a_i)}|\sigma^{(a_i)})$. If we treat each node m's configuration $c(m)$ as a random variable, then the distribution over configurations can be interpreted as the joint probability distribution over the set of random variables $\{c(m)\}_{m\in\nu(a_i)}$. Given an AGG-FN, a player i and an action $a_i \in A_i$, we can construct an *induced Bayesian network* $\mathscr{B}^i_{a_i}$:

- The nodes of $\mathscr{B}^i_{a_i}$ consist of (i) one node for each element of $\nu(a_i)$; (ii) one node for each neighbor of a function node belonging to $\nu(a_i)$; and (iii) one node for each neighbor of a function node added in the previous step, and so on until no more function nodes are added. Each of these nodes m represents the random variable $c(m)$. We further introduce another kind of node: (iv) n nodes $\sigma_1,\ldots,\sigma_n$, representing each player's mixed strategy. The domain of each random variable σ_i is A_i.
- The edges of $\mathscr{B}^i_{a_i}$ are constructed by keeping all edges that go into the function nodes that are included in $\mathscr{B}$, ignoring edges that go into action nodes. Furthermore for each player j, we create an edge from σ_j to each of j's actions $a_j \in A_j$.
- The conditional probability distribution (CPD) at each function node p is just the deterministic function f^p. The CPD at each action node α' is a deterministic function that returns the number of its parents (observe that these

are all mixed strategy nodes) that take the value α'. Mixed strategy nodes have no incoming edges; their (unconditional) probability distributions are the mixed strategies of the corresponding players, except for player i, whose node σ_i takes the deterministic value a_i.

It is straightforward to verify that $\mathscr{B}^i_{a_i}$ is a DAG, and that the joint distribution on random variables $\{c(m)\}_{m \in v(\alpha)}$ is exactly the distribution over configurations $\Pr(c^{(a_i)} | (a_i, \sigma_{-i}^{(a_i)}))$. This joint distribution can then be computed using a standard algorithm such as clique tree propagation or variable elimination. The running times of such algorithms are worst-case exponential; however, for Bayesian networks with bounded tree-width, their running times are polynomial.

Further speedups are possible at nodes in the induced Bayesian network that correspond to action nodes and contribution-independent function nodes. The deterministic CPDs at such nodes can be formulated using independent contributions from each player's strategy. This is an example of *causal independence* structure in Bayesian networks studied by Heckerman and Breese [1996] and Zhang and Poole [1996], who proposed different methods for exploiting such structure to speed up Bayesian network inference. Such methods share the common underlying idea of decomposing the CPDs into independent contributions, which is intuitively similar to our approach in Algorithm 1.[6]

3.4.3 Computing Expected Payoff with AGG-FNAs

Due to the linearity of expectation, the expected utility of i playing an action a_i with an additive utility function with coefficients $(\lambda_m)_{m \in v(a_i)}$ is

$$V^i_{a_i}(\sigma_{-i}) = \sum_{m \in v(a_i)} \lambda_m E[c(m) | a_i, \sigma_{-i}], \tag{3.4.11}$$

where $E[c(m)|a_i, \sigma_{-i}]$ is the expected value of $c(m)$ given the strategy profile (a_i, σ_{-i}). Thus we can compute these expected values for each $m \in v(a_i)$, then sum them up as in Equation (3.4.11) to get the expected utility. If m is an action node, then $E[c(m)|a_i, \sigma_{-i}]$ is the expected number of players that chose m, which is

[6] This approach of reducing expected utility computation to Bayesian network inference is further developed in Chapters 5 and 6, for Temporal Action-Graph Games and Bayesian Action-Graph Games respectively.

$\sum_{i \in N} \sigma_i(m)$. The more interesting case is when m is a function node. Recall that $c(m) \equiv f^m(c^{(m)})$ where $c^{(m)}$ is the configuration over the neighbors of m. We can write the expected value of $c(m)$ as

$$E[c(m)|a_i, \sigma_{-i}] = \sum_{c^{(m)} \in C^{(m)}} f^m(c^{(m)}) \Pr(c^{(m)}|a_i, \sigma_{-i}). \tag{3.4.12}$$

This has the same form as Equation (3.4.5) for the expected utility $V^i_{a_i}(\sigma_{-i})$, except that we have f^m instead of u^α. Thus our results for the computation of Equation (3.4.5) also apply here. That is, if the neighbors of m are action nodes and/or contribution-independent function nodes, then $E[c(m)|a_i, \sigma_{-i}]$ can be computed in polynomial time.

Theorem 3.4.6. *Suppose u^α is represented as an additive utility function in a given AGG-FNA. If each of the neighbors of α is either (i) an action node, or (ii) a function node whose neighbors are action nodes and/or contribution-independent function nodes, then the expected utility $V^i_\alpha(\sigma_{-i})$ can be computed in time polynomial in the size of the representation. Furthermore, if the in-degrees of the neighbors of α are bounded by a constant, and the sizes of ranges $|\mathscr{R}(f^p)|$ for all $p \in \mathscr{P}$ are bounded by a polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$, then the expected utility can be computed in time polynomial in n, $|\mathscr{A}|$ and $|\mathscr{P}|$.*

It is straightforward to verify that our AGG-FNA representations of polymatrix games, congestion games, player-specific congestion games and the game in Example 3.3.4 all satisfy the conditions of Theorem 3.4.6.

3.5 Computing Sample Equilibria with AGGs

In this section we consider some theoretical and practical applications of our expected utility algorithm. In Section 3.5.1 we analyze the complexity of finding a sample ε-Nash equilibrium in an AGG and show that it is PPAD-complete. In Section 3.5.2 we extend our expected utility algorithm to the computation of payoff Jacobians, which is a key step in several algorithms for computing ε-Nash equilibria, including the Govindan-Wilson algorithm. In Section 3.5.3 we show that it can also speed up the simplicial subdivision algorithm, and in Section 3.5.4 we show that it can be used to find a correlated equilibrium in polynomial time.

3.5.1 Complexity of Finding a Nash Equilibrium

In this section we consider the complexity of finding a Nash equilibrium of an AGG. As discussed in Section 2.2.1, since a Nash equilibrium for a game of more that two players may require irrational numbers in the probabilities, for practical computation it is necessary to consider approximations to Nash equilibria. Here we consider the frequently-used notion of ε-Nash equilibrium as defined in Definition 2.2.3. Recall from Section 2.2 that for any game representation, its NASH problem is defined to be the problem of finding an ε-Nash equilibrium of a game encoded in that representation, for some ε given as part of the input. Also recall from Section 2.2.1 that the NASH problem for n-player normal-form games with $n \geq 2$ is complete for the complexity class PPAD, which is contained in NP but not known to be in P. Turning to compact representations, recall from Section 2.2.2 and in particular Theorem 2.2.4 that the complexity of computing expected utility plays a vital role in the complexity of finding an ε-Nash equilibrium. By leveraging Algorithm 1, we are able to apply Theorem 2.2.4 to AGGs.

Corollary 3.5.1. *The complexity of NASH for AGG-$\emptyset$s is PPAD-complete.*

Remark. It may not be clear why this would be surprising or encouraging; indeed, the PPAD-hardness part of the claim is neither. However, the PPAD-membership part of the claim is a positive result. Specifically, it implies that the problem of finding a Nash equilibrium in an AGG-$\emptyset$ can be reduced to the problem of finding a Nash equilibrium in a two-player normal-form game with size polynomial in the size of the AGG-$\emptyset$. This is in contrast to the normal form representation of the original game, which can be exponentially larger than the AGG-$\emptyset$. In other words, if we instead try to solve for a Nash equilibrium using the normal form representation of the original game, we would face a PPAD-complete problem with an input exponentially larger than the AGG-$\emptyset$ representation.

Proof sketch. The first condition of Theorem 2.2.4—polynomial type—is satisfied by all AGG variants, since action sets are represented explicitly. We first show that the problem belongs to PPAD, by constructing a circuit that computes expected utility and satisfies the second condition of Theorem 2.2.4.[7] Recall that our expected utility algorithm consists of Equation (3.4.4), then Algorithm 1, and finally Equa-

tion (3.4.5). Equations (3.4.4) and (3.4.5) can be straightforwardly translated into arithmetic circuits using addition and multiplication nodes. Algorithm 1 involves for loops that cannot be directly translated to an arithmetic circuit, but we observe that we can unroll the for loops and still end up with a polynomial number of operations. The resulting circuit resembles a lattice with n levels; at the k-th level there are $|C_k^{(a_i)}|$ addition nodes. Each addition node corresponds to a configuration $c_k^{(a_i)} \in C_k^{(a_i)}$, and calculates $P_k[c_k^{(a_i)}]$ as in iteration k of Algorithm 1. Also there are $|A_k^{(a_i)}|$ multiplication nodes for each $c_k^{(a_i)}$, in order to carry out the multiplications in iteration k of Algorithm 1.

To show PPAD-hardness, we observe that an arbitrary graphical game can be encoded as an AGG-$\emptyset$ without loss of compactness (see Section 3.2.1). Thus the problem of finding a Nash equilibrium in a graphical game can be reduced to the problem of finding a Nash equilibrium in an AGG-$\emptyset$. Since finding a Nash equilibrium in a graphical game is known to be PPAD-hard, finding a Nash equilibrium in an AGG-$\emptyset$ is PPAD-hard. □

For AGG-FNs that satisfy the conditions for Theorem 3.4.5 or AGG-FNAs that satisfy Theorem 3.4.6, similar arguments apply, and we can prove PPAD-completeness for those subclasses of games if we make the reasonable assumption that the operator $*$ used to define the CI function nodes can be implemented as an arithmetic circuit of polynomial length that satisfies the second condition of Theorem 2.2.4.

3.5.2 Computing a Nash Equilibrium: The Govindan-Wilson Algorithm

Now we move from the theoretical to the practical. The PPAD-hardness result of Corollary 3.5.1 implies that a polynomial-time algorithm for Nash equilibrium is unlikely to exist, and indeed known algorithms for identifying sample Nash equilibria have worst-case exponential running times. Nevertheless, we will show that our dynamic programming algorithm for expected utility can be used to achieve exponential speedups in such algorithms, as well as an algorithm for computing a

[7] Observe that the second condition in Theorem 2.2.4 implies that the expected utility algorithm must take polynomial time; however, some polynomial algorithms (e.g., those that rely on division) do not satisfy this condition.

sample correlated equilibrium. Specifically, we use a *black-box* approach as discussed in Section 2.2.2.

First we consider Govindan and Wilson's [2003] global Newton method, a state-of-the-art method for finding mixed-strategy Nash equilibria in multi-player games. Recall from Sections 2.2.1 and 2.3 that a bottleneck of the algorithm is the computation of payoff Jacobians, and the Gametracer package provides a black-box implementation of the global Newton method that allows one to directly plug in representation-specific subroutines for this task.

The payoff Jacobian is defined to be the Jacobian of the function $V : \Sigma \to \mathbb{R}^{\sum_i |A_i|}$, whose (i, α_i)-th component is the expected utility $V^i_{\alpha_i}(\sigma_{-i})$. The corresponding Jacobian at σ is a $(\sum_i |A_i|) \times (\sum_i |A_i|)$ matrix with entries

$$\frac{\partial V^i_{a_i}(\sigma_{-i})}{\partial \sigma_{i'}(a_{i'})} \equiv \nabla V^{i,i'}_{a_i,a_{i'}}(\overline{\sigma}) \tag{3.5.1}$$

$$= \sum_{\overline{a} \in \overline{A}} u(a_i, \mathscr{C}(a_i, a_{i'}, \overline{a})) \Pr(\overline{a}|\overline{\sigma}) \tag{3.5.2}$$

if $i \neq i'$, and zero otherwise. Here an overbar is shorthand for the subscript $-\{i, i'\}$ where $i \neq i'$ are two players; e.g., $\overline{a} \equiv a_{-\{i,i'\}}$. The rows of the matrix are indexed by i and a_i while the columns are indexed by i' and $a_{i'}$. Given entry $\nabla V^{i,i'}_{a_i,a_{i'}}(\overline{\sigma})$, we call a_i its *primary action node*, and $a_{i'}$ its *secondary action node*.

We note that efficient computation of the payoff Jacobian is important for more than simply Govindan and Wilson's global Newton method. For example, recall from Section 2.2.1 that the iterated polymatrix approximation (IPA) method [Govindan and Wilson, 2004] has the same computational problem at its core.

Computing the Payoff Jacobian

Now we consider how the payoff Jacobian may be computed. Equation (3.5.2) shows that the $\nabla V^{i,i'}_{a_i,a_{i'}}(\overline{\sigma})$ element of the Jacobian can be interpreted as the expected utility of agent i when she takes action a_i, agent i' takes action $a_{i'}$, and all other agents use mixed strategies according to $\overline{\sigma}$. So a straightforward—and quite effective—approach is to use our expected utility algorithm to compute each entry of the Jacobian.

However, the Jacobian matrix has certain extra structure that allows us to achieve

further speedup. For example, observe that some entries of the Jacobian are identical. If two entries have the same primary action node α, then they are expected payoffs on the same utility function u^α, and so have the same values if their induced probability distributions over $C^{(\alpha)}$ are the same. We need to consider two cases:

1. The two entries come from the same row of the Jacobian, say player i's action a_i. There are two sub-cases to consider:
   (a) The columns of the two entries belong to the same player j, but different actions a_j and a'_j. If $a_j^{(a_i)} = {a'}_j^{(a_i)}$, i.e., a_j and a'_j both project to the same projected action in a_i's projected action graph,[8] then $\nabla V^{i,j}_{a_i,a_j} = \nabla V^{i,j}_{a_i,a'_j}$. This implies that when $a_j, a'_j \notin \nu(a_i)$, $\nabla V^{i,j}_{a_i,a_j} = \nabla V^{i,j}_{a_i,a'_j}$.
   (b) The columns of the entries correspond to actions of different players. We observe that for all j and a_j such that $\sigma^{(a_i)}(a_j^{(a_i)}) = 1$, $\nabla V^{i,j}_{a_i,a_j}(\overline{\sigma}) = V^i_{a_i}(\sigma_{-i})$. As a special case, if $A_j^{(a_i)} = \{\emptyset\}$, i.e., agent j does not affect i's payoff when i plays a_i, then for all $a_j \in A_j$, $\nabla V^{i,j}_{a_i,a_j}(\overline{\sigma}) = V^i_{a_i}(\sigma_{-i})$.
2. If a_i and a_j correspond to the same action node α (but owned by agents i and j respectively), thus sharing the same payoff function u^α, then $\nabla V^{i,j}_{a_i,a_j} = \nabla V^{j,i}_{a_j,a_i}$. Furthermore, if there exist $a'_i \in A_i, a'_j \in A_j$ such that $a_i'^{(\alpha)} = a_j'^{(\alpha)}$ (or $\delta^{(\alpha)}_{a'_i} = \delta^{(\alpha)}_{a'_j}$ for contribution-independent AGG-FNs), then $\nabla V^{i,j}_{a_i,a'_j} = \nabla V^{j,i}_{a_j,a'_i}$.

A consequence of 1(a) is that any Jacobian of an AGG has at most $\sum_i \sum_{a_i \in A_i} (n-1)(\nu(a_i)+1)$ distinct entries. For AGGs with bounded in-degree, this is $O(n\sum_i |A_i|)$. For each set of identical entries, we only need to do the expected utility computation once. Even when two entries in the Jacobian are not identical, we can exploit the similarity of the projected strategy profiles (and thus the similarity of the induced distributions) between entries, reusing 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.

[8] For contribution-independent AGG-FNs, the condition becomes $\delta^{(a_i)}_{a_j} = \delta^{(a_i)}_{a'_j}$, i.e., a_j and a'_j have the same projected contribution under a_i.

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

Now suppose we fix the row (i, a_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 AGG-$\emptyset$s, we can exploit this similarity by computing the distribution $\Pr(c_{n-1}|\sigma_{-i}^{(a_i)})$, then for each $j \neq i$, we "undo" j's mixed strategy to get the distribution induced by $\sigma_{-\{i,j\}}$, by treating distributions $\Pr(c_{n-1}|\sigma_{-i}^{(a_i)})$ and σ_j as coefficients of polynomials and computing their quotient using long division. (See Section 2.3.5 of [Jiang, 2006] for a more detailed discussion of interpreting distributions over configurations as polynomials.)

Finding equilibria of symmetric and k-symmetric games

Nash proved [1951] that all finite symmetric games have at least one symmetric Nash equilibrium. The Govindan-Wilson algorithm can be adapted to find symmetric Nash equilibria in symmetric AGG-$\emptyset$s. The modified algorithm now operates in the space of symmetric mixed strategy profiles $\Sigma_* = \varphi(\mathscr{A})$, and follows a path of symmetric equilibria of perturbed symmetric games to a symmetric equilibrium of the unperturbed game. A key step of the algorithm is the computation of the Jacobian of the function $V : \Sigma_* \to \mathbb{R}^{|\mathscr{A}|}$, whose α-th entry $V_\alpha(\sigma_*)$ is the expected utility of one player choosing α while the others play mixed strategy σ_*. This Jacobian at σ_* is a $|\mathscr{A}| \times |\mathscr{A}|$ matrix whose entry at row α and column α' is $n-1$ multiplied by the expected utility of a player choosing action α, when another player is choosing action α' and the rest of the players play mixed strategy σ_*. Such an entry can be efficiently computed using the techniques for symmetric expected utility computation discussed in Section 3.4.1, which are faster than our expected utility algorithm for general AGGs. Techniques discussed in the current section can further be used to speed up the computation of Jacobians in the symmetric case. In particular, it is straightforward to check that the Jacobian has at most $\sum_{\alpha \in \mathscr{A}}(\nu(\alpha)+1) = O(|E|)$ identical entries, where E is the set of edges of the action graph.

A straightforward corollary of Nash's [1951] proof is that any k-symmetric

AGG-$\emptyset$ has at least one k-symmetric Nash equilibrium. For each equivalence class ℓ of the players let Σ_*^ℓ denote the set of symmetric strategy profiles for N_ℓ, and let A^ℓ denote the set of actions of a player in N_ℓ. Relying on similar arguments as above, we can adapt the Govindan-Wilson algorithm to find k-symmetric equilibria in k-symmetric AGG-$\emptyset$s. The bottleneck is the computation of the Jacobian of the function $V : \prod_\ell \Sigma_*^\ell \to \mathbb{R}^{\Sigma_\ell |A^\ell|}$, whose (ℓ, α)-th entry is the utility of a player in N_ℓ playing action α, while the others play according to the given k-symmetric strategy profile $(\sigma_*^1, \ldots, \sigma_*^k)$. The entry at row ℓ, α and column ℓ', α' of the Jacobian matrix is equal to $(|N_{\ell'}| - \mathbf{1}_{\ell=\ell'})$ multiplied by the expected utility of a player in N_ℓ choosing action α, when another player in N'_ℓ is choosing action α' and the others play according to the given k-symmetric strategy profile. Such expected utilities can be efficiently computed using the techniques discussed in Section 3.4.1.

3.5.3 Computing a Nash Equilibrium: The Simplicial Subdivision Algorithm

Another algorithm for computing a sample Nash equilibrium is van der Laan, Talman & van der Heyden's [1987] simplicial subdivision algorithm. Recall from Section 2.2.1 that one of the bottlenecks is the computation of labels of a given subsimplex in a simplicial subdivision of Σ, which in turn depends on computation of expected utilities under mixed strategy profiles. The GAMBIT package [McKelvey et al., 2006] provides an implementation of the simplicial subdivision algorithm for the normal form. We adapted this code into a black-box implementation that allows one to plug in representation-specific subroutines for expected utility computation. Combining this with an implementation of our AGG-based Algorithm 2 is then sufficient for an exponential speedup compared to the normal-form-based implementation of the simplicial subdivision algorithm. An advantage of the black-box implementation is that this is useful for other representations besides AGGs; e.g., in Chapter 6 we are able to use this for computing sample Bayes-Nash equilibria for Bayesian Action-Graph Games.

3.5.4 Computing a Correlated Equilibrium

In Section 2.2.7 we gave an overview of the literature on the computation of a sample correlated equilibrium. In summary, Papadimitriou and Roughgarden [2008] proposed a polynomial-time algorithm for computing a sample correlated equilibrium given a game representation with polynomial type and a polynomial-time subroutine for computing expected utility under mixed strategy profiles. Recently, Stein et al. [2010] showed that Papadimitriou and Roughgarden's algorithm can fail to find an exact correlated equilibrium, and presented a slight modification of the algorithm that efficiently computes an ε-correlated equilibrium. (An ε-correlated equilibrium is an approximation of the correlated equilibrium solution concept, where ε measures the extent to which the incentive constraints for correlated equilibrium are violated.) Incorporating this fix, we have the following.

Theorem 3.5.2 ([Papadimitriou and Roughgarden, 2008])**.** *If a game representation has polynomial type, and has a polynomial algorithm for computing expected utility, then an ε-correlated equilibrium can be computed in time polynomial in* $\log \frac{1}{\varepsilon}$ *and the representation size.*

In Chapter 7 we present a modified version of Papadimitriou and Roughgarden's algorithm that is able to compute an exact correlated equilibrium in polynomial time.

Theorem 3.5.3 (Restatement of Theorem 7.4.5; also [Jiang and Leyton-Brown, 2011])**.** *If a game representation has polynomial type, and has a polynomial algorithm for computing expected utility, then a correlated equilibrium can be computed in time polynomial in the representation size.*

The second condition in both theorems involve the computation of expected utility. As a direct corollary of Theorem 3.5.3 and Theorem 3.4.1, there exists a polynomial algorithm for computing an exact correlated equilibrium given an AGG-$\emptyset$.

Corollary 3.5.4. *Given a game represented as an AGG-$\emptyset$, an exact correlated equilibrium can be computed in time polynomial in the size of the AGG-$\emptyset$.*

Similarly, for AGG-FNs and AGG-FNAs for which the expected utility problem can be solved in polynomial time (see Theorems 3.4.5 and 3.4.6), correlated equilibria can be computed in polynomial time.

3.6 Experiments

Although our theoretical results show that there are significant benefits to working with AGGs, they might leave the reader with two worries. First, the reader might be concerned that while AGGs offer asymptotic computational benefits, they might not be practically useful. Second, even if convinced about the usefulness of AGGs, the reader might want to know the size of problems that can be tackled by the computational tools we have developed so far. We address both of these worries in this section, by reporting on the results of extensive computational experiments. Specifically, we compare the performance of the AGG representation and our AGG-based algorithms against normal-form-based solutions using the (highly optimized) GameTracer package [Blum et al., 2002]. As benchmarks, we used AGG and normal-form representations of instances of Coffee Shop games, Job Market games, and symmetric AGG-$\emptyset$s on random graphs. We compared the representation sizes of AGG and normal-form representations, and compared their performance resulting from using these representations to compute expected utility, to compute Nash equilibria using the Govindan-Wilson algorithm, and to compute Nash equilibria using the simplicial subdivision algorithm. Finally, we show how sample equilibria of these games can be visualized on action graphs.

3.6.1 Software Implementation and Experimental Setup

We implemented our algorithms in a freely-available software package, in order to make it easy for other researchers to use AGGs to model problems of interest. Our software is capable of:

- reading in a description of an AGG;
- computing expected utility and Jacobian given mixed strategy profile;
- computing Nash equilibria by adapting GameTracer's [Blum et al., 2002] implementation of Govindan and Wilson's [2003] global Newton method;

and

- computing Nash equilibria by adapting GAMBIT's [McKelvey et al., 2006] implementation of the simplicial subdivision algorithm [van der Laan et al., 1987].

We extended GAMUT [Nudelman et al., 2004], a suite of game instance generators, by implementing generators of instances of AGGs including Ice Cream Vendor games (Example 3.2.5), Coffee Shop games (Example 3.2.7), Job Market games (Example 3.3.1) and symmetric AGG-∅s on a random action graph with random payoffs. Finally, with Damien Bargiacchi, we also developed a graphical user interface for creating and editing AGGs. More details on these as well as software implementations of other algorithms from this thesis are given in Appendix A. All of our software is freely available at http://agg.cs.ubc.ca.

When using Coffee Shop games in our experiments, we set payoffs randomly in order to test on a wide set of utility functions. For the visualization of equilibria in Section 3.6.7 we set the Coffee Shop game utility functions to be

$$u^{\alpha}(c(\alpha), c(p'_{\alpha}), c(p''_{\alpha})) = 20 - [c(\alpha)]^2 - c(p'_{\alpha}) - \log(c(p''_{\alpha}) + 1),$$

where p'_{α} is the function node representing the number of players choosing adjacent locations and p''_{α} is the function node representing the number of players choosing other locations.

When using Job Market games in our experiments, we set the utility functions to be

$$u^{\alpha}(c^{(\alpha)}) = \frac{R_{\alpha}}{c(\alpha) + \sum_{\alpha' \in v(\alpha) - \{\alpha\}} 0.1 c(\alpha')} - K_{\alpha},$$

with R_{α} set to $2, 4, 6, 8, 10$ and K_{α} set to $1, 2, 3, 4, 5$ for the five levels from high school to PhD.

When using Ice Cream Vendor games for the visualization of equilibria in Section 3.6.7 we set the utilities so that for a player i choosing action α, each vendor choosing a location $\alpha' \in v(\alpha)$ contributes $w_f w_l$ utility to i. w_f is -1 when α' has the same food type as α, and 0.8 otherwise. w_l is 1 when α' and α correspond to the same location, and 0.6 when they correspond to different (but neighboring) locations. In other words, there is a negative effect from players choosing the same

food type, and a weaker positive effect from players choosing a different food type. Furthermore, effects from neighboring locations are weaker than effects from the same location.

All our experiments were performed using a computer cluster consisting of 55 machines with dual Intel Xeon 3.2GHz CPUs, 2MB cache and 2GB RAM, running Suse Linux 10.1.

3.6.2 Representation Size

First, we compared the representation sizes of AGG-FNs and their induced normal forms. For each game instance we counted the number of payoff values that needed to be stored.

We first looked at 5×5 block Coffee Shop games, varying the number of players. Figure 3.9 (left) 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 became impractical. The size of the AGG representation grew polynomially with respect to n. As we can see from Figure 3.9 (right), even for a game instance with 80 players, the AGG-FN representation stored only about 2 million numbers. In contrast, the corresponding normal form representation would have had to store 1.2×10^{115} numbers.

We then fixed the number of players at 4 and varied the number of actions; for ease of comparison we fixed the number of columns at 5 and only changed the number of rows. Recall from Section 3.2.2 that the representation size of Coffee Shop games—expressed both as AGGs and in the normal form—depends only on the number of players and number of actions, but not on the shape of the region. (Recall that the number of actions is $B+1$, where B is the total number of blocks.) Figure 3.9 (left) shows a log-scale plot of the number of payoff values versus the number of actions, and Figure 3.9 (right) gives a plot for just the AGG-FN representation. 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. For a Coffee Shop game with 4 players on an 80×5 grid, the AGG-FN representation stores only about 8000 numbers, whereas the normal form

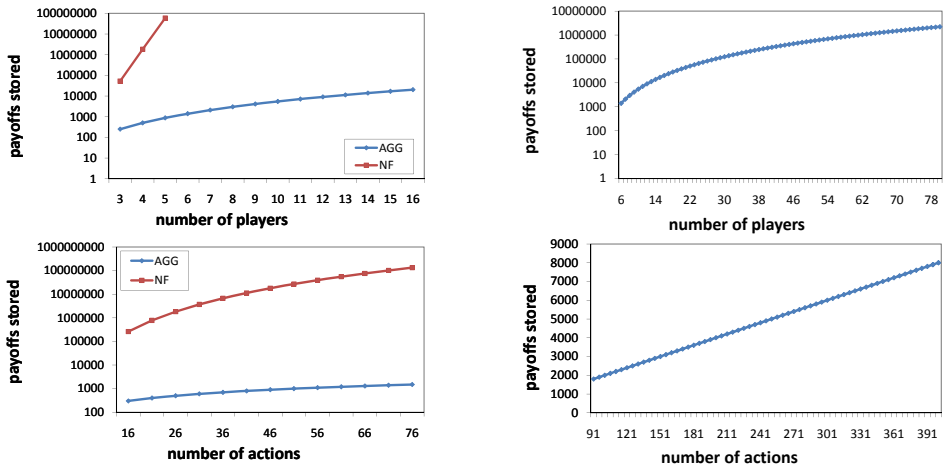

Figure 3.9: Representation sizes of coffee shop games. Top left: 5×5 grid with 3 to 16 players (log scale). Top right: AGG only, 5×5 grid with up to 80 players (log scale). Bottom left: 4-player $r \times 5$ grid, r varying from 3 to 15 (log scale). Bottom right: AGG only, up to 80 rows.

representation would have to store 1.0×10^{11} numbers.

We also tested on Job Market games from Example 3.3.1, which have 13 actions. We varied the number of players from 3 to 24. The results are similar, as shown in Figure 3.11 (left). This is consistent with our theoretical observation that the sizes of normal form representations grow exponentially in n while the sizes of AGG representations grow polynomially in n.

3.6.3 Expected Utility Computation

We tested the performance of our dynamic programming algorithm for computing expected utilities in AGG-FNs against GameTracer's normal-form-based algorithm for computing expected utilities. For each game instance, we generated 1000 random strategy profiles with full support, and measured the CPU (user) time spent computing $V_{a_n}^n(\sigma_{-n})$ under these strategy profiles. Then we divided this measurement by 1000 to obtain the average CPU time.

We first looked at Coffee Shop games of different sizes. We fixed the size of blocks at 5×5 and varied the number of players. Figure 3.10 shows plots of the results. For very small games the normal-form-based algorithm is faster due

to its smaller bookkeeping overhead; as the number of players grows larger, our AGG-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. Meanwhile, our AGG-based algorithm scaled to much larger numbers of players, averaging about a second to compute an expected utility for an 80-player Coffee Shop game.

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

We also 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 utilities for the Coffee Shop game instances mentioned above, while our AGG-based algorithm required about 20% of its full-support running times.

We also tested on Job Market games, varying the numbers of players. The results are shown in Figure 3.11 (right). The normal-form-based implementation ran out of memory for more than 6 players, while the AGG-based implementation averaged about a quarter of a second to compute expected utility in a 24-player game.

3.6.4 Computing Payoff Jacobians

We ran similar experiments to investigate the computation of payoff Jacobians. As discussed in Section 3.5.2, 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 entries. In Section 3.5.2 we discussed methods that exploit the structure of the Jacobian to further speed up the computation. GameTracer's normal-form-based implementation also exploits the structure of the Jacobian by reusing partial results of expected payoff computations. When comparing our AGG-based

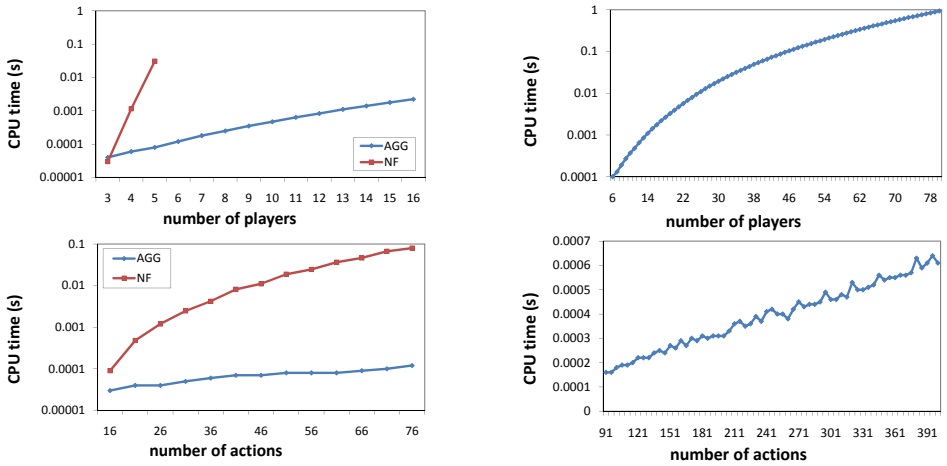

Figure 3.10: Running times for payoff computation in the Coffee Shop game. Top left: 5×5 grid with 3 to 16 players. Top right: AGG only, 5×5 grid with up to 80 players. Bottom left: 4-player $r \times 5$ grid, r varying from 3 to 15. Bottom right: AGG only, up to 80 rows.

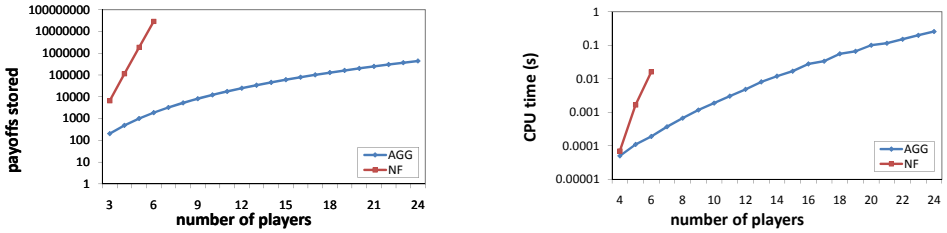

Figure 3.11: Job Market games, varying numbers of players. Left: comparing representation sizes. Right: running times for computing 1000 expected utilities.

Jacobian algorithm (as described in Section 3.5.2) to GameTracer's implementation, we observed results very similar to those for computing expected payoffs: our implementation scaled polynomially in n while GameTracer scaled exponentially in n. We instead focus on the question of how much speedup the methods in Section 3.5.2 provided, by comparing our algorithm in Section 3.5.2 against the algorithm that computes expected payoffs (using our AGG-based algorithm described in Section 3.4) for each of the Jacobian's entries. We tested on Coffee Shop games on a 5×5 grid with 3 to 10 players, as well as Coffee Shop games with 4

players, 5 columns and varying numbers of rows. For each instance of the game we randomly generated 100 strategy profiles with partial support. For each of these game instances, our algorithm as described in Section 3.5.2 was consistently about 50 times faster than computing expected payoffs for each of the Jacobian's entries. This confirms that the methods discussed in Section 3.5.2 provide significant speedup for computing payoff Jacobians.

3.6.5 Finding a Nash Equilibrium Using Govindan-Wilson

Now we show experimentally that the speedup we achieved for computing Jacobians using the AGG representation led to a speedup in the Govindan-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 follow the same path and return exactly the same Nash equilibrium.

Again, we tested the two algorithms on Coffee Shop games of varying sizes: first we fixed the sizes 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 ran the two versions of the Govindan-Wilson algorithm. Although the algorithm can (sometimes) find more than one equilibrium, we stopped both versions of the algorithm after one equilibrium was found. Since the running time of the Govindan-Wilson algorithm is very sensitive to the initial perturbation, for each game instance the running times with different initial perturbations had large variance. To control for this, for each initial perturbation we looked at the *ratio* of running times between the normal-form implementation and the AGG implementation (i.e., a ratio greater than 1 means the AGG implementation ran more quickly than the normal form implementation). We present the results in Figure 3.12 (left). We see that as the size of the games grew (either in the number of players or in the number of actions), the speedup of

the AGG implementation over that of the normal-form implementation increased. The normal-form implementation ran out of memory for game instances with more than 5 players, preventing us from reporting ratios above $n = 5$. Thus, we ran the AGG-based implementation alone on game instances with larger numbers of players, giving the algorithm a one-day cutoff time. As shown by the log-scale boxplot of CPU times in Figure 3.12 (top right), for game instances with up to 12 players, the algorithm terminated within one day for most initial perturbations. A normal form representation of such a game would have needed to store 7.0×10^{15} numbers. Figure 3.12 (bottom right) shows a boxplot of the CPU times for the AGG-based implementation, varying the number of actions while fixing the number of players at 4. For game instances with up to 49 actions (a 4×12 grid plus one action for not entering the market), the algorithm terminated within an hour.

We also tested on Job Market games with varying numbers of players. The results are shown in Figure 3.13. For the game instance with 6 players, the AGG-based implementation was about 100 times faster than the normal-form-based implementation. While the normal-form-based implementation ran out of memory for Job Market games with more than 6 players, the AGG-based implementation was able to solve games with 16 players in an average of 24 minutes.

3.6.6 Finding a Nash Equilibrium Using Simplicial Subdivision

As discussed in Section 3.5.3, we can speed up the normal-form-based simplicial subdivision algorithm by replacing the subroutine that computes expected utility by our AGG-based algorithm. We have done so to GAMBIT's implementation of simplicial subdivision. As with the Govindan-Wilson algorithm, from a given starting point both the original version of simplicial subdivision and our AGG version follow a deterministic path to determine exactly the same equilibrium. Thus, all performance differences are due to the choice of representation. We compared the performance of AGG-based simplicial subdivision against normal-form-based simplicial subdivision on instances of Coffee Shop games as well as instances of randomly-generated symmetric AGG-$\emptyset$s on small world graphs. We always started from the mixed strategy profile in which each player gives equal probability to each of her actions.

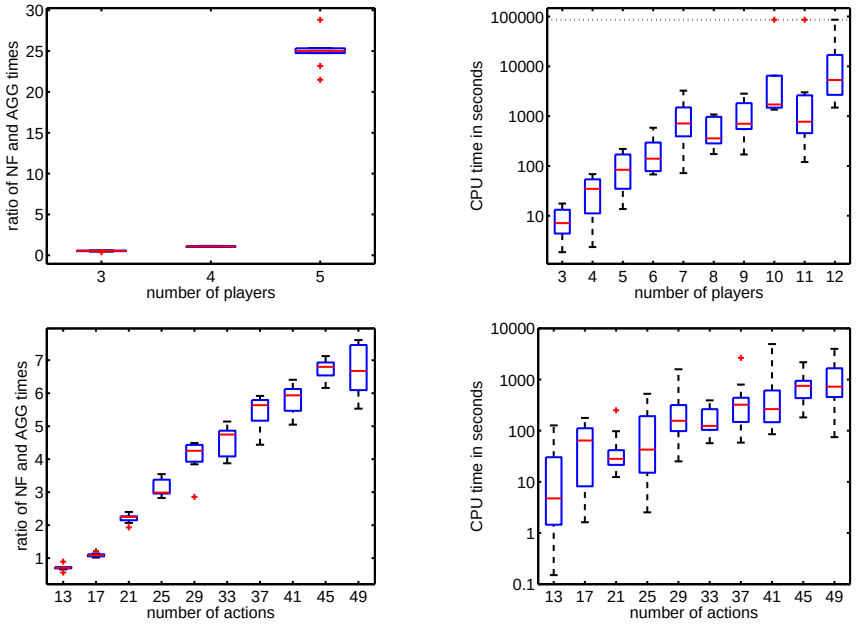

Figure 3.12: Govindan-Wilson algorithm; Coffee Shop game. Top row: 4×4 grid, varying number of players. Bottom row: 4-player $r \times 4$ grid, r varying from 3 to 12. For each row, the left figure shows ratio of running times; the right figure shows logscale plot of CPU times for the AGG-based implementation. The dashed horizontal line indicates the one day cutoff time.

We first considered instances of Coffee Shop games with 4 rows, 4 columns and varying numbers of players. For each game size we generated 10 instances with random payoffs. Figure 3.14 (left) gives a boxplot of the ratio of running times between the two implementations. The AGG-based implementation was about 3 times faster for the 3-player instances and about 30 times faster for the 4-player instances. We also tested on Coffee Shop games with 3 players, 3 columns and numbers of rows varying from 4 to 7, again generating 10 instances with random payoffs at each size. Figure 3.14 (right) gives a boxplot of the ratio of running times. As expected, the AGG-based implementation was faster and the gap in performance widened as games grew.

We then investigated symmetric AGG-$\emptyset$s on randomly generated small world

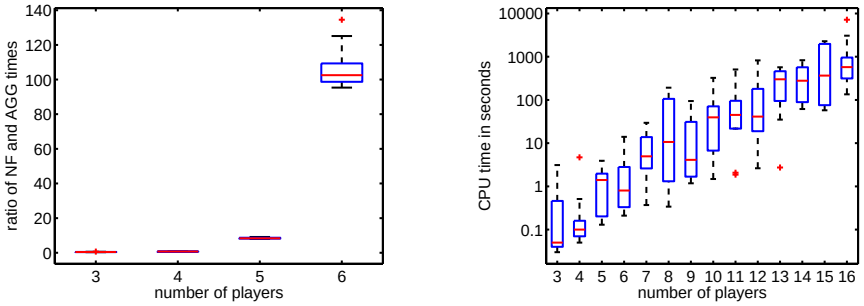

Figure 3.13: Govindan-Wilson algorithm; Job Market games, varying numbers of players. Left: ratios of running times. Right: logscale plot of CPU times for the AGG-based implementation.

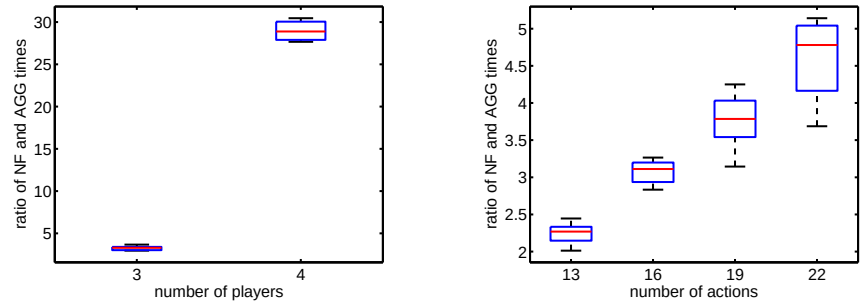

Figure 3.14: Ratios of running times of simplicial subdivision algorithms on Coffee Shop games. Left: 4×4 grid with 3 to 4 players. Right: 3-player $r \times 3$ grid, r varying from 4 to 7.

graphs with random payoffs. The small world graphs were generated using GAMUT's implementation with parameters $K = 1$ and $p = 0.5$. For each game size we generated 10 instances. We first fixed the number of action nodes at 5 and varied the number of players. Results are shown in Figure 3.15 (top row). While there was large variance in the absolute running times across different instances, the ratios of running times between normal-form-based and AGG-based implementations showed a clear increasing trend as the number of players increased. The normal-form-based implementation ran out of memory for instances with more than 5 players. Meanwhile, we ran the AGG-based implementation on larger instances with a one-day cutoff time. As shown by the boxplot, the AGG-based implementation solved most

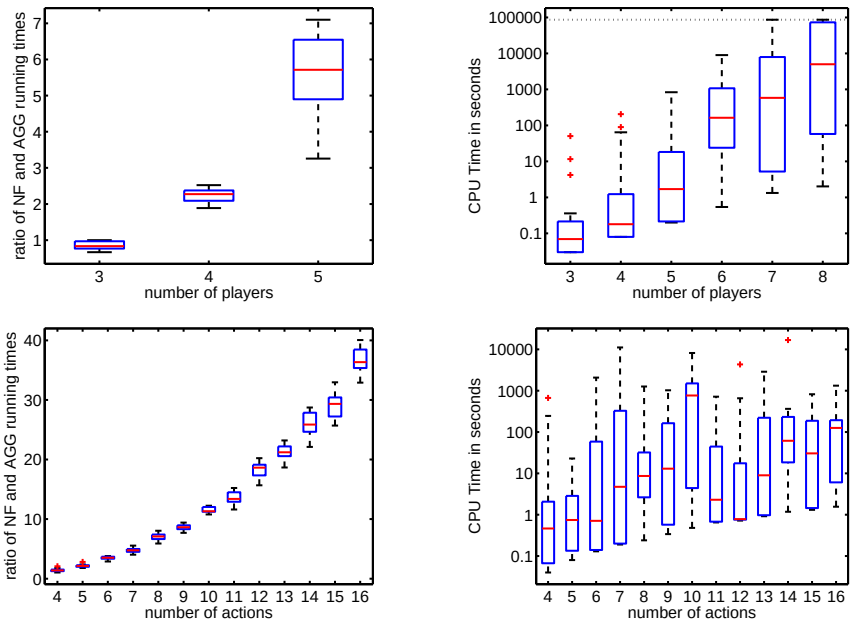

Figure 3.15: Simplicial subdivision algorithm; symmetric AGG-∅s on small world graphs. Top row: 5 actions, varying number of players. Bottom row: 4 players, varying number of actions. The left figures show ratios of running times; the right figures show logscale plots of CPU times for the AGG-based implementation. The dashed horizontal line indicates the one day cutoff time.

instances with up to 8 players within 24 hours. We then fixed the number of players at 4 and varied the number of action nodes from 4 to 16. Results are shown in Figure 3.15 (bottom row). Again, while the actual running times on different instances varied substantially, the ratios of running times showed a clear increasing trend as the number of actions increased. The AGG-based implementation was able to solve a 16-action instance in an average of about 3 minutes, while the normal-form-based implementation averaged about 2 hours.

3.6.7 Visualizing Equilibria on the Action Graph

Besides facilitating representation and computation, the action graph can also be used to visualize strategy profiles in a natural way. A strategy profile σ (e.g., a Nash

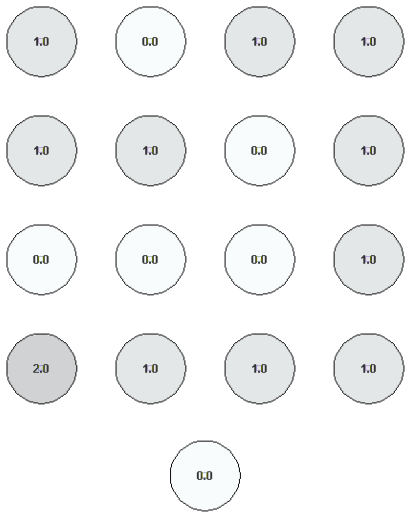

Figure 3.16: Visualization of a Nash equilibrium of a 16-player Coffee Shop game on a 4×4 grid. The function nodes and the edges of the action graph are not shown. The action node at the bottom corresponds to not entering the market.

equilibrium) can be visualized on the action graph by displaying the expected numbers of players that choose each of the actions. We call such a tuple the *expected configuration* under σ. This can be easily computed given σ: for each action node α, we sum the probabilities of playing α, i.e. $E[c(\alpha)] = \sum_{i \in N} \sigma_i(\alpha)$ where $\sigma_i(\alpha)$ is 0 when $\alpha \notin A_i$. When the strategy profile consists of pure strategies, the result is simply the corresponding configuration.

The expected configuration often has natural interpretations. For example in Coffee Shop games and other scenarios where actions correspond to location choices, an expected configuration can be seen as a density map describing expected player locations. We illustrate using a 16-player Coffee Shop game on a 4×4 grid. We ran the (AGG-based) Govindan-Wilson algorithm, finding a Nash equilibrium in 77 seconds. The expected configuration of this (pure strategy) equilibrium is visualized in Figure 3.16.

We also examined a Job Market game with 20 players. A normal form representation of this game would have needed to store 9.4×10^{134} numbers. We ran the AGG-based Govindan-Wilson algorithm, finding a Nash equilibrium in 860 sec-

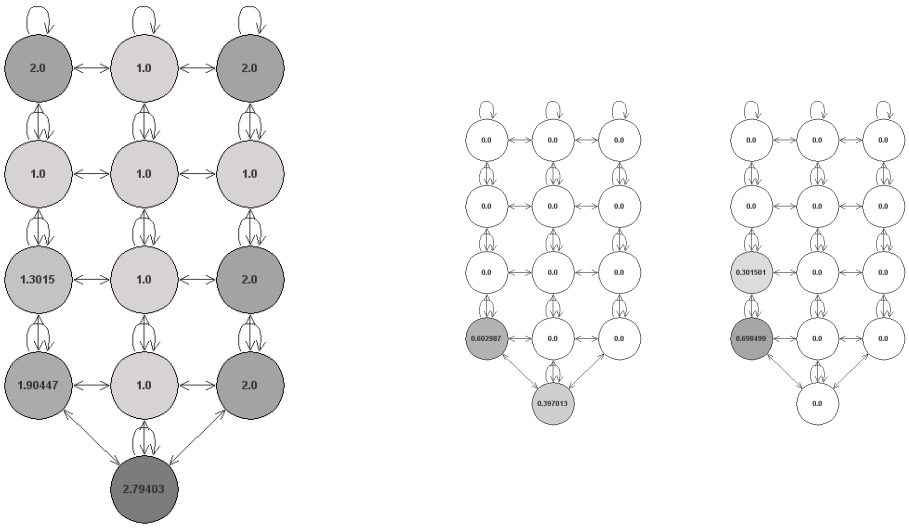

Figure 3.17: Visualization of a Nash equilibrium of a Job Market game with 20 players. Left: expected configuration of the equilibrium. Right: two mixed equilibrium strategies.

onds. The expected configuration of this equilibrium is visualized in Figure 3.17 (left). Note that the equilibrium expected configuration on some of the nodes are non-integer values, as a result of mixed strategies by some of the players. We also visualize two players' mixed equilibrium strategies in Figure 3.17 (right).

Finally, we examined an Ice Cream Vendor game (Example 3.2.5) with 4 locations, 6 ice cream vendors, 6 strawberry vendors, and 4 west-side vendors. The Govindan-Wilson algorithm found an equilibrium in 9 seconds. The expected configuration of this (pure strategy) equilibrium is visualized in Figure 3.18. Observe that the west side is relatively denser due to the west-side vendors. The locations at the east and west ends were chosen relatively more often than the middle locations, because the ends have fewer neighbors and thus experience less competition.

3.7 Conclusions

We proposed action-graph games (AGGs), a fully expressive game representation that can compactly express utility functions with structure such as context-specific independence and anonymity. We also extended the basic AGG representation by

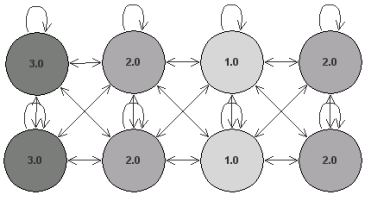

Figure 3.18: Visualization of a Nash equilibrium of an Ice Cream Vendor game.

introducing function nodes and additive utility functions, allowing us to compactly represent a wider range of structured utility functions. We showed that AGGs can efficiently represent games from many previously studied compact classes including graphical games, symmetric games, anonymous games, and congestion games. We presented a polynomial-time algorithm for computing expected utilities in AGG-$\emptyset$s and contribution-independent AGG-FNs. For symmetric and k-symmetric AGG-$\emptyset$s, we gave more efficient, specialized algorithms for computing expected utilities under symmetric and k-symmetric strategy profiles respectively. We also showed how to use these algorithms to achieve exponential speedups of existing methods for computing a sample Nash equilibrium and a sample correlated equilibrium. We showed experimentally that using AGGs allows us to model and analyze dramatically larger games than can be addressed with the normal-form representation.

In several later chapters of this thesis we present our efforts to extend and generalize our AGG framework. In Chapter 4 we consider the problem of computing PSNE. In Chapter 6 we propose Bayesian action-graph games (BAGGs) for representing Bayesian games, and in Chapter 5 we propose temporal action-graph games (TAGGs) for representing imperfect-information dynamic games.

Chapter 4

Computing Pure-strategy Nash Equilibria in Action-Graph Games

4.1 Introduction

In this chapter, we analyze the problem of computing pure-strategy Nash equilibria (PSNE) in AGGs. Recall from Section 2.2.6 that PSNEs do not always exist in a game. We focus on the problems of deciding if a PSNE exists, and of finding a PSNE, and later extend our analysis to the problem of computing a PSNE with optimal social welfare. The existence problem for AGGs is known to be NP-complete, even for symmetric AGG-$\emptyset$s with bounded in-degrees. Our goal in this chapter is to identify classes of AGGs for which this problem is tractable. We propose a dynamic programming approach and show that if the AGG-$\emptyset$ is symmetric and the action graph has bounded treewidth, our algorithm determines the existence of pure equilibria in polynomial time. We then extend our approach beyond symmetric AGG-$\emptyset$s.[1]

[1]This chapter is based on joint work with Kevin Leyton-Brown. Our earlier publication [Jiang and Leyton-Brown, 2007a] was restricted to the case of symmetric AGG-$\emptyset$s, and furthermore the proposed algorithm contained an error. In the current chapter we describe the corrected algorithm for symmetric AGGs, and furthermore extend the algorithm to certain classes of asymmetric AGGs.

We give a brief overview of our approach, and contrast it with some of the related literature mentioned in Section 2.2.6. Recall from Definition 2.2.5 that a PSNE is a pure-strategy profile satisfying certain incentive constraints. For symmetric AGGs, we can cast the problem in terms of configurations and constraints on configurations. With the graphical structure of AGGs, a natural idea is to construct global solutions (i.e., configurations corresponding to PSNE) from *partial solutions*, which are configurations over a subset of action nodes satisfying certain local constraints on the corresponding subgraph of the action graph. One difficulty when combining partial solutions from subgraphs is that of *inconsistency*. For the PSNE problem on graphical games, Gottlob et al. [2005] and Daskalakis and Papadimitriou [2006] showed that an effective technique for dealing with inconsistency is tree decomposition (and the related concept of hypertree decomposition). Roughly, a tree decomposition [Robertson and Seymour, 1986] of a graph consists of a family of overlapping subsets of vertices of the graph, and a tree structure with these subsets as nodes, satisfying certain properties such that algorithms for trees can be adapted to work on the tree decomposition, with running time exponential only in the tree decomposition's width (which measures the size of the largest subset). The *treewidth* of a graph is defined to be the width of the best tree decomposition for that graph. As a result, many NP-hard problems on graphs can be solved in polynomial time for graphs with bounded treewidth (see e.g., the recent survey by Bodlaender [2007]). For graphical games on bounded-treewidth graphs, it is sufficient to combine partial solutions from the leaves to the root of the tree decomposition while maintaining consistency across adjacent subsets, resulting in a polynomial-time algorithm for PSNE [Daskalakis and Papadimitriou, 2006]. However, whereas in graphical games the incentive constraints can be defined locally at each neighborhood, for AGGs we face an additional difficulty, because an agent could profitably deviate from playing an action in one part of the action graph to another. That is, the incentive constraints for PSNE in an AGG cannot be entirely captured by local constraints on subgraphs of the action graph. A simplified version of this difficulty was successfully dealt with in Ieong et al. [2005]'s polynomial-time algorithm for finding PSNE in *singleton congestion games*, which correspond to symmetric AGGs with only self edges. Their dynamic-programming algorithm is able to check against such deviations without having to store the exponential-sized

set of partial solutions, by maintaining *sufficient statistics* (specifically, bounds on utilities) that summarize the partial solutions compactly. Recall from *Chapter*3 that AGGs unify these existing representations; it turns out that our algorithm for AGGs also generalizes the existing algorithms for graphical games and singleton congestion games. Specifically, we define *restricted games* as AGGs played on subgraphs, equilibria of which satisfy the local incentive constraints; we then use tree-decomposition techniques to divide the action graph into subgraphs, allowing us construct equilibria of the game from equilibria of restricted games while maintaining consistency; and we use sufficient statistics (corresponding to the concept of characteristics [e.g., Bodlaender, 2007]) to check against deviations across partial solutions. Compared to the case of singleton congestion games, the edges (i.e., utility dependence) between action nodes in AGGs complicates the design of the sufficient static. Nevertheless we are able to overcome this technical challenge by further exploiting properties of tree decompositions.

4.2 Preliminaries

4.2.1 AGGs

We refer readers to Chapter 3 for definitions of AGG-$\emptyset$s, symmetric AGG-$\emptyset$s and k-symmetric AGG-$\emptyset$s. Recall that $\mathscr{I}$ is the maximum in-degree of the action graph. For an AGG-$\emptyset$ $\Gamma = (N, A, G, u)$, let $||\Gamma||$ denote the number of utility values the representation stores. Recall from Proposition 3.2.6 that this number is less or equal to $|\mathscr{A}|\frac{(n-1+\mathscr{I})!}{(n-1)!\mathscr{I}!}$, with equality holding when the AGG-$\emptyset$ is symmetric. Let $\mathscr{U}$ be the set of distinct utilities of the game Γ.

Whereas in Chapter 3 we only need to consider configurations restricted to the neighborhood of some action node, in this chapter we will need to talk about configurations over arbitrary sets of action nodes. For a configuration c and a set of actions $X \subset \mathscr{A}$, let $c[X]$ denote the restriction of c over X, i.e. $c[X] = (c[\alpha])_{\alpha \in X}$, where $c[\alpha]$ is the number of players choosing action α. Let $\mathscr{C}[X]$ denote the set of restricted configurations over X. Given an action graph $G = (\mathscr{A}, E)$ and a set of actions $X \subset \mathscr{A}$, let G_X be the action graph restricted to the action nodes X. Formally, $G_X \equiv (X, \{(\alpha, \alpha') \in E | \alpha, \alpha' \in X\})$. For a set of actions $X \subset \mathscr{A}$,

define $\nu(X) \equiv \{\alpha \in \mathscr{A} \setminus X | \exists x \in X$ such that $(\alpha, x) \in E\}$: the set of actions not in X that are neighbors of some action in X. Also define $\overline{X} \equiv \mathscr{A} \setminus X$ to be the complement of X. Then $\nu(\overline{X}) \equiv \{x \in X | \exists \alpha \in \mathscr{A} \setminus X$ such that $(x, \alpha) \in E\}$, the set of actions in X that are neighbors of some action not in X. Define $\tau(X) \equiv \{x \in X | \exists \alpha \in \mathscr{A} \setminus X$ such that $(x, \alpha) \in E$ or $(\alpha, x) \in E\}$. Given a configuration $c[X]$, let $\#c[X] \equiv \sum_{x \in X} c[x]$.

4.2.2 Complexity of Computing PSNE

Consider the problem determining whether a PSNE exists in a given AGG-$\emptyset$. Recall from Section 2.2.6 that the obvious algorithm of checking every possible action profile runs in linear time in the normal form representation of the game. However, since AGGs can be exponentially more compact than the normal form, the running time of this algorithm is worst-case exponential in the size of the AGG. Indeed, the PSNE problem becomes NP-complete when the input is an AGG-$\emptyset$.

Proposition 4.2.1. *The problem of determining whether a pure Nash equilibrium exists in an AGG-$\emptyset$ is NP-complete.*

Proof Sketch. It is straightforward to see that the problem is in NP, because given a pure strategy profile it takes polynomial time to verify whether that profile is a Nash equilibrium. NP-hardness follows from the fact that any graphical game can be transformed (in polynomial time) to an equivalent AGG-$\emptyset$ having the same space complexity, and the fact that the problem of determining the existence of pure equilibrium in graphical games is NP-hard [Daskalakis and Papadimitriou, 2006, Gottlob et al., 2005]. □

Perhaps more interestingly, the problem remains hard even if we restrict the games to be symmetric, in which case we cannot leverage existing results about graphical games. The following theorem was proved independently by Vincent Conitzer (personal communication) and Daskalakis et al. [2009].

Theorem 4.2.2 (Conitzer [pers. comm., 2004], Daskalakis et al. [2009])**.** *The problem of determining whether a pure Nash equilibrium exists in a symmetric AGG is NP-complete, even when the in-degree of the action graph is at most 3.*

4.3 Computing PSNE in AGGs with Bounded Number of Action Nodes

Now we look at classes of AGGs in which $|\mathscr{A}|$, the number of action nodes, is bounded by some constant. We show that in this case, the problem of finding pure equilibria can be solved in polynomial time. While this is a very restricted class of AGGs, we will use these results as building blocks for our dynamic programming approach for solving more complex AGGs.

We first look at symmetric AGGs. We restate the following well-known property of symmetric games [e.g., Brandt et al., 2009] in the language of AGGs:

Lemma 4.3.1. *Suppose Γ is a symmetric AGG. If a and α' induce the same configuration, then a is a PSNE of Γ iff α' is a PSNE of Γ.*

This is because the configuration determines the utilities, and since in a symmetric AGG any player can choose any action in $\mathscr{A}$, the configuration determines whether the incentive constraints for PSNE are satisfied. Note that this argument requires the symmetry property; in particular, the lemma no longer holds for asymmetric AGGs.

Lemma 4.3.1 allows us to consider only the configurations instead of all the pure strategy profiles. We say a configuration c is a PSNE of Γ if its corresponding pure strategy profiles are PSNE. The following straightforward lemma (a specialization of known facts about symmetric games [e.g., Brandt et al., 2009]) gives the incentive constraints for PSNE in terms of configurations.

Lemma 4.3.2. *A configuration c^* is a PSNE of a symmetric game iff for all $\alpha, \alpha' \in \mathscr{A}$, if $c^*[\alpha] > 0$,*

$$u^{\alpha}(c^*) \geq u^{\alpha'}(c^*_{\alpha \to \alpha'}) \tag{4.3.1}$$

where $c^*_{\alpha \to \alpha'}$ is the resulting configuration when one agent playing α in c^* deviates to α'. Formally, for all $x \in \mathscr{A}$,

$$c^*_{\alpha \to \alpha'}[x] = \begin{cases} c^*[x] - 1 & \text{if } x = \alpha \\ c^*[x] + 1 & \text{if } x = \alpha' \\ c^*[x] & \text{otherwise} \end{cases}$$

Given a configuration c, we can check whether it is a pure equilibrium in polynomial time.

Theorem 4.3.3. *The problem of determining whether a pure Nash equilibrium exists in a symmetric AGG with bounded $|\mathscr{A}|$ is in P.*

Proof. A polynomial algorithm is to check all configurations. Since $|\mathscr{A}|$ is bounded, the number of configurations $\binom{n+|\mathscr{A}|-1}{|\mathscr{A}|-1} = O(n^{|\mathscr{A}|-1})$ is polynomial. □

This can easily be extended to k-symmetric AGGs.

Definition 4.3.4. *Suppose Γ is a k-symmetric AGG in which the players are partitioned into equivalence classes $\{N_1,\ldots,N_k\}$ with the corresponding distinct action sets $\{\mathscr{A}^1,\ldots,\mathscr{A}^k\}$. Then given a pure strategy profile a, its corresponding* k-configuration *is a tuple $(c_\ell)_{1\le\ell\le k}$ where c_ℓ is the configuration over $\mathscr{A}^\ell$ induced by the players in N_ℓ. In other words, for all $\alpha \in \mathscr{A}^\ell$, $c_\ell[\alpha] = |\{i \in N_\ell | a_i = \alpha\}|$.*

Just as configurations capture all relevant information about pure strategy profiles in symmetric games, k-configurations capture all relevant information about pure strategy profiles in k-symmetric games. Thus we can determine the existence of pure equilibrium by checking all k-configurations. When k is bounded by a constant, there are polynomial number of k-configurations.

Lemma 4.3.5. *The problem of determining whether a pure Nash equilibrium exists in a k-symmetric AGG with bounded $|\mathscr{A}|$ and bounded k is in P.*

Proof. A polynomial algorithm is to check all k-configurations. Since $|\mathscr{A}|$ is bounded, for each $l \in \{1,\ldots,k\}$ the number of distinct c_l is $\binom{|N_l|+|\mathscr{A}^l|-1}{|\mathscr{A}^l|-1} = O(|N_l|^{|\mathscr{A}^l|-1})$. Therefore the number of distinct k-configurations is $O(n^{k(|\mathscr{A}|-1)})$, which is polynomial when k is bounded. For each k-configuration, checking whether it forms a Nash equilibrium takes polynomial time. Therefore the algorithm runs in polynomial time. □

Now consider the full class of AGGs with bounded $|\mathscr{A}|$. Interestingly, our problem remains easy to solve.

Theorem 4.3.6. *The problem of determining whether a pure Nash equilibrium exists in an arbitrary AGG with bounded $|\mathscr{A}|$ is in P.*

Proof. Any AGG Γ is k-symmetric by definition, where k is the number of distinct action sets. Since $A_i \subseteq \mathscr{A}$ for all i, the number of distinct nonempty action sets is at most $2^{|\mathscr{A}|} - 2$. This is bounded, since $|\mathscr{A}|$ is bounded by a constant. Thus Γ is k-symmetric with bounded k, and Lemma 4.3.5 applies. □

4.4 Computing PSNE in Symmetric AGGs

We now consider classes of AGGs in which $|\mathscr{A}|$ is not bounded. We first focus on symmetric AGG-$\emptyset$s. Since in this case all players have the same action set $\mathscr{A}$, we can identify a symmetric AGG-$\emptyset$ by the tuple $\langle n, G = (\mathscr{A}, E), u\rangle$. Whereas enumerating the configurations works well for AGGs with bounded $|\mathscr{A}|$, this approach is less effective in the general case with unbounded $|\mathscr{A}|$: in a symmetric AGG-$\emptyset$, the number of configurations over $\mathscr{A}$ is $\binom{n+|\mathscr{A}|-1}{|\mathscr{A}|-1}$, which is superpolynomial in $||\Gamma||$ when $\mathscr{I}$ is bounded.

Our approach is to use dynamic programming to construct PSNE of the game from PSNE of games restricted to parts of the action graph. This approach belongs to a large family of tree-decomposition-based dynamic programming algorithms for problems on graphs. In particular, in this section we adapt the standard concepts of *partial solutions* and *characteristics* [e.g., Bodlaender, 1997] to the PSNE problem in AGGs.

4.4.1 Restricted Games and Partial Solutions

We first introduce the concept of a *restricted game* on $R \subset \mathscr{A}$, which intuitively is the game played by a subset of players when we "restrict" them to the subgraph G_R, i.e., require them to choose their actions from R. Of course, the utility functions of this restricted game are not defined until we specify a configuration on $\nu(R)$.

Definition 4.4.1. *Given a symmetric AGG-$\emptyset$ Γ, a set of actions $R \subset \mathscr{A}$, a configuration $c[\nu(R)]$ and $n' \leq n$, we define the* restricted game $\Gamma(n', R, c[\nu(R)])$ *to be a symmetric AGG with n' players and with G_R as the action graph. Each action $\alpha \in R$ has the utility function $u^\alpha|_{c[\nu(R)]}$, which is the same as u^α as defined in Γ except that the configuration of nodes outside R is assigned by $c[\nu(R)]$. Formally,* $\Gamma(n', R, c[\nu(R)]) = \left\langle n', G_R, \left(u^\alpha|_{c[\nu(R)]}\right)_{\alpha \in R}\right\rangle$.

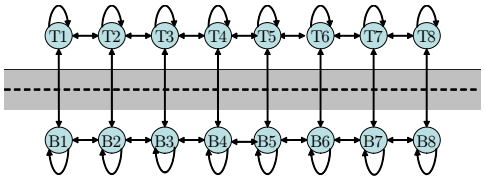

Figure 4.1: The road game with $m = 8$ and the action graph of its AGG representation.

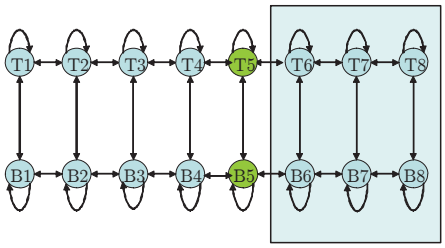

Figure 4.2: Restricted game on the rightmost 6 actions.

Example 4.4.2. *Suppose each of n agents is interested in opening a business, and can choose to locate in any block along either side of a road of length m. Multiple agents can choose the same block. Agent i's payoff depends on the number of agents who chose the same block as he did, as well as the numbers of agents who chose each of the adjacent blocks of land. This game can be compactly represented as a symmetric AGG, whose action graph is illustrated in Figure 4.1. To specify a restricted game on the rightmost 6 action nodes* $R = \{T6, T7, T8, B6, B7, B8\}$ *of the road game of Figure 4.1, we need to specify the number of players on R as well as the configuration over* $\nu(R) = \{T5, B5\}$*. This is illustrated in Figure 4.2, with R enclosed by the shaded rectangle and* $\nu(R)$ *in green.*

Lemma 4.3.1 tells us that we only need to consider configurations instead of strategy profiles. Likewise, for a restricted game on the subgraph $X \subset \mathscr{A}$, we only need to consider restricted configurations $c[X]$. The following lemma is straightforward.

Lemma 4.4.3. *If* c^* *is a pure equilibrium of* Γ*, then* $c^*[X]$ *is a pure equilibrium of the restricted game* $\Gamma(\#c^*[X], X, c^*[\nu(X)])$.

We want to use equilibria of restricted games as building blocks to construct equilibria of the entire game. Of course, a restricted game on $X \subset \mathscr{A}$ is not well-defined until we specify $c[\nu(X)]$. Thus we define a *partial solution* as a configuration on $X \cup \nu(X)$ which describes a restricted game on X as well as a pure equilibrium of it.

Definition 4.4.4. *A partial solution on* $X \subseteq \mathscr{A}$ *is a configuration* $c[X \cup \nu(X)]$ *such that* $c[X]$ *is a pure equilibrium of the restricted game* $\Gamma(\#c[X], X, c[\nu(X)])$.

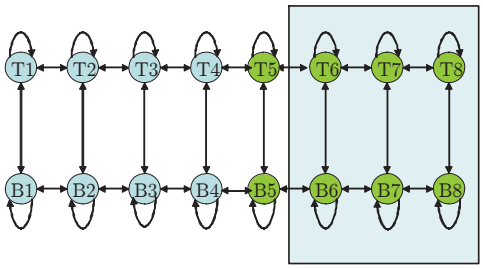

Figure 4.3: A partial solution on the rightmost 6 actions describes the configuration over these 8 actions.

For the restricted game in Figure 4.2, the corresponding partial solution on $R = \{\text{T6}, \text{T7}, \text{T8}, \text{B6}, \text{B7}, \text{B8}\}$ is a configuration over $R \cup \nu(R)$, illustrated in Figure 4.3 as green nodes.

We say a partial solution $c[X \cup \nu(X)]$ can be *extended* if there exists a configuration c^* such that c^* is a PSNE of Γ and $c^*[X \cup \nu(X)] = c[X \cup \nu(X)]$.

4.4.2 Combining Partial Solutions

In order to combine partial solutions to form a partial solution on a larger subgraph, we need to make sure that the result is a valid restricted strategy profile. We say two partial solutions $c'[X]$ and $c''[Y]$ are *consistent* if there exists a configuration c of the AGG-$\emptyset$ such that $c[X] = c'[X]$ and $c[Y] = c''[Y]$. The following lemma shows that it is simple to check whether $c[X]$ and $c'[Y]$ are consistent.

Lemma 4.4.5. *Given $X, Y \subseteq \mathscr{A}$, $c[X]$ is consistent with $c'[Y]$ iff*

1. *for all $\alpha \in X \cap Y$, $c[\alpha] = c'[\alpha]$, and*
2. *Let $n' = \#c[X] + \#c'[Y \setminus X]$, then $n' \leq n$. Furthermore, if $X \cup Y = \mathscr{A}$ then $n' = n$.*

We omit the straightforward proof. For two configurations $c[X], c'[Y]$ that are consistent with each other, we define $c[X] \cup c'[Y]$ to be the (unique) configuration on $X \cup Y$ that is consistent with both $c[X]$ and $c'[Y]$.

However, if we simply combine two consistent partial solutions that describe equilibria of restricted games on two disjoint sets $X, Y \in \mathscr{A}$, the result would not

necessarily induce an equilibrium of the restricted game on $X \cup Y$. This is because an agent who was playing an action in X might profitably deviate by playing an action in Y, and vice versa.

We could deal with this problem by keeping track of all pure equilibria of each restricted game, and determine case-by-case whether two equilibria can be combined (by checking whether agents could profitably deviate from one restricted game to the other). But as we combine the restricted games to form larger restricted games and eventually the unrestricted game on the entire action graph G, the number of equilibria we would have to store could grow exponentially.

4.4.3 Dynamic Programming via Characteristics

Perhaps we don't need to keep track of all partial solutions. Imagine we had a function ch that summarized them, i.e. it mapped each partial solution to a *characteristic* from a finite set $\mathscr{C}$ which is smaller than the set of partial solutions. For this characteristic function to be useful, it need to be *equilibrium-preserving*, defined as follows.

Definition 4.4.6. *For $X \subset \mathscr{A}$, a function* ch() *that maps partial solutions to their characteristics is* equilibrium-preserving *if for all pairs of partial solutions $c[X]$ and $c'[X]$, if $ch(c[X]) = ch(c'[X])$ then ($c[X]$ can be extended) $\Leftrightarrow$ ($c'[X]$ can be extended).*

Thus an equilibrium-preserving characteristic function ch() induces a partition of the set of partial solutions into equivalence classes. All partial solutions with the same characteristic behave the same way, so we only need to consider the set of all distinct characteristics. For $X \subset \mathscr{A}$, we define $\mathscr{C}_X \subset \mathscr{C}$ to be the set of characteristics of partial solutions on X. Formally, $\mathscr{C}_X = \{\text{ch}(c[X \cup \nu(X)]) \quad | \quad c[X \cup \nu(X)]$ is a partial solution on $X\}$.

Given such a function ch, a dynamic-programming algorithm for determining the existence of PSNE of Γ has the following high-level structure:

1. Construct $\mathscr{X} = \{X_1, \ldots, X_m\}$ such that $\bigcup_{1 \le j \le m} X_j = \mathscr{A}$.
2. For each $X_i \in \mathscr{X}$, compute $\mathscr{C}_{X_i}$, the set of characteristics of partial solutions on X_i.
3. While $|\mathscr{X}| \ge 2$:

(a) Take $X, Y \in \mathscr{X}$. Remove them from $\mathscr{X}$.

(b) Compute $\mathscr{C}_{X \cup Y}$ from $\mathscr{C}_X$ and $\mathscr{C}_Y$.

(c) Add $X \cup Y$ to $\mathscr{X}$.

4. Now $\mathscr{X}$ has only one member, $\mathscr{A}$. Return TRUE iff $\mathscr{C}_{\mathscr{A}}$ is not empty.

Since a partial solution on $\mathscr{A}$ is by definition a pure equilibrium of Γ, there exists a pure equilibrium of Γ if and only if $\mathscr{C}_{\mathscr{A}}$ is not empty. For this algorithm to run in polynomial time, the function ch() must satisfy the following properties:

Property 1: At all times during the algorithm, for all $X \in \mathscr{X}$, the size of $\mathscr{C}_X$ is polynomial. This is necessary since all restricted strategy profiles could potentially be partial solutions, and so $\mathscr{C}_X$ could potentially be the set of all possible characteristics for X.

Property 2: For each of the initial X_j, $\mathscr{C}_{X_j}$ can be computed in polynomial time.

Property 3: $\mathscr{C}_{X \cup Y}$ can be computed from $\mathscr{C}_X$ and $\mathscr{C}_Y$ in polynomial time.

One algorithm having the above structure is Ieong et al. [2005]'s algorithm for computing PSNE in singleton congestion games (corresponding to symmetric AGG-0s with only self-edges). Given such an AGG-0, the algorithm starts by partitioning $\mathscr{A}$ into sets each containing one action, and combines them in an arbitrary order. Consider two restricted games Γ' and Γ'' on two disjoint sets of action nodes X and Y respectively. Observe that in this case, to check consistency between two equilibria of Γ' and Γ'' respectively, it is sufficient to check the numbers of players in Γ' and Γ''. Given a restricted game Γ' on $X \subset \mathscr{A}$ and an equilibrium c^* of Γ', define the *worst current utility* $\mathrm{WCU}(c^*, \Gamma')$ to be the utility of the worst-off player in Γ', or ∞ if Γ' has 0 players. Define the *best entrance utility* $\mathrm{BEU}(c^*, \Gamma')$ to be the best payoff a player currently playing an action outside of X can get by playing an action in X, assuming the current players in Γ' play c^*. If Γ' already has all n players, $\mathrm{BEU}(c^*, \Gamma') = -\infty$. Since all players in a symmetric game are identical, if any player can profitably deviate out of Γ', then the worst-off player (with utility $\mathrm{WCU}(c^*, \Gamma')$) can profitably deviate out of Γ'; similarly if an agent can profitably deviate to any action in Γ', then she can achieve utility $\mathrm{BEU}(c^*, \Gamma')$. Therefore, to check whether agents could profitably deviate from Γ' currently in

equilibrium c' to Γ'' in equilibrium c'', we just need to check whether $\text{WCU}(c',\Gamma')$ is greater than $\text{BEU}(c'',\Gamma')$. Thus $\text{WCU}(c',\Gamma')$ and $\text{BEU}(c',\Gamma')$ can be used as sufficient statistics for checking existence of profitable deviations out of and into the restricted game Γ', and $\#c[X]$ for checking consistency. The resulting characteristics are equilibrium-preserving, and require less space than keeping track of the partial solutions on X because WCU and BEU are utility values and thus there are at most $||\Gamma||^2$ possible pairs.

We adapt Ieong et al. [2005]'s characteristic function to general symmetric AGGs. First of all, we now need $c[\nu(X)]$ in order to specify restricted games and partial solutions on X. As a result, to check consistency between a partial solution on X and partial solutions on other parts of the graph, we need to keep track of the number of players in X, the configuration over $\nu(X)$, and the configuration over $\nu(\overline{X})$.

Furthermore, in general action graphs, we may have sets $X,Y \subset \mathscr{A}$ such that $\nu(X) \cap Y \neq \emptyset$. In such cases deviating from an action in $\nu(X) \cap Y$ to a restricted game Γ' on X changes the configuration on $\nu(X)$, which in turn affects the utility functions of Γ'. In other words, the best utility a player originally playing an action $\alpha \in \overline{X}$ can get by deviating into Γ' on X with current configuration c^* is a quantity that depends on (1) whether α is in $\nu(X)$ and (2) if so, α itself. As a result, simply using $\text{BEU}(c^*,\Gamma')$ and $\text{WCU}(c^*,\Gamma')$ is no longer sufficient for checking profitable deviations.

We thus need more sophisticated sufficient statistics for checking deviations in this case. One approach is to extend our definition of $\text{BEU}(c^*,\Gamma')$ by making it vector-valued, specifying the best utilities when the deviating player is an outside player and when the player is playing each of the actions in $\nu(X)$. The length of the resulting vector is thus $|\nu(X)|+1$. Furthermore we could extend $\text{WCU}(c^*,\Gamma')$ by making it a vector consisting of the worst utility from $X \setminus \nu(\overline{X})$ and from each of the actions in $\nu(\overline{X})$. Although it is intuitive, it turns out that this approach yields a polynomial-time algorithm only in the case of symmetric AGG-$\emptyset$s with bounded treewidth and bounded in-degree.

Instead, in this chapter we describe a different approach that yields a polynomial-time algorithm for bounded-treewidth symmetric AGG-$\emptyset$s, thus eliminating the separate requirement on in-degree. First, we redefine $\text{BEU}(c^*,\Gamma')$ in terms of devia-

tions from players outside of $X \cup \nu(X)$.

Definition 4.4.7. *Given a restricted game Γ' on $X \subset \mathscr{A}$ and an equilibrium c^* of Γ', the* best entrance utility $BEU(c^*, \Gamma')$ *is the best payoff an* outside player *(a player currently playing an action outside of $X \cup \nu(X)$) can get by playing an action in X, assuming the current players in Γ' play c^*. If there are 0 outside players, $BEU(c^*, \Gamma') = -\infty$.*

In order to check deviations into and out of X, we partition X into P and $X \setminus P$, and check the corresponding restricted games separately. We will specify P in Section 4.4.4; for now we only require that $X \supseteq P \supseteq \tau(X)$. Recall that $\tau(X)$ are the set of nodes in X with outgoing edges to and/or incoming edges from nodes outside X. Intuitively, P contains all nodes in X that we cannot apply BEU and WCU to. This implies $\nu(X \setminus P) \cap \overline{X} = \emptyset$ and $\nu(\overline{X}) \cap (X \setminus P) = \emptyset$. Thus we can use WCU and BEU for restricted games on $X \setminus P$ as sufficient statistics for checking deviations between $X \setminus P$ and nodes outside X. The remaining task is to check deviations between P and nodes outside X. We do this by explicitly keeping track of configurations on $Q \supseteq P \cup \nu(P)$. We will exactly specify Q in Section 4.4.4. In other words, we keep track of the partial solutions on P. Note in particular that this provides enough information to specify the corresponding restricted games on P. Finally, since configurations over $X \setminus P$ will not be referred to by partial solutions on any $Y \subset \mathscr{A}$ that is disjoint from X, in order to maintain consistency it is sufficient to keep track of the number of players playing in X and the configuration over $P \cup \nu(X)$, which is a subset of Q.

Taking these together, we have the following characteristic function.

Lemma 4.4.8. *Given $X \subset \mathscr{A}$, $P \subseteq X$ such that $P \supseteq \tau(X)$, and $Q \supseteq P \cup \nu(P)$, consider the characteristic function $ch_{P,Q}$ that maps a partial solution $c[X \cup \nu(X)]$ to*

$$ch_{P,Q}(c[X \cup \nu(X)]) = (c[Q], \#c[X], WCU(c[X'], \Gamma'), BEU(c[X'], \Gamma')),$$

where $\Gamma' = \Gamma(\#c[X'], X', c[\nu(X')])$ and $X' = X \setminus P$. Then $ch_{P,Q}$ is equilibrium-preserving.

Proof. Suppose we have two partial solutions $c[X \cup \nu(X)]$ and $c'[X \cup \nu(X)]$ such that $\mathrm{ch}_{P,Q}(c[X \cup \nu(X)]) = \mathrm{ch}_{P,Q}(c'[X \cup \nu(X)])$. Furthermore $c[X \cup \nu(X)]$ can be

extended, i.e., there exists a PSNE c^* of the game such that $c^*[X \cup \nu(X)] = c[X \cup \nu(X)]$. We need to show that $c'[X \cup \nu(X)]$ can be extended. Since $c^*[\overline{X} \cup \nu(\overline{X})]$ and $c[X \cup \nu(X)]$ are consistent, and since $c[X \cup \nu(X)]$ and $c'[X \cup \nu(X)]$ have the same characteristic (in particular, the same configuration on $\nu(\overline{X}) \cup \nu(X)$ and the same number of players in X), therefore $c^*[\overline{X} \cup \nu(\overline{X})]$ and $c'[X \cup \nu(X)]$ are consistent. Consider the configuration $c'^* \equiv c^*[\overline{X} \cup \nu(\overline{X})] \cup c'[X \cup \nu(X)]$. We claim that c'^* is a PSNE of the game (which directly implies that $c'[X \cup \nu(X)]$ can be extended). To show this, we observe that since $c^*[\overline{X} \cup \nu(\overline{X})]$ and $c'[X \cup \nu(X)]$ are already partial solutions on $\overline{X}$ and X respectively (and are consistent with each other), we only need to make sure there are no profitable deviations between them. We partition X into P and $X' = X \setminus P$. Since there were no profitable deviations between partial solutions $c[P \cup \nu(P)]$ and $c^*[\overline{X} \cup \nu(\overline{X})]$, and since $c[P \cup \nu(P)] = c'[P \cup \nu(P)]$, there are no profitable deviations between partial solutions $c'[P \cup \nu(P)]$ and $c^*[\overline{X} \cup \nu(\overline{X})]$. Suppose there is a profitable deviation from X' under partial solution $c'[X' \cup \nu(X')]$ to $\overline{X}$ under partial solution $c^*[\overline{X} \cup \nu(\overline{X})]$. Then there is a profitable deviation from the worst-off player in X' under $c'[X' \cup \nu(X')]$. Since her utility is equal to that of the worst-off player in X' under $c[X' \cup \nu(X')]$, there must be a profitable deviation from the partial solution $c[X' \cup \nu(X')]$ to $c^*[\overline{X} \cup \nu(\overline{X})]$, a contradiction. A similar argument shows that there is no profitable deviation from $\overline{X}$ under $c^*[\overline{X} \cup \nu(\overline{X})]$ to X' under $c'[X' \cup \nu(X')]$. □

We denote by $\mathscr{C}_X^{P,Q}$ the set of characteristics on X under the characteristic function $\mathrm{ch}_{P,Q}$. For the restricted game in Example 4.4.2, we can use $P = \{\mathrm{T6}, \mathrm{B6}\}$ and $Q = P \cup \nu(P) = \{\mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{B5}, \mathrm{B6}, \mathrm{B7}\}$. These are illustrated in Figure 4.4.

The following lemma shows how sets of characteristics from two subsets X' and X'' of $\mathscr{A}$ (with characteristic functions $\mathrm{ch}_{P',Q'}$ and $\mathrm{ch}_{P'',Q''}$ respectively) can be combined together. Here we require that X' and X'' have a limited amount of overlap; specifically, we require that $X' \cap X'' \subseteq P' \cup P''$. Intuitively, the combination of subsets with such overlap is manageable because (1) we can calculate the total number of players in $X' \cup X''$ from the characteristics because we know the configuration of (and thus the number of players in) $X' \cap X''$; and (2) since the configuration of $X' \cap X''$ is already "in equilibrium" with both sides, it is sufficient to check deviations from $X'' \setminus X'$ to $X' \setminus X''$ and vice versa. We do this by partitioning

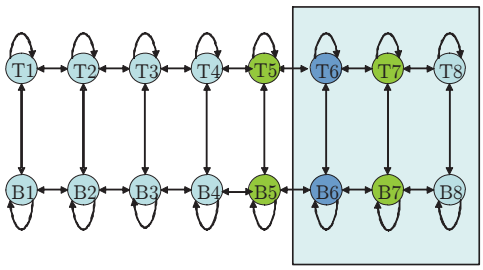

Figure 4.4: Characteristic function $\mathrm{ch}^{P,Q}$ for the rightmost 6 actions with $P = \{\mathrm{T6}, \mathrm{B6}\}$ and $Q = \{\mathrm{T5}, \mathrm{T6}, \mathrm{T7}, \mathrm{B5}, \mathrm{B6}, \mathrm{B7}\}$.

the former into $X'' \setminus P''$ and $P'' \setminus X'$, and the latter into $X' \setminus P'$ and $P' \setminus X''$, then checking the resulting set of deviations using information provided by the characteristics.

Lemma 4.4.9. *Suppose that $X, P, Q, X', P', Q', X'', P'', Q''$ are subsets of $\mathscr{A}$ such that $\tau(X) \subseteq P \subseteq X$, $\tau(X') \subseteq P' \subseteq X'$, $\tau(X'') \subseteq P'' \subseteq X''$, $Q \supseteq P \cup \nu(P)$, $Q' \supseteq P' \cup \nu(P')$, $Q'' \supseteq P'' \cup \nu(P'')$, $X' \cap X'' \subseteq P' \cup P''$, and $X' \cup X'' = X$. For all $c[Q] \in C[Q]$, integer $B \leq n$, and $U_c, U_e \in \mathscr{U}$, the tuple $(c[Q], B, U_c, U_e) \in \mathscr{C}_X^{P,Q}$ if and only if there exist $c'[Q']$, $c''[Q'']$, B', B'', and U_c', U_c'', U_e', and U_e'' such that*

1. $(c'[Q'], B', U_c', U_e') \in \mathscr{C}_{X'}^{P',Q'}$,
2. $(c''[Q''], B'', U_c'', U_e'') \in \mathscr{C}_{X''}^{P'',Q''}$,
3. $c'[Q']$ *is consistent with* $c''[Q'']$,
4. $c[Q] = c'''[Q]$ *where* $c''' = c'[Q'] \cup c''[Q'']$,
5. $B = B' + B'' - c'''[X' \cap X'']$, *and if* $X = \mathscr{A}$ *then* $B = n$,
6. $U_c' \geq U_e''$ *and* $U_c'' \geq U_e'$,
7. $U_c' \geq BEU(c''[P'' \setminus X'], \Gamma'')$, $WCU(c'[P' \setminus X''], \Gamma') \geq U_e''$, $U_c'' \geq BEU(c'[P' \setminus X''], \Gamma')$, $WCU(c''[P'' \setminus X'], \Gamma'') \geq U_e'$ *where* $\Gamma' = \Gamma(\#c'[P' \setminus X''], P' \setminus X'', c'[\nu(P' \setminus X'')])$ *and* $\Gamma'' = \Gamma(\#c''[P'' \setminus X'], P'' \setminus X', c''[\nu(P'' \setminus X')])$,
8. $c[P' \cup P'']$ *is an equilibrium of* $\Gamma(\#c[P' \cup P''], P' \cup P'', c'''[\nu(P' \cup P'')]$,
9. $U_c = \min\{U_c', U_c'', WCU(c'''[Z], \Gamma_Z)\}$ *and* $U_e = \max\{U_e', U_e'', BEU(c'''[Z], \Gamma_Z)\}$, *where* $Z = (P' \cup P'') \setminus P$ *and* $\Gamma_Z = \Gamma(\#c'''[Z], Z, c'''[\nu(Z)])$.

Proof Sketch. $\Rightarrow$ ("only if") part: Suppose $c[X \cup \nu(X)]$ is a partial solution on X with characteristic $(c[Q], B, U_c, U_e)$. Then let $c'[X' \cup \nu(X')] = c[X' \cup \nu(X')]$. It is straightforward to see that $c'[X']$ is an equilibrium of the restricted game $\Gamma(\#c'[X'], X', c[\nu(X')])$. Therefore $c'[X' \cup \nu(X')]$ is a partial solution on X'. Similarly, let $c''[X'' \cup \nu(X'')] = c[X' \cup \nu(X'')]$, and the same argument applies. Then it is straightforward to verify that the characteristics of $c'[X' \cup \nu(X')]$ and $c''[X'' \cup \nu(X'')]$ satisfy the above conditions.

$\Leftarrow$ ("if") part: Suppose $c'[X' \cup \nu(X')]$ and $c''[X'' \cup \nu(X'')]$ are partial solutions with characteristics $(c'[Q'], B', U'_c, U'_e)$ and $(c''[Q''], B'', U''_c, U''_e)$ respectively, and there exists $c[Q], B, U_c, U_e$ such that conditions 3 to 9 are satisfied. Then conditions 3 and 5 together with Lemma 4.4.5 imply that $c'[X' \cup \nu(X')]$ and $c''[X'' \cup \nu(X'')]$ are consistent. Let $c = c'[X' \cup \nu(X')] \cup c''[X'' \cup \nu(X'')]$. By a similar argument as in the proof of Lemma 4.4.8, conditions 6 to 8 ensure that there are no profitable deviations between the partial solutions $c'[X' \cup \nu(X')]$ and $c''[X'' \cup \nu(X'')]$, and therefore $c[X]$ is an equilibrium of the restricted game $\Gamma(B, X, c[\nu(X)])$. Let $Y = X \setminus P$. Then $X' \setminus P'$, $X'' \setminus P''$ and Z partitions Y. By the definition of worst current utility, $\mathrm{WCU}(c[Y], \Gamma(\#c[Y], Y, c[\nu(Y)]))$ is the minimum of $\{U'_c, U''_c, \mathrm{WCU}(c'''[Z], \Gamma_Z)\}$, which are the worst current utilities on $X' \setminus P$, $X'' \setminus P''$ and Z respectively. Therefore $\mathrm{WCU}(c[Y], \Gamma(\#c[Y], Y, c[\nu(Y)])) = U_c$. Similarly $\mathrm{BEU}(c[X], \Gamma(B, X, c[\nu(X)])) = U_e$. Therefore $c[X \cup \nu(X)]$ is a partial solution with characteristic $(c[Q], B, U_c, U_e)$. $\square$

Lemma 4.4.9 implies that it takes polynomial time to check if two characteristics $(c'[Q'], B', U'_c, U'_e) \in \mathscr{C}_{X'}^{P',Q'}$ and $(c''[Q''], B'', U''_c, U''_e) \in \mathscr{C}_{X''}^{P'',Q''}$ are consistent and if there are no profitable deviations between them, and if so to construct a characteristic in $\mathscr{C}_X^{P,Q}$ for their combined partial solutions. Thus if we iterate over all pairs of characteristics in $\mathscr{C}_{X'}^{P',Q'}$ and $\mathscr{C}_{X''}^{P'',Q''}$ respectively, we can construct $\mathscr{C}_X^{P,Q}$ in time polynomial in the sizes of $\mathscr{C}_{X'}^{P',Q'}$ and $\mathscr{C}_{X''}^{P'',Q''}$.

Let us now consider the size of $\mathscr{C}_X^{P,Q}$ for an arbitrary $X \subseteq \mathscr{A}$. Recall that the WCU and BEU are utility values and thus each has at most $|\mathscr{U}| \leq ||\Gamma||$ distinct values. Also $\#c[X] \in \{0, \ldots, n\}$ by definition. So the number of distinct characteristics can be much smaller than the number of corresponding partial solutions $c[X \cup \nu(X)]$ when $|Q| \ll |X \cup \nu(X)|$. However, since $Q \supseteq \nu(X)$ and $|\nu(X)|$ is $|X|\mathscr{I}$

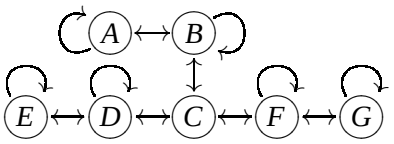

Figure 4.5: An action graph G.

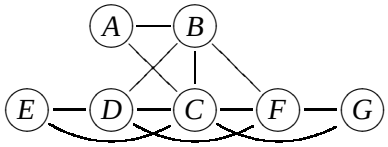

Figure 4.6: The primal graph G'.

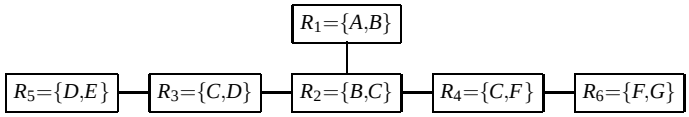

Figure 4.7: Tree decomposition of $und(G)$

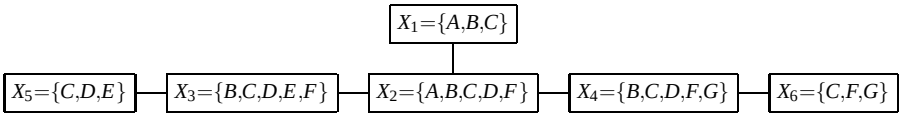

Figure 4.8: Tree decomposition of primal graph G', satisfying the conditions of Lemma 4.4.11.

in the worst case, the number of possible configurations over Q is superpolynomial in $||\Gamma||$ in the worst case. Since $\mathscr{C}_X^{P,Q}$ could potentially include every distinct tuple $(c[Q], B, U_c, U_e)$, the size of $\mathscr{C}_X^{P,Q}$ is superpolynomial in the worst case. Indeed, Theorem 4.2.2 showed that we will not find a poly-time algorithm for general symmetric AGGs unless P = NP. Nevertheless, we next show that if the action graph G has bounded treewidth, we can combine the restricted games in a way such that the number of configurations $|C[Q]|$ (and thus $|\mathscr{C}_X^{P,Q}|$) remains polynomial in $||\Gamma||$ as X grows.

4.4.4 Algorithm for Symmetric AGGs with Bounded Treewidth

We first introduce some notation. Given an action graph $G = (\mathscr{A}, E)$, define $\mathscr{H}(G)$ to be the hypergraph $(\mathscr{A}, \mathscr{E})$ with $\mathscr{E} = \{\{\alpha\} \cup \nu(\alpha) | \alpha \in \mathscr{A}\}$. In other words, for each action $\alpha \in \mathscr{A}$, there is a hyperedge containing α and its neighbors. Duplicate hyperedges are removed. Let G' be the *primal graph* of the hypergraph $\mathscr{H}(G)$. G' is a undirected graph on the same set of vertices, and there is an edge between two nodes if they are in some hyperedge in $\mathscr{H}(G)$. $G' = (\mathscr{A}, \{\{u,v\} | \exists h \in \mathscr{E} \text{ such that } u,v \in h\})$. Thus for each $\alpha \in \mathscr{A}$, α and its neighbors in G form a clique in G'. In the Bayes net literature G' is also known

as the *moral graph* of G. For example, Figure 4.5 shows the action graph G of a symmetric AGG. Its hypergraph $\mathscr{H}(G)$ has the same set of vertices and the hyperedges $\{A,B\}$, $\{A,B,C\}$, $\{D,E\}$, $\{C,D,E\}$, $\{F,G\}$, $\{C,F,G\}$, and $\{B,C,D,E\}$. Figure 4.6 shows G's primal graph G'.

The concept of tree decomposition and treewidth was introduced by Robertson and Seymour [1986].

Definition 4.4.10. *A* tree decomposition *of an undirected graph $G' = (V,E)$ is a pair $(\mathscr{X},T)$ with $T = (I,F)$ a tree (where I and F are the nodes and edges of the tree respectively), and $\mathscr{X} = \{X_i | i \in I\}$ a family of subsets of V, one for each node of T, such that*

1. $\bigcup_{i\in I} X_i = V$,
2. *for all edges $\{v,w\} \in E$ there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$, and*
3. *for all $i,j,k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of a tree decomposition is* $\max_{i\in I} |X_i| - 1$. *The* treewidth $tw(G')$ *of a graph G' is the minimum width over all tree decompositions of G'.*

Condition 3 of the definition can be equivalently stated as the following: for all $v \in V$, the set $\{i \in I | v \in X_i\}$ induces a subtree of T.

Let the treewidth $tw(\Gamma)$ of an AGG Γ be the treewidth of $und(G)$, the undirected version of its action graph G (excluding self-edges). Figure 4.7 shows a tree decomposition $(\{R_i | i \in I\}, T = (I,F))$ of the undirected version of the action graph G in Figure 4.5. In this case $und(G)$ is a tree. The width of the tree decomposition is 1 since each tree node contains at most 2 vertices of $und(G)$. This is a tree decomposition of minimum width, since any tree decomposition must have nodes containing e.g., both A and B since $\{A,B\}$ is an edge in $und(G)$. In fact, it is known in general that the treewidth of a connected tree is 1.

A tree decomposition of $und(G)$ provides a family of subsets ($R_1,\ldots,R_6$ in Figure 4.7) of vertices that cover $\mathscr{A}$, and if the width of the decomposition is bounded by a constant that implies the sizes of R_i are bounded. We will be using R_i as the P's in Lemmas 4.4.8 and 4.4.9. However, we also need to control the size of $Q \supseteq P \cup \nu(P)$ in those lemmas in order to control the running time of the

resulting dynamic programming algorithm. It turns out that a tree decomposition of the primal graph can be constructed that yields the appropriate Q's of Lemmas 4.4.8 and 4.4.9. Given a tree graph $T = (I, F)$ and $J \subset I$, let T_J be the subgraph of T restricted to J.

Lemma 4.4.11. *Given a symmetric AGG-$\emptyset$ Γ with treewidth w, there exists a tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ of the primal graph G' of width at most $(w+1)(\mathscr{I}+1)-1$, and $\{R_i | i \in I\}$ such that*

1. *$\bigcup_{i \in I} R_i = \mathscr{A}$, and $R_i \cup \nu(R_i) \subseteq X_i$ for all $i \in I$,*
2. *Let $J \subset I$ such that T_J is a connected graph and connects to the rest of the tree via only one edge $\{j, j'\} \in F$ with $j \in J$. Let $Y_J = \bigcup_{i \in J} R_i$. Then $\tau(Y_J) \subseteq R_j$.*

Proof. By assumption there exists a tree decomposition of $\mathit{und}(G)$ of width w. Denote this decomposition $(\{R_i | i \in I\}, T = (I, F))$. Then $\bigcup_{i \in I} R_i = \mathscr{A}$. Let $X_i = R_i \cup \nu(R_i)$ for all $i \in I$. Daskalakis and Papadimitriou [2006] proved that the resulting $(\{X_i | i \in I\}, T)$ is a tree decomposition of the primal graph G' having width at most $(w+1)(\mathscr{I}+1)-1$. Then $R_i \cup \nu(R_i) \subseteq X_i$.

Given J, j and Y_J as defined in the statement of the lemma, we claim that $\tau(Y_J) \subseteq R_j$. To see this, consider each $\alpha \in \tau(Y_J)$. Then by definition there must be an $\alpha' \in \overline{Y_J}$ such that $\{\alpha, \alpha'\}$ is an edge in $\mathit{und}(G)$. We note that $T_{I \setminus J}$ is also connected. Since $Y_J = \bigcup_{i \in J} R_i$, we have $\overline{Y_J} \subseteq \bigcup_{i \in I \setminus J} R_i = Y_{I \setminus J}$ and thus $\alpha' \in Y_{I \setminus J}$. Since $\{\alpha, \alpha'\}$ is an edge in $\mathit{und}(G)$, by condition 2 of Definition 4.4.10 there exists $i' \in I$ such that $\alpha, \alpha' \in R_{i'}$. Furthermore such i' must be in $I \setminus J$ since $\alpha' \notin Y_J$. Since α is contained in some R_i with $i \in J$, by condition 3 of Definition 4.4.10 α must be contained in all $R_{i''}$ such that i'' is on the path from i to i' in T. Since j is on this path, $\alpha \in R_j$. □

Since the undirected version of the action graph in Figure 4.5 has treewidth 1, Lemma 4.4.11 guarantees a tree decomposition of the primal graph with width at most 7 satisfying the above conditions. Figure 4.8 shows such a tree decomposition (with width 4) of the primal graph G' from Figure 4.6. Each node $i \in I$ of the tree is labeled with X_i.

Lemma 4.4.11 together with Lemma 4.4.8 imply that:

Corollary 4.4.12. *Given any J, j and Y_J satisfying condition 2 of Lemma 4.4.11, ch_{R_j,X_j} is an equilibrium-preserving characteristic function on Y_J.*

Also observe that for all $i \in I$, ch_{R_i,X_i} is trivially an equilibrium-preserving characteristic function on R_i.

Pick an arbitrary node $r \in I$ to be the root of T. We say node j is a descendant of node i (equivalently i is an ancestor of j) if i is on the path from r to j. Define $Z_i = \{v \in R_j | j = i \text{ or } j \text{ is a descendant of } i\}$. Then $Z_r \equiv \mathscr{A}$. Intuitively, when we combine the restricted games associated with node i and its descendants in T, we would get a restricted game on Z_i. For each node $i \in I$ with children $q_1, \ldots, q_m \in I$, for each $j \leq m$, define $Z_{i,j} = R_i \cup Z_{q_1} \cup \ldots \cup Z_{q_j}$. This implies that $Z_{i,m} \equiv Z_i$. Then Corollary 4.4.12 implies that for any $Z_{i,j}$, ch_{R_i,X_i} is an equilibrium-preserving characteristic function. We write $\mathscr{C}_{Z_{i,j}} \equiv \mathscr{C}^{R_i,X_i}_{Z_{i,j}}$. For our tree decomposition in Figure 4.8, if we let node 1 be the root r, then $Z_5 = R_5$, $Z_6 = R_6$, $Z_3 = R_3 \cup R_5 = \{C,D,E\}$, $Z_4 = R_4 \cup R_6 = \{C,F,G\}$, $Z_2 = R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 = \{B,C,D,E,F,G\}$, and $Z_1 = \mathscr{A}$. Since node 2 has two children $q_1 = 3$ and $q_2 = 4$, then $Z_{2,1} = R_2 \cup Z_3 = \{B,C,D,E\}$ and $Z_{2,2} = Z_{2,1} \cup Z_4 = Z_2 = \{B,C,D,E,F,G\}$.

We adapt our dynamic programming algorithm from the previous section so that $\{R_i | i \in I\}$ is the initial family of subsets that covers $\mathscr{A}$, and the order in which the subsets are combined is guided by the tree decomposition, from the leaves to the root.

1. For each R_i, compute $\mathscr{C}_{R_i}$. This can be done by enumerating all possible configurations $c[X_i]$ and keeping those that induce a pure equilibrium of the restricted game on R_i.
2. Initialize the set Done $\subseteq I$ to contain the leaves of the tree T.
3. While $\exists i \in I \setminus \text{Done}$ such that $\{i' \in I | i' \text{ is a child of } i\} \subseteq \text{Done}$:
   (a) Let $\mathscr{C}_{Z_{i,0}} := \mathscr{C}_{R_i}$
   (b) Let $q_1, \ldots, q_m$ be the children of i.
   (c) For $j = 1$ to m, compute $\mathscr{C}_{Z_{i,j}}$ from $\mathscr{C}_{Z_{i,j-1}}$ and $\mathscr{C}_{Z_{q_j}}$ by applying Lemma 4.4.9.
   (d) $\mathscr{C}_{Z_i} := \mathscr{C}_{Z_{i,m}}$

(e) Add i to Done.

4. Return TRUE iff $\mathscr{C}_{Z_r}$ is nonempty.

For the tree decomposition in Figure 4.8 with node 1 being the root, our algorithm would start from the leaves 5 and 6, then compute $\mathscr{C}_{Z_3} = \mathscr{C}Z_{3,1}$ by combining $\mathscr{C}_{R_3}$ and $\mathscr{C}_{R_5}$, compute $\mathscr{C}_{Z_4} = \mathscr{C}Z_{4,1}$ by combining $\mathscr{C}_{R_4}$ and $\mathscr{C}_{R_6}$, compute $\mathscr{C}_{Z_{2,1}} = \mathscr{C}_{\{B,C,D,E\}}$ by combining $\mathscr{C}_{R_2}$ and $\mathscr{C}_{Z_3}$, then compute $\mathscr{C}_{Z_2} = \mathscr{C}_{Z_{2,2}} = \mathscr{C}_{\{B,C,D,E,F,G\}}$ by combining $\mathscr{C}_{Z_{2,1}}$ and $\mathscr{C}_{Z_4}$, and finally compute $\mathscr{C}_{Z_1}$ by combining $\mathscr{C}_{R_1}$ and $\mathscr{C}_{Z_2}$.

Theorem 4.4.13. *Deciding the existence of pure equilibrium in symmetric AGG-∅s with bounded treewidth is in P.*

Proof. Suppose the treewidth of the AGG is bounded by a constant, w. Then a tree decomposition of $und(G)$ having width at most w can be constructed in time exponential only in w, i.e., in polynomial time (see e.g. [Bodlaender, 1996, Kloks, 1994]). Then we can apply Lemma 4.4.11 to construct in polynomial time the tree decomposition $(\{X_i|i \in I\}, T = (I,F))$ of the primal graph G' and $\{R_i|i \in I\}$.

It is straightforward to check that our algorithm above correctly computes all $\mathscr{C}_{Z_{i,j}}$. Specifically, at step 3c, since $Z_{i,j-1}$ and Z_{q_j} correspond to disjoint subgraphs of T connected by edge $\{i,q_j\} \in F$, we have $Z_{i,j-1} \cup Z_{q_j} \subseteq R_i$. Therefore we can apply Lemma 4.4.9. Since $Z_r \equiv \mathscr{A}$, the algorithm correctly determines the existence of pure equilibrium in Γ.

The running time of the algorithm is polynomial in the size of the $\mathscr{C}_{Z_i}$'s. The size of each $\mathscr{C}_{Z_i}$ is bounded by $n||\Gamma||^2|\mathscr{C}[X_i]|$. Since the tree decomposition has width at most $(w+1)(\mathscr{I}+1)-1$, $|\mathscr{C}[X_i]| \leq \binom{n+(w+1)(\mathscr{I}+1)}{(w+1)(\mathscr{I}+1)}$. The latter is the number of ordered combinatorial compositions of n into $(w+1)(\mathscr{I}+1)+1$ nonnegative integers. An equivalent way of counting this number is as follows:

1. break n into $w+1$ nonnegative integers $x_1,\ldots,x_{w+1}$ such that $\sum_{i=1}^{w+1} x_i = n$.

2. then break each of the first w integers into $\mathscr{I}+1$ nonnegative parts in the same way, and the last one (x_{w+1}) into $\mathscr{I}+2$ nonnegative parts.

There are $\binom{n+w}{w}$ different ways of carrying out step 1. Since each integer considered in step 2 is at most n, there are at most $\binom{n+\mathscr{I}+1}{\mathscr{I}+1}$ ways of breaking each

integer. Therefore $\binom{n+(w+1)(\mathscr{I}+1)}{(w+1)(\mathscr{I}+1)} \leq \binom{n+w}{w}\binom{n+\mathscr{I}+1}{\mathscr{I}+1}^{w+1}$. Since w is a constant, this is polynomial in $||\Gamma||$. Hence our algorithm runs in polynomial time. □

When the input is an AGG-$\emptyset$ encoding of a singleton congestion game, i.e., a symmetric AGG-$\emptyset$ with only self-edges, the resulting $und(G)$ has treewidth 0 and by Theorem 4.4.13 the existence of PSNE can be determined in polynomial time. Of course, our result applies to a much larger class of games. Road games (Example 4.4.2) have treewidth 2 for all m. Thus by Theorem 4.4.13 the existence of PSNE can be determined in polynomial time for these games.

Our approach can be straightforwardly extended to the computation of related solution concepts such as pure-strategy ε-Nash equilibrium and strict equilibrium. For example, for pure-strategy ε-Nash equilibrium, we define partial solutions such that they induce ε-Nash equilibria of the corresponding restricted games, and use a modified version of Lemma 4.4.9 where the conditions that compare best entrance utilities and worst current utilities are relaxed by ε; e.g., $U_c' \geq U_e''$ is replaced by $U_c' + \varepsilon \geq U_e''$.

4.4.5 Finding PSNE

So far we have focused on the problem of deciding the existence of PSNE. Our dynamic programming approach can also be used to find these equilibria if they exist. We first consider the problem of constructing a single PSNE. After the bottom-up pass of the tree decomposition as discussed above, if $\mathscr{C}_{Z_r}$ is not empty, we do a top-down pass as follows:

1. Initialize Done $\subseteq I$ to be $\{r\}$,
2. Pick an arbitrary $(c[X_r], B_r, U_c^r, U_e^r) \in \mathscr{C}_{Z_r}$
3. Set $\mathscr{C}_{Z_r} = \{(c[X_r], B_r, U_c^r, U_e^r)\}$,
4. While Done $\neq I$:
   (a) Take $i \in$ Done such that $\{i' | i' \text{ is a child of } i\} \cap \text{Done} = \emptyset$
   (b) Let $q_1, \ldots, q_m$ be the children of i.
   (c) $\mathscr{C}_{Z_i} \equiv \mathscr{C}_{Z_{i,m}}$ will have a single element $(c[X_i], B_i, U_c^i, U_e^i)$.

(d) Let $\mathscr{C}_{Z_{i,0}} := \mathscr{C}_{R_i} = \{\mathrm{ch}(c[X_i])\}$

(e) For each $j \in m, m-1, \ldots, 1$:

i. pick $(c[X_{q_j}], B_{q_j}, U_c^{q_j}, U_e^{q_j}) \in \mathscr{C}_{Z_{q_j}}$ and $(c[X_i], B_{i,j-1}, U_c^{i,j-1}, U_e^{i,j-1}) \in \mathscr{C}_{Z_{i,j-1}}$ such that they combine to form the single element of $\mathscr{C}_{i,j}$ while satisfying the conditions of Lemma 4.4.9.

ii. set $\mathscr{C}_{Z_{q_j}} := \{(c[X_{q_j}], B_{q_j}, U_c^{q_j}, U_e^{q_j})\}$ and $\mathscr{C}_{Z_{i,j-1}} := \{(c[X_i], B_{i,j-1}, U_c^{i,j-1}, U_e^{i,j-1})\}$.

iii. add q_j to Done.

5. Now each $\mathscr{C}_{R_i}$ contains a single element $\mathrm{ch}(c[X_i])$. Output configuration $\bigcup_{i \in I} c[X_i]$.

Since the bottom-up pass has established the correct $\mathscr{C}_{Z_{i,j}}$, step 4(e)i can always be carried out. Therefore the algorithm is correct, and by the same argument as in the proof of Theorem 4.4.13 the algorithm runs in polynomial time. This proves:

Corollary 4.4.14. *The problem of finding a PSNE is in P for symmetric AGG-$\emptyset$s with bounded treewidth.*

A similar top-down pass would make sure that each $\mathscr{C}_{Z_{i,j}}$ contains exactly the characteristics of extendable partial solutions. Although the number of pure equilibria of an AGG could be exponential in the representation size $||\Gamma||$, the resulting set of $\mathscr{C}_{Z_{i,j}}$ along with the tree decomposition constitutes a *succinct description* of the set of PSNE of the game, analogous to Daskalakis and Papadimitriou [2006]'s construction of succinct descriptions of the set of PSNE of graphical games. Given a symmetric AGG-$\emptyset$ with bounded treewidth, such a succinct description can be computed in polynomial time. The succinct description can be used e.g., to enumerate the set of all PSNE in time polynomial in the size of input and output, and to check if there exists a PSNE with a specific configuration at certain action nodes.

4.4.6 Computing Optimal PSNE

Recall from Chapter 2 that the social welfare is the sum of the players' utilities. Given a configuration c in a symmetric AGG-$\emptyset$ Γ, the social welfare can be written as

$$W_\Gamma(c) = \sum_{\alpha \in \mathscr{A}} c[\alpha] u^\alpha(c[\nu(\alpha)]).$$

Our algorithm can be extended to compute the socially optimal PSNE if one exists. The characteristics now also store the social wealth of the restricted games. Specifically, we use the characteristic function

$$\mathrm{ch}^{\mathrm{opt}}(c[Z_{i,j} \cup \nu(Z_{i,j})]) = (\mathrm{ch}_{R_i,X_i}(c[Z_{ij} \cup \nu(Z_{i,j})]), W_{\Gamma'}(c[Z_{i,j}]))$$

where $\Gamma' = \Gamma(\#c[Z_{ij}], Z_{ij}, c[\nu(Z_{ij})])$ is the restricted game on Z_{ij} induced by the partial solution. Let $\mathscr{C}^{\mathrm{opt}}_{Z_{i,j}}$ be the corresponding set of characteristics.

The way characteristics from two sets $X', X'' \subseteq \mathscr{A}$ are combined is also slightly different from Lemma 4.4.9. Once we have checked consistency and profitable deviations as in Lemma 4.4.9, we now need to compute the social welfare of the resulting characteristic from the given characteristics of X' and X''. Simply adding the social welfare values would not be correct due to the possible overlap of X' and X''; fortunately we know the configuration over $X' \cap X''$ and their neighbors (by assumption of Lemma 4.4.9) so we are able to calculate the social welfare of the overlap and subtract it from the sum.

Corollary 4.4.15. *Suppose $X = X' \cup X''$, and $X', X'', P, P', P'', Q, Q', Q''$ satisfy the prerequisites of Lemma 4.4.9. For all $c[Q], B, U_c, U_e, W_\cup$, we have $(c[Q], B, U_c, U_e, W_\cup) \in \mathscr{C}^{opt}_X$ if and only if there exist $(c'[Q'], B', U'_c, U'_e, W') \in \mathscr{C}^{opt}_{X'}$ and $(c''[Q''], B'', U''_c, U''_e, W'') \in \mathscr{C}^{opt}_{X''}$ satisfying the conditions of Lemma 4.4.9, and*

$$W_\cup = W' + W'' - W_{\Gamma_\cap}(c[X' \cap X''])$$

where $\Gamma_\cap = \Gamma(\#c[X' \cap X''], X' \cap X'', c[\nu(X' \cap X'')])$.

Using this characteristic function together with the bottom-up pass above, we can compute the optimal social welfare achieved by a PSNE, if one exists. A top-down pass then constructs such a PSNE. One issue with this approach is that due to the additional social welfare term in a characteristic, the number of characteristics in each $\mathscr{C}^{\mathrm{opt}}_{Z_{i,j}}$ can be greater than $|\mathscr{C}_{Z_{i,j}}|$. Fortunately, it is straightforward to show that:

Lemma 4.4.16. *Suppose partial solutions $c[X \cup \nu(X)]$ and $c'[X \cup \nu(X)]$ induce the same characteristic under ch^{opt} except that the former's social welfare is less than*

the latter's. Then the former can be extended to a PSNE if and only if the latter can be extended to a PSNE with greater social welfare.

This implies that whenever we have multiple characteristics in $\mathscr{C}^{\text{opt}}_{Z_{i,j}}$ that differ only in their social welfare values, we can safely prune away all but the one with the greatest social welfare. The resulting $\mathscr{C}^{\text{opt}}_{Z_{i,j}}$ has the same cardinality as $\mathscr{C}_{Z_{i,j}}$, therefore the algorithm runs in polynomial time.

Corollary 4.4.17. *Computing a maximum social welfare PSNE in symmetric AGG-$\emptyset$s with bounded treewidth is in P.*

4.5 Beyond symmetric AGGs

4.5.1 Algorithm for k-Symmetric AGG-$\emptyset$s

Our results for symmetric AGG-$\emptyset$s can be straightforwardly extended to k-symmetric AGG-$\emptyset$s with bounded k. Consider a k-symmetric AGG-$\emptyset$ Γ with player classes $N_1,\ldots,N_k$. As discussed in Section 4.3, it is sufficient to consider k-configurations. Define restricted game $\Gamma((n'_\ell)_{1\le\ell\le k},X,(c_\ell[\nu_\ell(X)])_{1\le\ell\le k})$ to be the k-symmetric AGG-$\emptyset$ played on G_X, in which each player class $\ell\in\{1\ldots k\}$ has $n'_\ell\le|N_\ell|-c_\ell[\nu(X)]$ players, and the utility function for each $\alpha\in X$ is $u^\alpha|_{(c_\ell(\nu(X)))_{1\le\ell\le k}}$, i.e., the same as u^α of Γ except that the configuration of nodes outside X are given by the k-configuration $(c_\ell(\nu(X)))_{1\le\ell\le k}$. We define a partial solution on X to be a k-configuration $(c_\ell[X\cup\nu(X)])_{1\le\ell\le k}$ such that $(c_\ell[X])_{1\le\ell\le k}$ is a PSNE of the restricted game $\Gamma((\#c_\ell[X])_{1\le\ell\le k},X,(c_\ell[\nu_\ell(X)])_{1\le\ell\le k})$.

Similarly, we extend the characteristic functions of Section 4.4 by replacing each component of the characteristic with its k-tuple version.

Definition 4.5.1. *Given a restricted game Γ' on $X\subset\mathscr{A}$ and a PSNE $(c^*_\ell)_{1\le\ell\le k}$ of Γ', player class ℓ's* worst current utility $WCU_\ell((c^*_\ell)_{1\le\ell\le k},\Gamma')$ *is the utility of the worst-off player from class ℓ in Γ', or ∞ if Γ' has 0 players in class ℓ. Player class ℓ's* best entrance utility $BEU_\ell((c^*_\ell)_{1\le\ell\le k},\Gamma')$ *is the best payoff an* outside player *(a player currently playing an action outside of $X\cup\nu(X)$) from class ℓ can get by playing an action in $X\cap\mathscr{A}^\ell$, assuming the current players in Γ' play $(c^*_\ell)_{1\le\ell\le k}$. If there are 0 outside players from class ℓ or $X\cap\mathscr{A}^\ell=\emptyset$, $BEU((c^*_\ell)_{1\le\ell\le k},\Gamma')=-\infty$.*

Lemma 4.5.2. *Given a k-symmetric AGG-∅ Γ, $X \subset \mathscr{A}$, $P \subseteq X$ such that $P \supseteq \tau(X)$, and $Q \supseteq P \cup \nu(P)$, consider the characteristic function $ch^k_{P,Q}$ that maps a partial solution $(c_\ell[X \cup \nu(X)])_{1\leq\ell\leq k}$ to*

$$(c_\ell[Q], \#c_\ell[X], WCU_\ell(c[X'], \Gamma'), BEU_\ell(c[X'], \Gamma'))_{1\leq\ell\leq k},$$

where $\Gamma' = \Gamma((\#c_\ell[X'])_{1\leq\ell\leq k}, X', (c_\ell[\nu(X')])_{1\leq\ell\leq k})$ and $X' = X \setminus P$. Then $ch^k_{P,Q}$ is equilibrium-preserving.

Lemma 4.4.9 can be similarly extended to the k-symmetric case. Therefore we can use this characteristic function together with our bottom-up pass algorithm to determine the existence of PSNE in k-symmetric AGG-∅s, and use the top-down algorithm to find a PSNE if one exists. For k-symmetric AGG-∅s with bounded k and bounded treewidths, each of the k components of $\mathrm{ch}^k_{R_i,X_i}$'s output can take at most $poly(||\Gamma||)$ values, and as a result the number of characteristics is polynomial in $||\Gamma||$. We thus have the following generalization of Theorem 4.4.13, Corollary 4.4.14 and Corollary 4.4.17.

Corollary 4.5.3. *For k-symmetric AGG-∅s with bounded k and bounded treewidths, the problems of determining the existence of PSNE, of constructing a PSNE, and of finding maximum social welfare PSNE are all in P.*

We observe that when $k = 1$, i.e., when the game is symmetric, $\mathrm{ch}^k_{P,Q}$ degenerates into $\mathrm{ch}_{P,Q}$ we previously defined for the symmetric case, and this algorithm simplifies into our algorithm for symmetric AGG-∅s.

4.5.2 General AGG-∅s and the Augmented Action Graph

We now consider the case of general AGG-∅s. We note that such games can still be viewed as k-symmetric (with k at most n), but now k may grow with the input size. Our approach in Section 4.5.1 for k-symmetric AGG-∅s works well only when k is bounded by a constant, since the number of characteristics under $\mathrm{ch}^k_{P,Q}$ grows exponentially in k. Can this approach be extended to the case of general AGG-∅s? We observe that in order to check deviations out of and into $X \setminus P$, we do not need to keep track of information about player classes whose action sets are either (1) fully contained in $X \setminus P$, or (2) disjoint from $X \setminus P$. In the former case, no player

of that class can deviate outside $X \setminus P$; this is reflected in $\text{ch}^k_{P,Q}$ as best entrance utilities of $-\infty$ for that class in the restricted game on $X \setminus P$, but we also do not need to keep track of the worst current utilities for the class. Similarly, in the latter case, no player of that class can deviate into $X \setminus P$. To check deviations out of and into P, we only need to keep track of information on player classes whose action sets intersect Q. In other words, it is sufficient to define a characteristic function in terms of the player classes that are relevant to the current subset of nodes. Formally,

Lemma 4.5.4. *Consider a k-symmetric AGG-0* Γ *with player classes* $1,\ldots,k$ *corresponding to sets of players* $N_1,\ldots,N_k$ *and action sets* $\mathscr{A}^1,\ldots,\mathscr{A}^k$*. Given* $X \subset \mathscr{A}$*,* $P \subseteq X$ *such that* $P \supseteq \tau(X)$*, and* $Q \supseteq P \cup \nu(P)$*, let* $L(X,P) = \{\ell | 1 \leq \ell \leq k, \mathscr{A}^\ell \not\subseteq (X \setminus P), \mathscr{A}^\ell \cap (X \setminus P) \neq \emptyset\}$*, and let* $K(Q) = \{\ell | 1 \leq \ell \leq k, \mathscr{A}^\ell \cap Q \neq \emptyset\}$*. Consider the characteristic function* $ch^+_{P,Q}$ *that maps a partial solution* $(c_\ell[X \cup \nu(X)])_{1\leq\ell\leq k}$ *to*

$$\left((c_\ell[Q])_{\ell\in K(Q)}, (\#c_\ell[X \setminus P])_{\ell\in L(X,P)}, (WCU_\ell(c[X'],\Gamma'))_{\ell\in L(X,P)}, (BEU_\ell(c[X'],\Gamma'))_{\ell\in L(X,P)}\right),$$

where $\Gamma' = \Gamma((\#c_\ell[X'])_{1\leq\ell\leq k}, X', (c_\ell[\nu(X')])_{1\leq\ell\leq k})$ *and* $X' = X \setminus P$*. Then* $ch^+_{P,Q}$ *is equilibrium-preserving.*

Lemma 4.4.9 can be similarly extended. The number of characteristics under $\text{ch}^+_{P,Q}$ is exponential in $|Q|$, $|K(Q)|$ and $|L(X,P)|$. Intuitively, as we combine these characteristics to form characteristics on larger subgraphs, $|L(X,P)|$ will also grow, unless we "finish off" certain player classes, i.e., player class ℓ such that $\mathscr{A}^\ell$ become a subset of $X \setminus P$. Can we divide the action graph and combine the restricted games in a way that keeps $|Q|$, $|K(Q)|$ and $|L(X,P)|$ small? A natural idea is to turn to tree decompositions of G, as we did in Section 4.4.4. However, [Daskalakis et al., 2009] proved that the problem of determining the existence of PSNE is NP-hard even for AGGs with tree-width 1 and constant in-degree. In other words, we cannot hope for a polynomial-time algorithm for general AGGs with constant treewidths, unless P = NP. On the other hand, there exist classes of asymmetric AGGs that are poly-time solvable, e.g. those corresponding to tree graphical games. This implies that looking at the action graph alone is insufficient for identifying such tractable classes of AGGs. We have seen that information about the action sets of the AGG

is needed in order to define $\mathrm{ch}^{+}_{P,Q}$. Thus a natural idea is to define an object that incorporates information about the action sets as well as the action graph of the AGG.

Definition 4.5.5. *Given an AGG-$\emptyset$ Γ with player classes $1,\ldots,k$, define the* augmented action graph[2] *to be a directed graph*

$$AG = (V^+, E^+) = (\mathscr{A} \cup \{1,\ldots,k\}, E \cup \{(\ell,\alpha) | (\{\alpha\} \cup \nu(\alpha)) \cap \mathscr{A}^\ell \neq \emptyset\}).$$

Let $\mathscr{I}^+$ be the maximum in-degree of AG.

In other words, we add to the action graph G new vertices $\{1,\ldots,k\}$ corresponding to the player classes, and an edge from each player class ℓ to action α if α or any of its neighbors are in the action set of class ℓ. Intuitively, the edges from player class nodes to action nodes in the augmented action graph ensure that in the resulting tree decomposition, the set of tree nodes to which a player class is relevant forms a connected subgraph of the tree. This is formalized in the following result for augmented action graphs, which is analogous to Lemma 4.4.11 for action graphs.

Lemma 4.5.6. *Given a k-symmetric AGG-$\emptyset$ Γ whose augmented action graph AG has treewidth w, there exists a tree decomposition $(\{X_i \cup K_i | X_i \subseteq \mathscr{A}, K_i \subseteq \{1,\ldots,k\}, i \in I\}, T = (I,F))$ of AG's primal graph AG′ of width at most $(w+1)(\mathscr{I}^+ + 1) - 1$, and $\{R_i \subseteq \mathscr{A} | i \in I\}$ such that*

1. *$\bigcup_{i \in I} R_i = \mathscr{A}$, and $R_i \cup \nu(R_i) \subseteq X_i$ for all $i \in I$,*
2. *Let $J \subset I$ such that T_J is a connected graph and connects to the rest of the tree via only one edge $\{j, j'\} \in F$ with $j \in J$. Let $Y_J = \bigcup_{i \in J} R_i$. Then $\tau(Y_J) \subseteq R_j$, $K(X_j) \subseteq K_j$, and $L(Y_J, R_j) \subseteq L_j$.*

Proof. The construction is very similar to that of Lemma 4.4.11: given a tree decomposition $(\{R_i \cup L_i | R_i \subseteq \mathscr{A}, L_i \subseteq \{1,\ldots,k\}, i \in I\}, T = (I,F))$ of AG, we build

[2] We note that our definition of augmented action graph is different from the augmented graph of Daskalakis et al. [2009]. The computational problem that Daskalakis et al. [2009] were trying to solve (finding approximate mixed-strategy Nash equilibria) is different from the PSNE problem considered in this chapter.

a tree decomposition of AG' by adding to each tree node $i \in I$ the neighboring vertices of R_i (vertices in L_i have no neighbors). Lemma 4.5 of [Daskalakis and Papadimitriou, 2006] ensures that the result is a tree decomposition of AG' with width at most $(w+1)(\mathscr{I}^+ + 1) - 1$. The resulting tree decomposition $(\{X_i \cup K_i | X_i \subseteq \mathscr{A}, K_i \subseteq \{1,\ldots,k\}, i \in I\}, T = (I,F))$ will have $X_i = R_i \cup \nu(R_i)$ as in the proof of Lemma 4.4.11, and $K_i = L_i \cup \{\ell | 1 \leq \ell \leq k, \mathscr{A}^\ell \cap (R_i \cup \nu(R_i)) \neq \emptyset\}$. This implies $K(X_i) \subseteq K_i$ for all $i \in I$.

By the same argument as in the proof of Lemma 4.4.11, we have $\tau(Y_J) \subseteq R_j$. It remains to show that $L(Y_J, R_j) \subseteq L_j$. Consider an arbitrary $\ell \in L(Y_J, R_j)$. This implies that $\mathscr{A}^\ell \cap (Y_J \setminus R_j) \neq \emptyset$ and $\mathscr{A}^\ell \cap (\overline{Y_J \setminus R_j}) \neq \emptyset$. But this implies that there exists $\alpha \in (Y_J \setminus R_j)$ such that $(\ell, \alpha) \in E^+$, and there exists $\alpha' \in \overline{Y_J \setminus R_j}$ such that $(\ell, \alpha') \in E^+$. Since the tree nodes that contain α must be in $J \setminus \{j\}$, and by condition 2 of Definition 4.4.10 (ℓ, α) must be contained in some tree node, we must have that $\ell \in L_i$ for some $i \in J \setminus \{j\}$. Similarly we must have $\ell \in L_{i'}$ for some $i' \in \overline{J \setminus \{j\}}$. But by condition 3 of Definition 4.4.10 we must have $\ell \in L_j$, and therefore $L(Y_J, R_j) \subseteq L_j$. □

Lemma 4.5.6 implies that we can apply the bottom-up pass algorithm using the characteristic function $\mathrm{ch}^+_{R_i,X_i}$ for $Z_{i,j}$, and correctly determines the existence of PSNE. If a PSNE exists then a top-down pass constructs one. Let us consider the running time of this approach. If we assume that AG has bounded indegree and bounded treewidth, this immediately implies that $|X_i|$ and K_i are bounded for all $i \in I$, and the number of characteristics are polynomial in n and $|\mathscr{A}|$. This in turn implies that our algorithm runs in polynomial time in this case.

Proposition 4.5.7. *For AGG-∅s whose action graphs have bounded indegree and bounded treewidth, the problems of determining the existence of PSNE and of finding a PSNE are in P.*

One question is whether it is possible to show that the run time is polynomial in the input size when the augmented action graph has bounded treewidth, i.e., without any requirement on the in-degree. However, this turns out to be more difficult than in the symmetric case. Specifically, in order to prove such a result without any requirement on the in-degree, we would need to compare the runtime with (a lower

bound of) the input size. Whereas for symmetric AGGs we have exact estimates of the input size, for general AGGs we only proved upper bounds in Chapter 3. The complexity of PSNE for AGG-$\emptyset$s with bounded-treewidth augmented action graphs remains an open problem.

One interesting case is when the input is an AGG-$\emptyset$ encoding of a bounded-treewidth graphical game. Recall that the PSNE problem for such games is known to be tractable [Daskalakis and Papadimitriou, 2006, Gottlob et al., 2005]. We show that our algorithm runs in polynomial time given AGG-$\emptyset$ encodings of such games, thus providing another proof of this result.

Proposition 4.5.8. *Determining the existence of PSNE in bounded-treewidth graphical games is in P.*

Proof. Recall from Chapter 3 that the size of the AGG-$\emptyset$ encoding is proportional to that of the graphical game, which is $\Theta(\sum_{\ell \in N} |A_\ell| \prod_{j \in \nu_g(\ell)} |A_j|)$, where $\nu_g(\ell)$ is the set of neighboring players of ℓ in the graphical game. The AGG has $k = n$ player classes, each containing a single player. We denote by ℓ the player class corresponding to player $\ell \in N$. Suppose the underlying graph (N, E_g) of the graphical game has treewidth w and maximum in-degree $\mathscr{I}_g$. Then the corresponding action graph $G = (\mathscr{A}, E)$ is given in Chapter 3 and the corresponding augmented action graph $AG = (\mathscr{A} \cup N, E \cup \{(\ell, \alpha) | \ell \in N, \alpha \in A_\ell\})$. Given a tree decomposition $(\{L_i | i \in I\}, T = (I, F))$ of the graph (N, E_g) with width w, it is straightforward to show that $(\{R_i \cup L_i | i \in I\}, T)$, where $R_i = \bigcup_{\ell \in L_i} A_\ell$ for all $i \in I$, is a tree decomposition of the augmented action graph AG. The width of this decomposition is $O(\max_{\ell \in N} |A_\ell| w)$.

Construct the tree decomposition $(\{X_i \cup K_i | i \in I\}, T)$ for the primal graph AG' according to Lemma 4.5.6. It is straightforward to verify that $K_i = L_i \cup \nu_g(L_i)$ and $X_i = \bigcup_{\ell \in K_i} A_\ell$. Therefore $|K_i| = O(w\mathscr{I}_g)$, $|X_i| = O(\max_{\ell \in K_i} |A_\ell| w\mathscr{I}_g)$, and the width of the decomposition is $O(\max_{\ell \in N} |A_\ell| w\mathscr{I}_g)$.

Now consider the number of characteristics under $\mathrm{ch}^+_{R_i, X_i}$. Since for each $\ell \in N$ and $J \subseteq I$ we either have $A_\ell \subseteq Y_J$ or $A_\ell \cap Y_J = \emptyset$, this implies that $L(Y_J, R_j) = \emptyset$ for all $j \in I$ and $J \subseteq I$. Thus the only nontrivial component of the characteristic is $(c_\ell[X_i])_{\ell \in K_i}$. Since each $\ell \in K_i$ corresponds to a single player, $|C_\ell[X_i]| = |A_\ell|$. Thus the number of possible $(c_\ell[X_i])_{\ell \in K_i}$ is $\prod_{\ell \in K_i} |A_\ell|$, which is polynomial in the input

size since $|K_i| = O(w\mathscr{I}_g)$. Thus the number of characteristics is polynomial in $||\Gamma||$, which implies that our algorithm runs in polynomial time. □

We see from the above proof that in this case the characteristic degenerates into $(c_\ell[X_i])_{\ell \in K_i}$, which carries the same amount of information as the partial pure strategy profile of players in K_i. This is exactly the same sufficient statistic used by Daskalakis and Papadimitriou [2006]'s algorithm for graphical games, and as a result our algorithm simplifies to the equivalent of Daskalakis and Papadimitriou [2006]'s algorithm when given an AGG-$\emptyset$ encoding of a graphical game.

We also note that our algorithms for symmetric and k-symmetric AGG-$\emptyset$s can be seen as special cases of our augmented-action-graph-based algorithm. In particular, consider a k-symmetric AGG-$\emptyset$ with action graph G, and suppose $und(G)$ has a tree decomposition $(\{R_i | i \in I\}, T = (I,F))$. Then our algorithm for k-symmetric AGG-$\emptyset$s corresponds to applying the augmented-action-graph-based algorithm to the tree decomposition $(\{R_i \cup \{1,\ldots,k\} | i \in I\}, T)$ for AG', i.e., having all k player classes in each of the tree nodes of the decomposition.

4.6 Conclusions and Open Problems

In this chapter we analyzed the problem of computing PSNE in AGGs. We proposed a dynamic programming algorithm and showed that for symmetric AGG-$\emptyset$s with bounded treewidth, our algorithm determines the existence of PSNE in polynomial time. We extended our approach to certain classes of asymmetric AGG-$\emptyset$s, and showed that our algorithm generalizes existing dynamic-programming approaches for computing PSNE in graphical games and singleton congestion games.

One question is whether our approach has captured all the tractable classes of AGG-$\emptyset$s for the PSNE problem. The answer is no. For example, consider an asymmetric AGG-$\emptyset$ whose action graph has no inter-vertex edges and only self edges. This is the same as the singleton congestion games studied by Ieong et al. [2005] except that here the game is not symmetric. It is straightforward to see that this game corresponds to a congestion game, and thus PSNE always exist. Furthermore, by a similar argument as Ieong et al. [2005], given such a game a PSNE can be found by iterated best response dynamics in polynomial time. On the other hand, the augmented graph of such an AGG-$\emptyset$ might have large treewidth.

This example can be generalized: if the action graph contains a set X of such singleton nodes, and the action sets that intersect X does not contain any node not in X, then the subgraph of the singleton nodes does not affect the existence of PSNE, i.e., a PSNE exists in the game if and only if a PSNE exists in the restricted game on the rest of the graph. We can even further generalize this: consider a subgraph G_X such that (as above) the action sets that intersect X does not contain any node not in X, and that X has only incoming edges from the rest of G and no out going edges (i.e., $\nu(\overline{X}) = \emptyset$), then G_X does not affect the existence of PSNE, and we can safely delete the subgraph and solve the rest of the graph. This process can be repeated. (This is analogous to, and indeed a generalization of, the case of graphical games with sinks which was discussed in [Jiang and Safari, 2010].) Note that for these examples, a greedy approach is used instead of (or in addition to) the dynamic programming approach used in this chapter. For the problem of existence of PSNE in graphical games, Jiang and Safari [2010] was able to completely characterize the tractable classes of bounded-indegree graphs. An open problem is completely characterizing the types of restrictions to the graphical structure of AGG-$\emptyset$s that make the PSNE problem tractable, perhaps by leveraging some of the techniques developed in [Jiang and Safari, 2010].

Another future direction is to extend our approach to AGG-FNs. Recall that the configuration on a function node is the value of a deterministic function of the configuration of its neighbors. Thus given a symmetric AGG-FN, its PSNE correspond to configurations over its action nodes and function nodes such that the configuration over each function node is equal to the appropriate value, and the configuration over action nodes satisfy the incentive and consistency constraints as before. Assuming the deterministic functions for the function nodes are explicitly represented, it is then relatively straightforward to extend our dynamic-programming approach to work on the action graphs of symmetric AGG-FNs. An interesting question is whether this can be extended to efficiently deal with compactly-represented function nodes such as summation function nodes. Finally, as we have seen in this chapter, one faces additional technical challenges when going beyond the symmetric case. It would be interesting to see if our approaches discussed in Section 4.5 can be extended to AGG-FNs.

Chapter 5

Temporal Action-Graph Games: A New Representation for Dynamic Games

5.1 Introduction

In this chapter we[1] turn our focus to compact representations of dynamic games. As mentioned in Section 2.1.2, the most influential compact representation for imperfect-information dynamic games is multiagent influence diagrams, or MAIDs [Koller and Milch, 2003]. MAIDs are compact when players' utility functions exhibit independencies; such compactness can also be leveraged for computational benefit (see Section 2.2.4).

Consider the following example of a dynamic game.

Example 5.1.1. *Twenty cars are approaching a tollbooth with three lanes. The drivers must decide which lane to use. The cars arrive in four waves of five cars each. In each wave, the drivers must pick lanes simultaneously, and can see the number of cars before them in each lane. A driver's utility decreases with the number of cars that chose the same lane either before him or at the same time.*

[1] This chapter is based on published joint work with Kevin Leyton-Brown and Avi Pfeffer [Jiang et al., 2009].

A straightforward MAID representation of the game of Example 5.1.1 contains very little structure; in particular, each player will have a utility node, whose parents are all the decision nodes of the drivers before her. As the number of players grow, the representation size of the utility functions grow exponentially. Computation using such a representation would be highly inefficient. However, the game really is highly structured: agents' payoffs exhibit context-specific independence (utility depends only on the number of cars in the chosen lane) and agents' payoffs exhibit anonymity (utility depends on the numbers of other agents taking given actions, not on these agents' identities). The problem with a straightforward MAID representation of this game is that it does not capture either of these kinds of payoff structure.

As we have seen in Chapter 2, a wider variety of compact game representations exist for simultaneous-move games. In particular, several of these game representations (including congestion games and local effect games) can compactly represent anonymity and context-specific independence (CSI) structures. We saw in Chapter 3 that AGGs unify these past representations by compactly representing both anonymity and CSI while still retaining the ability to represent any game. Furthermore, structure in AGGs can be leveraged for computational benefit. However, AGGs are unable to represent the game presented in Example 5.1.1 because they cannot describe sequential moves or imperfect information.

In this chapter we present a new representational framework called Temporal Action-Graph Games (TAGGs) that allows us to capture this kind of structure. Like AGGs, TAGGs can represent anonymity and CSI, but unlike AGGs they can also represent games with dynamics, imperfect information and uncertainty. We first define the representation of TAGGs, and then show formally how they define a game using an induced Bayesian network (BN). We demonstrate that TAGGs can represent any MAID, but can also represent situations that are hard to capture naturally as MAIDs. If the TAGG representation of a game contains anonymity or CSI, the induced BN will have special structure that can be exploited by inference algorithms. We present an algorithm for computing expected utility of TAGGs that exploits this structure. Our algorithm first transforms the induced BN to another BN that represents the structure more explicitly, then computes expected utility using a specialized inference algorithm on the transformed BN. We show that it

performs better than using a MAID in which the structure is not represented explicitly, and better than using a standard BN inference algorithm on the transformed BN.

5.2 Representation

5.2.1 Temporal Action-Graph Games

At a high level, *Temporal Action-Graph Games (TAGGs)* extend the AGG representation by introducing the concepts of *time*, *uncertainty* and *imperfect information*, while adapting the AGG concepts of action nodes and action-specific utility functions to the dynamic setting. We first give an informal description of these concepts.

Temporal structure. A TAGG describes a dynamic game played over a series of time steps $1,\ldots,T$, on a set of action nodes $\mathscr{A}$. At each time step a version of a static AGG is played by a subset of agents on $\mathscr{A}$, and the action counts on the action nodes are accumulated.

Chance variables. TAGGs model uncertainty via *chance variables*. Like random variables in a BN, a chance variable is associated with a set of parents and a conditional probability table (CPT). The parents may be action nodes or other chance variables. Each chance variable is associated with an instantiation time; once instantiated, its value stays the same for the rest of the game. Chance variables can be thought of as a generalization of the (deterministic) function nodes in AGG-FNs.

Decisions. At each time step one or more agents move simultaneously, represented by agent-specific *decisions*. TAGGs model imperfect information by allowing each agent to condition his decision on observed values of a given subset of decisions, chance variables, and the previous time step's action counts.

Action nodes. Each decision is a choice of one from a number of available *action nodes*. As in AGGs, the same action may be available to more than one player. Action nodes provide a time-dependent tally: the *action count* for each action A in each time step τ is the number of times A has been chosen during the time period $1,\ldots,\tau$.

Utility functions. There is a *utility function* U_A^τ associated with each action A at each time τ, which specifies the utility a player receives at time τ for having chosen action A. Each U_A^τ has a set of parents which must be action nodes or chance variables. The utility of playing action A depends only on what happens over these parents. An agent who took action A (once) may receive utility at multiple times (e.g., short-term cost and long-term benefit); this is captured by associating a set of payoff times with each decision. An agent's overall utility is defined as the sum of the utilities received at all time steps.

Play of a TAGG can be summarized as follows:

1. At time 0, action counts are initialized to zero; chance variables with instantiation time 0 are instantiated,

2. At each time $\tau \in \{1, \ldots, T\}$:

   (a) all agents with decisions at τ observe the appropriate action counts, chance variables, and decisions, if any.

   (b) all decisions at τ are made simultaneously.

   (c) action counts at τ are tallied.

   (d) chance variables at time τ are instantiated.

   (e) for each action A, utility function U_A^τ is evaluated, with this amount of utility accruing to every agent who took action a at a decision whose payoff times include τ; the result is not revealed to any of the players.[2]

3. At the end of the game, each agent receives the sum of all utility allocations throughout the game.

Intuitively, the process can be seen as a sequence of simultaneous-move AGGs played over time. At each time step τ, the players that have a decision at time τ participate in a simultaneous-move AGG on the set of action nodes, whose action counts are initialized to be the counts at $\tau - 1$. Each action A's utility function is U_A^τ and A's neighbors in the action graph correspond to the parents of U_A^τ.

[2] If an agent plays action A for two decisions that have the same payoff time τ, then the agent receives twice the value of U_A^τ.

We observe that decisions and chance variables in TAGGs are similar to decision nodes and chance nodes (respectively) in MAIDs, except that here their parents can be time-dependent action counts. Thus the need to specify the time steps that decisions and chance nodes in a TAGG are instantiated; but once instantiated their values stay fixed. We also observe that the time-dependent nature of action counts in TAGGs is similar to how *dynamic Bayesian networks (DBNs)* [Dean and Kanazawa, 1989, Murphy, 2002], a probabilistic graphical model of temporal domains, model their time-dependent random variables. Just as a DBN can be unrolled into a BN; later on we will see that a TAGG can also be unrolled into a MAID.

Before formally defining TAGGs, we need to first define the concept of a *configuration* at time τ over a set of action nodes, decisions and chance variables, which is intuitively an instantiation at time τ of a corresponding set of variables.

Definition 5.2.1. *Given a set of action nodes $\mathscr{A}$, a set of decisions $\mathscr{D}$, a set of chance variables $\mathscr{X}$, and a set $B \subseteq \mathscr{A} \cup \mathscr{X} \cup \mathscr{D}$, a* configuration *at time τ over B, denoted as C_B^τ, is a $|B|$-tuple of values, one for each node in B. For each node $b \in B$, the corresponding element in C_B^τ, denoted as $C^\tau(b)$, must satisfy the following:*

- *if $b \in \mathscr{A}$, $C^\tau(b)$ is an integer in $\{0,\ldots,|\mathscr{D}|\}$ specifying the action count on b at τ, i.e. the number of times action b has been chosen during the time period $1,\ldots,\tau$.*
- *if $b \in \mathscr{D}$, $C^\tau(b)$ is an action in $\mathscr{A}$, specifying the action chosen at D.*
- *if $b \in \mathscr{X}$, $C^\tau(b)$ is a value from the domain of the random variable, $Dom[b]$.*

Let $\mathscr{C}_B^\tau$ be the set of all configurations at τ over B.

We now offer formal definitions of chance variables, decisions, and utility functions.

Definition 5.2.2. *A* chance variable *X is defined by:*

1. *a domain $Dom[X]$, which is a nonempty finite set;*
2. *a set of parents $Pa[X]$, which consists of chance variables and/or actions;*

3. *an instantiation time $t(X)$, which specifies the time at which the action counts in $Pa[X]$ are instantiated;*
4. *a CPT* $\Pr(X|Pa[X])$*, which specifies the conditional probability distribution of X given each configuration $C^{t(X)}_{Pa[X]}$.*

We require that each chance variable's instantiation time be no earlier than its parent chance variable's instantiation times, i.e. if chance variable $X' \in Pa[X]$, then $t(X') \leq t(X)$.

Definition 5.2.3. *A* decision *D is defined by:*

1. *the player making the decision, $pl(D)$. A player may make multiple decisions; the set of decisions belonging to a player ℓ is denoted by $Decs[\ell]$.*
2. *its* decision time *$t(D) \in \{1,\ldots,T\}$. Each player has at most one decision at each time step.*
3. *its* action set *$Dom[D]$, a nonempty set of actions.*
4. *the set of* payoff times *$pt(D) \subseteq \{1,\ldots,T\}$. We assume that $\tau \geq t(D)$ for all $\tau \in pt(D)$.*
5. *its* observation set *$O[D]$: a set of decisions, actions, and chance variables, whose configuration at time $t(D)-1$ (i.e. $C^{t(D)-1}_{O[D]}$) is observed by $pl(D)$ prior to making the decision. We require that if decision D' is an observation of D, then $t(D') < t(D)$. Furthermore if chance variable X is an observation of D, then $t(X) < t(D)$.*

Definition 5.2.4. *Each action A at each time τ is associated with one* utility function *U^τ_A. Each U^τ_A is associated with a set of parents $Pa[U^\tau_A]$, which is a set of actions and chance variables. We require that if chance variable $X \in Pa[U^\tau_A]$, then $t(X) \leq \tau$. Each utility function U^τ_A is a mapping from the set of configurations $\mathscr{C}^\tau_{Pa[U^\tau_A]}$ to a real value.*

We can now formally define TAGGs.

Definition 5.2.5. *A Temporal Action-Graph Game (TAGG) is a tuple $(N, T, \mathscr{A}, \mathscr{X}, \mathscr{D}, \mathscr{U})$, where:*

1. $N = \{1, \ldots, n\}$ *is a set of* players.
2. T *is the* duration *of the game.*
3. $\mathscr{A}$ *is a set of* actions.
4. $\mathscr{X}$ *is a set of* chance variables. *Let* $\mathscr{G}$ *be the induced directed graph over* $\mathscr{X}$. *We require that* $\mathscr{G}$ *be a directed acyclic graph (DAG).*
5. $\mathscr{D}$ *is the set of* decisions. *We require that each decision* D*'s action set* $Dom[D] \subseteq \mathscr{A}$.
6. $\mathscr{U} = \{U_A^\tau : A \in \mathscr{A}, 1 \leq \tau \leq T\}$ *is the set of* utility functions.

First, let us see how to represent Example 5.1.1 as a TAGG. The set N corresponds to the cars. The duration $T = 4$. We have one action node for each lane. For each time τ, we have five decisions, each belonging to a car that arrives at time τ. The action set for each decision is the entire set $\mathscr{A}$. The payoff time for each decision is the time the decision is made, i.e., $\text{pt}(D) = \{t(D)\}$. Each decision has all actions as observations. For each A and τ, the utility U_A^τ has A as its only parent. The representation size of each utility function is at most n; the size of the entire TAGG is $O(|\mathscr{A}|Tn)$.

The TAGG representation is useful beyond compactly representing MAIDs. The representation can also be used to specify information structures that would be difficult to represent in a MAID. For example, we can represent games in which agents' abilities to observe the decisions made by previous agents depend on what actions these agents took.

Example 5.2.6. *There are* $2T$ *ice cream vendors, each of which must choose a location along a beach. For every day from 1 to* T*, two of the vendors simultaneously set up their ice cream stands. Each vendor lives in one of the locations. When a vendor chooses an action, it knows the location of vendors who set up stands in previous days in the location where it lives or in one of the neighboring locations. The payoff to a vendor in a given day depends on how many vendors set up stands in the same location or in a neighboring location.*

Example 5.2.6 can be represented as a TAGG, the key elements of which are as follows. There is an action A for each location. Each player j has one decision

D_j, whose observations include actions for the location j lives in and neighboring locations. The payoff time for each decision is T, and the utility function U_A^T has A and its neighboring locations as parents.

Let us consider the size of a TAGG. It follows from Definition 5.2.5 that the space bottlenecks of the representation are the CPTs $\Pr(X|\text{Pa}[X])$ and the utility functions U_A^τ, which have polynomial sizes when the numbers of their parents are bounded by a constant.

Lemma 5.2.7. *Given TAGG $(N, T, \mathscr{A}, \mathscr{X}, \mathscr{D}, \mathscr{U})$, if $\max_{X\in\mathscr{X}} |Pa[X]|$ and $\max_{U\in\mathscr{U}} |Pa[U]|$ are bounded by a constant, then the size of the TAGG is bounded by a polynomial in $\max_{X\in\mathscr{X}} Dom[X]$, $|\mathscr{X}|$, $|\mathscr{D}|$, $|\mathscr{U}|$, and T.*

5.2.2 Strategies

In Section 2.2.4 we introduced the standard concepts of pure, mixed and behavior strategies in dynamic games. We now apply these concepts to the case of TAGGs. We start with *pure strategies*, where at each decision D, an action is chosen deterministically as a function of observed information, i.e., the configuration $C_{O[D]}^{t(D)-1}$. A *mixed strategy* of a player i is a probability distribution over pure strategies of i. Recall that since there can be an exponential number of pure strategies in a dynamic game, a mixed strategy is generally an exponential-sized object. We thus restrict our attention to *behavior strategies*, in which the action choices at different decisions are randomized independently.

Definition 5.2.8. *A behavior strategy at decision D is a function $\sigma^D : \mathscr{C}_{O[D]}^{t(D)-1} \to \varphi(Dom[D])$, where $\varphi(Dom[D])$ is the set of probability distributions over $Dom[D]$. A behavior strategy for player i, denoted σ_i, is a tuple consisting of a behavior strategy for each of her decisions. A behavior strategy profile $\sigma = (\sigma_1, \ldots, \sigma_n)$ consists of a behavior strategy σ_i for all i.*

An agent has *perfect recall* when she never forgets her action choices and observations at earlier decisions. The TAGG representation does not enforce perfect recall; TAGGs can represent perfect recall games as well as non-perfect-recall games. A technical issue on representing perfect recall games as TAGGs is the following: in order to preserve the perfect-recall property of the resulting TAGG,

each decision D of player i should observe all of i's earlier decisions and observations. However, recall that if an action A is in the observation set of one of i's earlier decisions at time $t' < t(D)$, it means that the action count at time $t'-1$ was observed. Directly including A in $O[D]$ would instead imply that the action count of A at time $t(D)-1$ is observed by D, in which case the information structure of the TAGG is different from the original game and is thus not a faithful representation. Instead, we model the situation by creating a deterministic chance variable $X_A^{t'-1}$ with instantiation time $t'-1$; its only parent is A and its value is the action count of A at time $t'-1$. We then include $X_A^{t'-1}$ in $O[D]$. It is straightforward to see that $X_A^{t'-1}$ carries the information equivalent to observing the action count of A at time $t'-1$, and the resulting TAGG provides a correct representation of the perfect recall game.

5.2.3 Expected Utility

Now we use the language of Bayesian networks to formally define an agent's expected utility in a TAGG given a behavior strategy profile σ. Specifically, we define an *induced BN* that formally describes how the TAGG is played out. Given a behavioral strategy profile, decisions, chance variables and utilities can naturally be understood as random variables. On the other hand, action counts are time dependent. Thus, we have a separate action count variable for each action at each time step.

Definition 5.2.9. *Let $A \in \mathscr{A}$ be an action and $\tau \in \{1,\dots,T\}$ be a time point. A^τ denotes the* action count variable *representing the number of times A was chosen from time 1 to time τ. Let A^0 be the variable which is 0 with probability 1.*

We would like to define expected utility for each player, which is the sum of expected utilities of the player's decisions. On the other hand, the utility functions in TAGGs are action specific. To bridge the gap, we create new decision-payoff variables in the induced BN that represent the utilities of decisions received at each of their payoff time points.

Definition 5.2.10. *Given a TAGG and a behavior strategy profile σ, the* induced BN *is defined over the following* variables*: for each decision $D \in \mathscr{D}$ there is a*

behavior strategy variable which by abuse of notation we shall also denote by D; for each chance variable $X \in \mathscr{X}$ there is a variable which we shall also denote by X; there is a variable A^τ for each action $A \in \mathscr{A}$ and time step $\tau \in \{1,...,T\}$; for each utility function U_A^τ for actions $A \in \mathscr{A}$ and time points $\tau \in \{1,...,T\}$, there is a utility variable also denoted by U_A^τ; for each decision D and each time $\tau \in pt(D)$, there is a decision-payoff variable u_D^τ.

We define the actual parents *of each variable V, denoted APa[V], as follows: The actual parents of a behavior strategy variable D are the variables corresponding to O[D], with each action $A_k \in O[D]$ replaced by $A_k^{t(D)-1}$. The actual parents of an action count variable A^τ are all behavior strategy variables D whose decision time $t(D) \leq \tau$ and $A \in Dom[D]$. The actual parents of a chance variable X are the variables corresponding to Pa[X], with each action $A_k \in Pa[X]$ replaced by $A_k^{t(X)}$. The actual parents of a utility variable U_A^τ are the variables corresponding to $Pa[U_A^\tau]$, with each action $A_k \in Pa[U_A^\tau]$ replaced by A_k^τ. where $\{A_1,...,A_\ell\} = Dom[D]$.*

The CPDs of chance variables are the CPDs of the corresponding chance variables in the TAGG. The CPD of each behavior strategy variable D is the behavior strategy σ^D. The CPD of each utility variable U_A^τ is a deterministic function defined by the corresponding utility function U_A^τ. The CPD of each action count variable A^τ is a deterministic function that counts the number of decisions in APa[A] that are assigned value A. The CPD of each decision-payoff variable u_D^τ is a multiplexer, *i.e. a deterministic function that selects the value of its utility variable parent according to the choice of its decision parent. For example, if the value of D is A_k, then the value of u_D^τ is the value of $U_{A_k}^\tau$.*

Theorem 5.2.11. *Given a TAGG, let $\mathscr{F}$ be the directed graph over the variables of the induced BN in which there is an edge from V_1 to V_2 iff V_1 is an actual parent of V_2. Then $\mathscr{F}$ is acyclic.*

This follows from the definition of TAGGs and the way we set up the actual parents in Definition 5.2.10.

By Theorem 5.2.11, the induced BN defines a joint probability distribution over its variables, which we denote by P^σ. Given σ, denote by $E^\sigma[V]$ the expected value of variable V in the induced BN. We are now ready to define the expected utility to

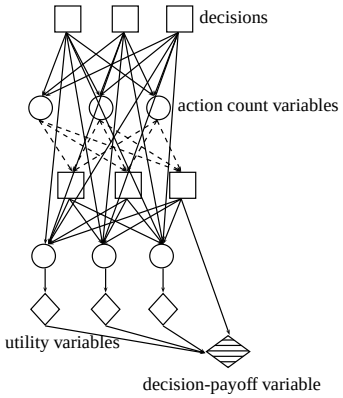

Figure 5.1: Induced BN of the TAGG of Example 5.1.1, with 2 time steps, 3 lanes, and 3 players per time step. Squares represent behavior strategy variables, circles represent action count variables, diamonds represent utility variables and shaded diamonds represent decision-payoff variables. To avoid cluttering the graph, we only show utility variables at time step 2 and a decision-payoff variable for one of the decisions.

players under behavior strategy profiles.

Definition 5.2.12. *The expected utility to player ℓ under behavior strategy profile σ is $EU^{\sigma}(\ell) = \sum_{D \in Decs[\ell]} \sum_{\tau \in pt(D)} E^{\sigma}[u_D^{\tau}]$.*

Figure 5.1 shows an induced BN of a TAGG based on Example 5.1.1 with six cars and three lanes. Note that although we use squares to represent behavior strategy variables, they are random variables and not actual decisions as in influence diagrams.

5.2.4 The Induced MAID of a TAGG

Given a TAGG we can construct a MAID that describes the same game. We use a similar construction as the induced Bayesian Network, but with two differences. First, instead of behavior strategy variables with CPDs assigned by σ, we have decision nodes in the MAID. Second, each decision-payoff variable u_D^{τ} becomes a utility node for player $\mathrm{pl}(D)$ in the MAID. The resulting MAID describes the same game as the TAGG, because it offers agents the same strategies and their expected utilities are defined by the same BN. We call this the *induced MAID* of the TAGG.

5.2.5 Expressiveness

It is natural to ask about the *expressiveness* of TAGGs: what games can we represent? It turns out that TAGGs are able to compactly represent all MAIDs.

Lemma 5.2.13. *Any MAID can be represented as a TAGG with the same space complexity.*

Proof. Recall that a MAID consists of a set of decisions, a set of chance nodes and a set of utility nodes. Given a MAID, we construct a TAGG in the following way:

- For each decision D' of the MAID and each value $d' \in \mathrm{Dom}[D']$, create an unique action $A_{d'}$ in the TAGG.
- Decisions and chance nodes of the MAID can be directly copied over to the TAGG.
- Utility nodes in MAIDs are player-specific: each utility node is associated with some player. Utility nodes in TAGGs are action specific. We can encode MAID utility nodes as TAGG utility nodes as follows: Given a MAID utility node U' associated with player j, create a dummy decision $D_{U'}$ belonging to player j, whose action set contains exactly one action $A_{U'}$. We then encode the utility function for U' in the MAID as the utility associated with action $A_{U'}$ in the TAGG.
- One difference between MAIDs and TAGGs is that in MAIDs decisions can be parents of chance and utility nodes; in TAGGs only chance variables and actions can be parents of chance and utility nodes. Nevertheless, MAID chance nodes and utility nodes can be encoded in TAGGs by replacing each decision parent D' by the corresponding set of actions in $\mathrm{Dom}[D']$.
- Decisions and chance nodes of MAIDs are not associated with time points. Nevertheless, since the MAID is a directed acyclic graph, we can assign decision times to decisions and instantiation times to chance variables that are consistent with the topological order of the MAID. The payoff times of each decision is assigned to be the singleton $\{T\}$, i.e. at the end of the game.

□

As a result, TAGGs can represent any extensive form game representable as a MAID. These include all perfect recall games, and the subclass of imperfect recall games where each information set does not involve multiple time steps.

Now consider the converse problem of reducing TAGGs to MAIDs. In this case, since the induced MAID of a TAGG is payoff equivalent to the TAGG, it trivially follows that any TAGG can be represented by a MAID. However, the induced MAID has a large in-degree, and can thus be exponentially larger than the TAGG. For example, in the games of Examples 5.1.1 and 5.2.6, the induced MAIDs have max in-degrees that are equal to the number of decisions, which implies that the sizes of the MAIDs grow exponentially with the number of decisions, whereas the sizes of the TAGGs for the same games grow linearly in the number of decisions. This is not surprising, since TAGGs can exploit more kinds of structure in the game (CSI, anonymity) compared to a straightforward MAID representation. In Section 5.3.1 we show that the induced MAID can be transformed into a MAID that explicitly represents the underlying structure. The size of the transformed MAID is polynomial in the size of the TAGG.

The TAGG representation is also a true generalization of AGGs, since any AGG-$\emptyset$ can be straightforwardly represented as a TAGG with $T = 1$. Function nodes in AGG-FNs and AGG-FNAs can be modeled as chance nodes with a deterministic CPT, thus AGG-FNs and AGG-FNAs can also be represented as TAGGs with $T = 1$.

5.3 Computing Expected Utility

In this section, we consider the task of computing expected utility $\mathrm{EU}^{\sigma}[j]$ to a player j given a mixed strategy profile σ. As mentioned in Section 2.2.4, computation of EU is an essential step in many game-theoretic computations for dynamic games, such as finding a best response given other players' strategy profile, checking whether a strategy profile is a Nash equilibrium, and heuristic algorithms such as fictitious play and iterated best response. In Section 5.4 we discuss extending our methods in this section to a subtask in the Govidan-Wilson algorithm for computing Nash equilibria.

One benefit of formally defining EU in terms of BNs is that now the problem of

computing EU can be naturally cast as a BN inference problem. (In Chapter 3 we discussed such a reduction in the context of AGGs.) By Definition 5.2.12, $\mathrm{EU}^{\sigma}[j]$ is the sum of a polynomial number of terms of the form $E^{\sigma}[u_D^{\tau}]$. We thus focus on computing one such $E^{\sigma}[u_D^{\tau}]$. This can be computed by applying a standard BN inference algorithm on the induced BN. In fact, BN inference is the standard approach for computing expected utility in MAIDs [Koller and Milch, 2003]. Thus the above approach for TAGGs is computationally equivalent to the standard approach for a natural MAID representation of the same game. In this section, we show that the induced BNs of TAGGs have special structure that can be exploited to speed up computation, and present an algorithm that exploits this structure.

5.3.1 Exploiting Causal Independence

The standard BN inference approach for computing EU does not take advantage of some kinds of TAGG structure. In particular, recall that in the induced network, each action count variable A^{τ}'s parents are all previous decisions that have A^{τ} in their action sets, implying large in-degrees for action variables. Considering for example the clique-tree algorithm, this means large clique sizes, which is problematic because running time scales exponentially in the largest clique size of the clique tree. However, the CPDs of these action count variables are structured counting functions. Such structure is an instance of *causal independence* in BNs [Heckerman and Breese, 1996]. It also corresponds to anonymity structure for static game representations like symmetric games and AGGs.

We can exploit this structure to speed up computation of expected utility in TAGGs. Our approach is a specialization of Heckerman and Breese's [1996] method for exploiting causal independence in BNs. At a high level, Heckerman and Breese's method transforms the original BN by creating new nodes that represent intermediate results, and re-wiring some of the arcs, resulting in an equivalent BN with small in-degree. They then apply conventional inference algorithms on the new BN. For example, given an action count variable A_k^{τ} with parents $\{D_1 \dots D_\ell\}$, create a node M_i for each $i \in \{1 \dots \ell - 1\}$, representing the count induced by $D_1 \dots D_i$. Then, instead of having $D_1 \dots D_\ell$ as parents of A_k^{τ}, its parents become D_ℓ and $M_{\ell-1}$, and each M_i's parents are D_i and M_{i-1}. The resulting graph would have in-degree at

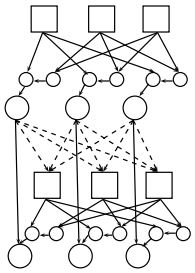

Figure 5.2: The transformed BN of the tollbooth game from Figure 5.1 with 3 lanes and 3 cars per time step.

most 2 for A_k^τ and the M_i's.

In our induced BN, the action count variables A_k^t at earlier time steps $t < \tau$ already represent some of these intermediate counts, so we do not need to duplicate them. Formally, we modify the original BN in the following way: for each action count variable A_k^τ, first remove the edges from its current parents. Instead, A_k^τ now has two parents: the action count variable $A_k^{\tau-1}$ and a new node $M_{A_k}^\tau$ representing the contribution of decisions at time τ to the count of A_k. If there is more than one decision at time τ that has A_k in its action set, we create intermediate variables as in Heckerman and Breese's method. We call the resulting BN the *transformed BN* of the TAGG. Figure 5.2 shows the transformed BN of the tollbooth game whose induced BN was given in Figure 5.1.

We can then use standard algorithms to compute probabilities $P(u_D^{t'})$ on the transformed BN. For classes of BNs with bounded treewidths, these probabilities (and thus $E[u_D^{t'}]$) can be computed in polynomial time.

5.3.2 Exploiting Temporal Structure

In practice, the standard inference approaches use heuristics to find an elimination ordering. This might not be optimal for our BNs. We present an algorithm based on the idea of eliminating variables in the temporal order. For the rest of the section, we fix D and a time $t' \in \mathrm{pt}(D)$ and consider the computation of $E^\sigma[u_D^{t'}]$.

We first group the variables of the induced network by time steps: variables at time τ include decisions at τ, action count variables A^τ, chance variables X with instantiation time τ, intermediate nodes between decisions and action counts at τ,

and utility variables U_A^τ. As we are only concerned about $E^\sigma[u_D^{t'}]$ for a $t' \in \text{pt}(D)$, we can safely discard the variables after time t', as well as utility variables before t'. It is straightforward to verify that the actual parents of variables at time τ are either at τ or before τ.

We say a network satisfies the *Markov property* if the actual parents of variables at time τ are either at τ or at $\tau - 1$. Parts of the induced BN (e.g. the action count variables) already satisfy the Markov property, but in general the network does not satisfy the property. Exceptions include chance variable parents and decision parents from more than one time step ago.

Given an induced BN, we can transform it into an equivalent network satisfying the Markov property. If a variable V_1 at t_1 is a parent of variable V_2 at t_2, with $t_2 - t_1 > 1$, then for each $t_1 < \tau < t_2$ we create a dummy variable V_1^τ belonging to time τ so that we copy the value of V_1 to $V_1^{t_2-1}$. We then delete the edge from V_1 to V_2 and add an edge from $V_1^{t_2-1}$ to V_2.

The Markov property is computationally desirable because variables in time τ d-separate past variables from future variables. A straightforward approach to exploiting the Markov property is the following: as τ goes from 1 to t', compute the joint distribution over variables at τ using the joint distribution over variables at $\tau - 1$.

In fact, we can do better by adapting the *interface algorithm* [Darwiche, 2001] for dynamic Bayesian networks to our setting.[3] Define the *interface* $\mathbf{I}^\tau$ to be the set of variables in time τ that have children in time $\tau + 1$. $\mathbf{I}^\tau$ d-separates *past* from *future*, where *past* is all variables before τ and non-interface variables in τ, and *future* is all variables after τ.

In an induced BN, $\mathbf{I}^\tau$ consists of: action count variables at time τ; chance variables X at time τ that have children in *future*; decisions at τ that are observed by *future* decisions; decision D which is a parent of $u_D^{t'}$, and dummy variables created by the transform.

We define the set of *effective variables* at time τ, denoted by $\mathbf{V}^\tau$, as the subset

[3] Whereas in DBNs the set of variables for each time step remains the same, for our setting this is no longer the case. It turns out that the interface algorithm can be adapted to work on our transformed BNs. Also, the transformed BNs of TAGGs have more structure than DBNs, particularly within the same time step, which we exploit for further computational speedup.

of $\mathbf{I}^\tau$ that are ancestors of $u_D^{t'}$. For time t', we let $\mathbf{V}^{t'} = \{u_D^{t'}\}$. Intuitively, at each time step τ we only need to keep track of the distribution $P(\mathbf{V}^\tau)$, which acts as a sufficient statistic as we go forward in time. For each τ, we calculate $P(\mathbf{V}^\tau)$ by conditioning on instantiations of $P(\mathbf{V}^{\tau-1})$. The interface algorithm for TAGGs can be summarized as the following:

1. compute distribution $P(\mathbf{V}^0)$
2. for $\tau = 1$ to t'
   (a) for each instantiation of $\mathbf{V}^{\tau-1}$, $v_j^{\tau-1}$, compute the distribution over $\mathbf{V}^\tau$: $P\left(\mathbf{V}^\tau | \mathbf{V}^{\tau-1} = v_j^{\tau-1}\right)$
   (b) $P(\mathbf{V}^\tau) = \sum_v P\left(\mathbf{V}^\tau | \mathbf{V}^{\tau-1} = v\right) P\left(\mathbf{V}^{\tau-1} = v\right)$
3. since $\mathbf{V}^{t'} = \{u_D^{t'}\}$, we now have $P(u_D^{t'})$
4. return the expected value $E[u_D^{t'}]$

We can further improve on this, in particular on the subtask of computing $P(\mathbf{V}^\tau | \mathbf{V}^{\tau-1})$. We observe that there is also a temporal order among variables in each time τ: first the decisions and intermediate variables, then action count variables, and finally chance variables. Partition $\mathbf{V}^\tau$ into four subsets consisting of action count variables $\mathbf{A}^\tau$, chance variables $\mathbf{X}^\tau$, behavior strategy variables $\mathbf{D}^\tau$ and dummy copy variables $\mathbf{C}^\tau$. Then $P(\mathbf{V}^\tau | \mathbf{V}^{\tau-1})$ can be factored into

$$P(\mathbf{C}^\tau | \mathbf{V}^{\tau-1}) P(\mathbf{D}^\tau, \mathbf{A}^\tau | \mathbf{V}^{\tau-1}) P(\mathbf{X}^\tau | \mathbf{A}^\tau, \mathbf{V}^{\tau-1}).$$

This allows us to first focus on decisions and action count variables to compute $P(\mathbf{D}^\tau, \mathbf{A}^\tau | \mathbf{V}^{\tau-1})$ and then carry out inference on the chance variables.

Calculating $P(\mathbf{D}^\tau, \mathbf{A}^\tau | \mathbf{V}^{\tau-1})$ involves eliminating all behavior strategy variables not in $\mathbf{D}^\tau$ as well as the intermediate variables. Note that conditioned on $\mathbf{V}^{\tau-1}$, all decisions at time τ are independent. This allows us to efficiently eliminate variables along the chains of intermediate variables. Let the decisions at time τ be $\{D_1^\tau, \ldots, D_\ell^\tau\}$. Let $\mathbf{M}^\tau$ be the set of intermediate variables corresponding to action count variables in $\mathbf{A}^\tau$. Let $\mathbf{M}_k^\tau$ be the subset of $\mathbf{M}^\tau$ that summarizes the contribution of $D_1^\tau, \ldots, D_k^\tau$. We eliminate variables in the order $D_1^\tau, D_2^\tau, \mathbf{M}_2^\tau, D_3^\tau, \mathbf{M}_3^\tau, \ldots, \mathbf{M}_\ell^\tau$, except for decisions in $\mathbf{D}^\tau$. The tables in the variable elimination algorithm need

to keep track of at most $|\mathbf{D}^\tau| + |\mathbf{A}^\tau|$ variables. Thus the complexity of computing $P(\mathbf{D}^\tau, \mathbf{A}^\tau | \mathbf{V}^{\tau-1})$ for an instantiation of $\mathbf{V}^{\tau-1}$ is exponential only in $|\mathbf{D}^\tau| + |\mathbf{A}^\tau|$.

Computing $P(\mathbf{X}^\tau | \mathbf{A}^\tau, \mathbf{V}^{\tau-1})$ for each instantiation of $\mathbf{A}^\tau, \mathbf{V}^{\tau-1}$ involves eliminating the chance variables not in $\mathbf{X}^\tau$. Any standard inference algorithm can be applied here. The complexity is exponential in the treewidth of the induced BN restricted on all chance variables at time τ, which we denote by G^τ.

Putting everything together, the bottleneck of our algorithm is constructing the tables for the joint distributions on $\mathbf{V}^\tau$, as well as doing inference on G^τ.

Theorem 5.3.1. *Given a TAGG and behavior strategy profile σ, if for all τ, both $|\mathbf{V}^\tau|$ and the treewidth of G^τ are bounded by a constant, then for any player j the expected utility $EU^\sigma[j]$ can be computed in time polynomial in the size of the TAGG representation and the size of σ.*

Our algorithm is especially effective for induced networks that are close to having the Markov property, in which case we only add a small number of dummy copy variables to $\mathbf{V}^\tau$. If only a constant number of dummy copy variables are added, the time complexity of computing expected utility then grows linearly in the duration of the game. On the other hand, for induced networks far from having the Markov property, $|\mathbf{V}^\tau|$ can grow linearly as τ increases, implying that the time complexity is exponential.

5.3.3 Exploiting Context-Specific Independence

TAGGs have action-specific utility functions, which allows them to express context-specific payoff independence: which utility function is used depends on which action is chosen at the decision. This is translated to context-specific independence structure in the induced BN, specifically in the CPD of u_D^τ. Conditioned on the value of D, u_D^τ only depends on one of its utility variable parents.

There are several ways of exploiting such structure computationally, including conditioning on the value of the decision D [Boutilier et al., 1996], or exploiting the context-specific independence in a variable elimination algorithm [Poole and Zhang, 2003]. One particularly simple approach that works for multiplexer utility nodes is to decompose the utility into a sum of utilities [Pfeffer, 2000]. For each utility node parent U_k^t of u_D^t, there is a utility function $u_{D,k}^t$ that depends on U_k^t

and D. If $D = k$, $u^t_{D,k}$ is equal to U^t_k. Otherwise, $u^t_{D,k}$ is 0. It is easy to see that $u^t_D(U^t_1,\ldots,U^t_m,D) = \sum_{k=1}^m u^t_{D,k}(U^t_k,D)$. We can then modify our algorithm to compute each $E[u^t_{D,k}]$ instead of $E[u^t_D]$. This results in a reduction in the set of effective variables $\mathbf{V}^\tau_k$, which are now the variables at τ that are ancestors of $u^t_{D,k}$. Furthermore, whenever $\mathbf{V}^\tau_k = \mathbf{V}^\tau_{k'}$ for some k,k', the distributions over them are identical and thus can be reused. For static games represented as TAGGs with $T = 1$, our algorithm is equivalent to the polynomial-time expected utility algorithm for AGGs described in Chapter 3.

Applying our algorithm to tollbooth games of Example 5.1.1 and ice cream games of Example 5.2.6, we observe that for both cases $\mathbf{V}^\tau$ consists of a subset of action count variables at τ plus the decision whose utility we are computing. Therefore the expected utilities of these games can be computed in polynomial time if $|\mathscr{A}|$ is bounded by a constant.

5.4 Computing Nash Equilibria

Since the induced MAID of a TAGG is payoff equivalent to the TAGG, algorithms for computing the Nash equilibria of MAIDs [Blum et al., 2006, Koller and Milch, 2003, Milch and Koller, 2008] can be directly applied to an induced MAID to find Nash equilibria of a TAGG. However, this approach does not exploit all TAGG structure. We can do better by constructing a transformed MAID, in a manner similar to the transformed BN, exploiting causal independence and CSI as in Sections 5.3.1 and 5.3.3.

We can do better yet and exploit the temporal structure as described in Section 5.3.2, if we use a solution algorithm that requires computation of probabilities and expected utilities. Govindan and Wilson [2002] presented an algorithm for computing equilibria in perfect-recall extensive-form games. Blum, Shelton and Koller [2006] adapted this algorithm to MAIDs. A key step in the algorithm is, for each pair of players i and j, and one of i's utility nodes, computing the marginal distribution over i's decisions and their parents, j's decisions and their parents, and the utility node. Our algorithm in Section 5.3.2 can be straightforwardly adapted to compute this distribution. This approach is efficient if each player only has a small number of decisions, as in the games in Examples 5.1.1 and 5.2.6.

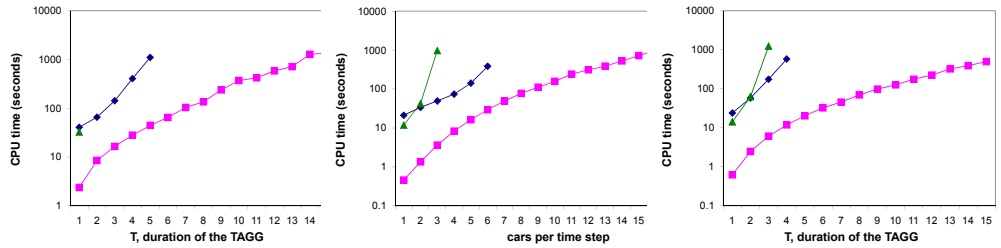

Figure 5.3: Running times for expected utility computation. Triangle data points represent Approach 1 (induced BN), diamonds represent Approach 2 (transformed BN), squares represent Approach 3 (proposed algorithm).

However, we did not implement these algorithms for TAGGs, because of a lack of publicly-available implementations for these algorithms. In particular, whereas Gametracer [Blum et al., 2002] provided an implementation of Govindan and Wilson's [2003] global Newton method for normal form games, it did not provide an implementation of Govindan and Wilson's [2002] algorithm for extensive-form games.

5.5 Experiments

We have implemented our algorithm for computing expected utility in TAGGs, and run experiments on the efficiency and scalability of our algorithm. We compared three approaches for computing expected utility given a TAGG:

Approach 1 applying the standard clique tree algorithm (as implemented by the Bayes Net Toolbox [Murphy, 2007]) on the induced BN;

Approach 2 applying the same clique tree algorithm on the transformed BN;

Approach 3 our proposed algorithm in Section 5.3.

All approaches were implemented in MATLAB. All our experiments were performed using a computer cluster consisting of machines with dual Intel Xeon 3.2GHz CPUs, 2MB cache and 2GB RAM.

We ran experiments on tollbooth game instances of varying sizes. For each game instance we measured the CPU times for computing expected utility of 100

random behavior strategy profiles. Figure 5.3 (left) shows the results in log scale for toll booth games with 3 lanes and 5 cars per time step, with the duration varying from 1 to 15. Approach 1 ran out of memory for games with more than 1 time step. Approach 2 was more scalable; but ran out of memory for games with more than 5 time steps. Approach 3 was the most scalable. On smaller instances it was faster than the other two approaches by an order of magnitude, and it did not run out of memory as we increased the size of the TAGGs to at least 20 time steps. For the toll booth game with 14 time steps it took 1279 seconds, which is approximately the time Approach 2 took for the game instance with 5 time steps. Figure 5.3 (middle) shows the results in log scale for tollbooth games with 3 time steps and 3 lanes, varying the number of cars per time step from 1 to 20. Approach 1 ran out of memory for games with more than 3 cars per time step; Approach 2 ran out of memory for games with more than 6 cars per time step; and again Approach 3 was the most scalable.

We also ran experiments on the ice cream games of Example 5.2.6. Figure 5.3 (right) shows the results in log scale for ice cream games with 4 locations, two vendors per time step, and durations varying from 1 to 15. The home locations for each vendor were generated randomly. Approaches 1 and 2 ran out of memory for games with more than 3 and 4 time steps, respectively. Approach 3 finished for games with 15 time steps in about the same time as Approach 2 took for games with 4 time steps.

5.6 Conclusions

TAGGs are a novel graphical representation of imperfect-information extensive-form games. They are an extension of simultaneous-move AGGs to the dynamic setting; and can be thought of as a sequence of AGGs played over T time steps, with action counts accumulating as time progresses. This process can be formally described by the induced BN. For situations with anonymity or CSI structure, the TAGG representation can be exponentially more compact than a direct MAID representation. We presented an algorithm for computing expected utility for TAGGs that exploits its anonymity, CSI as well as temporal structure. We showed both theoretically and empirically that our approach is significantly more efficient than

the standard approach on a direct MAID representation of the same game.

Another interesting solution concept is extensive-form correlated equilibrium [von Stengel and Forges, 2008]. EFCE was defined for perfect-recall extensive-form games, but the concept can be applied to other representations of perfect-recall dynamic games. One interesting direction is to adapt Huang and Von Stengel's [2008] polynomial-time algorithm for computing sample EFCE to compact representations like MAIDs and TAGGs.

As mentioned in Section 2.2.4, dynamic games with perfect recall have nice properties including the existence of Nash equilibria in behavior strategies. Furthermore, most of existing algorithmic approaches for dynamic games assume perfect recall. However, strategies in perfect-recall games can be computationally expensive to represent and reason about. For example in a perfect-recall TAGG, since each decision of a player has to condition on all previous decisions and observations of the player, the representation size of a behavior strategy grows exponentially in the number of previous decisions of that player. Representations like MAIDs and TAGGs can compactly express the utility functions, but this exponential blow-up of the strategy space is an inherent property of perfect recall. This blow-up already arises for two-player zero-sum games such as poker. Perfect recall is thus also problematic as a realistic model of rationality, since real-life agents do not have unlimited amount of memory. In light of this, an interesting direction is to explore imperfect-recall models, and solution concepts and algorithms for such models. In single-agent settings, there has been research on relaxing perfect recall using limited memory influence diagrams (LIMIDs) [Nilsson and Lauritzen, 2000]. However, for multi-agent imperfect recall games, existence of Nash equilibria in behavior strategies is not guaranteed. There has been some research on classes of imperfect recall games in which such equilibria do exist. One approach is based on "forgetting" certain "payoff-irrelevant" information from certain classes of perfect recall games, and showing that the resulting imperfect-recall game has a Nash equilibrium in behavior strategies that is also a Nash equilibrium of the original perfect recall game. Such equilibria are called Markov Perfect Equilibria (MPE) [e.g., Fudenberg and Tirole, 1991]. Milch and Koller [2008] took such an approach for MAIDs, in which case forgetting information corresponds to deleting certain edges into decision nodes. However, even if Nash equilibria in behavior strategies

exist in the resulting imperfect-recall game, there is currently no general-purpose algorithm for finding such equilibria. For the zero-sum game of poker, Waugh et al. [2009] considered the approach of formulating imperfect-recall models where players forget certain information. The reduction in strategy space allowed them to solve larger instances (corresponding to finer abstractions to the game of poker) than previously possible. They solved the resulting imperfect-recall game using *counterfactual regret minimization*, a heuristic algorithm without theoretical guarantees but appeared to empirically converge to approximate equilibria. Although unlike the MPE case, the transformation is not lossless (i.e., a Nash equilibrium of the imperfect-recall game is no longer a Nash equilibrium of the original game), they showed empirically that agents using the resulting strategies performed well. There have also been research on weaker solution concepts than MPE that allow players to ignore more information, such as Mean Field Equilibrium [e.g., Adlakha et al., 2010, Iyer et al., 2011].

Another approach is to consider restricted settings that admit stronger theoretical and practical properties. For instance, in Chapter 6 we consider Bayesian games, which (recall from Section 2.1.3) can be formulated as dynamic games; however they have specific structure that makes them computationally friendlier than arbitrary dynamic games. In particular, these games do not have the problem of exponential blow-up of strategy space. We are able to leverage techniques from simultaneous-move games for representing and computing with Bayesian games.

Chapter 6

Bayesian Action-Graph Games

6.1 Introduction

In this chapter we[1] consider static games of incomplete information (or Bayesian games) [Harsanyi, 1967], in which (recall from Section 2.1.3) players are uncertain about the underlying game. Bayesian games have found many applications in economics, including most notably auction theory and mechanism design.

Our interest is in computing with Bayesian games, and particularly in identifying sample Bayes-Nash equilibrium. We surveyed the relevant literature in Chapter 2, specifically Sections 2.1.3 and 2.2.3. To summarize, there are two key obstacles to performing such computations efficiently. The first is representational: recall that the straightforward tabular representation of Bayesian game utility functions (the Bayesian Normal Form) requires space exponential in the number of players. The second obstacle is the lack of existing algorithms for identifying sample Bayes-Nash equilibrium for arbitrary Bayesian games. Recall that a Bayesian game can be interpreted as an equivalent complete-information game via "induced normal form" or "agent form" interpretations. Thus one approach is to interpret a Bayesian game as a complete-information game, enabling the use of existing Nash-equilibrium-finding algorithms. However, generating the normal form representations under both of these complete-information interpretations causes an exponential blowup in representation size, even when the Bayesian game has only two players.

[1] This chapter is based on joint work with Kevin Leyton-Brown [2010].

In this chapter we propose Bayesian Action-Graph Games (BAGGs), a compact representation for Bayesian games. BAGGs can represent arbitrary Bayesian games, and furthermore can compactly express Bayesian games with commonly encountered types of structure. The type profile distribution is represented as a Bayesian network, which can exploit conditional independence structure among the types. BAGGs represent utility functions in a way similar to the AGG representation, and like AGGs, are able to exploit anonymity and action-specific utility independencies. Furthermore, BAGGs can compactly express Bayesian games exhibiting *type-specific independence*: each player's utility function can have different kinds of structure depending on her instantiated type. We provide an algorithm for computing expected utility in BAGGs, a key step in many algorithms for game-theoretic solution concepts. As in Chapter 5, our approach interprets expected utility computation as a probabilistic inference problem on an *induced Bayesian Network*. In particular, our algorithm runs in polynomial time for the important case of independent type distributions.

To compute Bayes-Nash equilibria for BAGGs, we consider the agent form interpretation of the BAGG. Howson and Rosenthal [1974] showed that the agent form of an arbitrary two-player Bayesian game is a polymatrix game, which can be represented compactly (thus avoiding the aforementioned blowup) and solved using a variant of the Lemke-Howson algorithm. However, for n-player BAGGs the corresponding agent forms do not correspond to polymatrix games or any other known representation, and the Lemke-Howson algorithm cannot be applied. Nevertheless, we are able to generalize Howson and Rosenthal's approach to propose an algorithm for finding sample Bayes-Nash equilibria for arbitrary BAGGs. Specifically, we show that BAGGs can act as a general compact representation of the agent form; in particular, computational tasks on the agent form can be done efficiently by leveraging our expected utility algorithm for BAGGs. We then apply black-box approaches for Nash equilibria in complete-information games discussed in Sections 2.2.2 and 3.4, specifically the simplicial subdivision algorithm [van der Laan et al., 1987] and Govindan and Wilson's [2003] global Newton method. We show empirically that our approach outperforms the existing approaches of solving for Nash on the induced normal form or on the normal form representation of the agent form.

Bayesian games can be interpreted as dynamic games with a initial move by Nature; thus, also related is the literature on representations for dynamic games, including MAIDs and TAGGs. Compared to these representations for dynamic games, BAGGs focus explicitly on structure common to Bayesian games; in particular, only BAGGs can efficiently express type-specific utility structure. Also, by representing utility functions and type distributions as separate components, BAGGs can be more versatile. For example, one future direction made possible by this separation is to model Bayesian games without common type distributions. Another future direction is to answer computational questions that do not depend on the type distribution, such as computing ex-post equilibria. Furthermore, we will see that BAGGs enjoy nicer computational properties than arbitrary dynamic games. For example, BAGGs can be solved by adapting Govindan and Wilson's global Newton method [2003] (see Section 2.2.1) for static games; this is generally more practical than their related Nash equilibrium algorithm [2002] that directly works on dynamic games: while both approaches avoid the exponential blowup of transforming to the induced normal form, the global Newton method for dynamic games has to solve an additional quadratic program at each step of the homotopy.

A limitation of BAGGs is that it requires the types to be discrete. There has been some research on heuristic methods for finding Bayes-Nash equilibria for Bayesian games with continuous types, including Reeves and Wellman [2004]'s work on iterated best response for certain classes of auction games and Rabinovich et al. [2009]'s work on fictitious play. Developing general compact representations and efficient algorithms for Bayes-Nash equilibria for such games remain interesting open problems.

6.2 Preliminaries

The standard definition of a Bayesian game $(N, \{A_i\}_{i \in N}, \Theta, P, \{u_i\}_{i \in N})$ is given in Definition 2.1.3. The standard concepts of pure strategy s_i, mixed strategy σ_i, expected utility for Bayesian games, and Bayes-Nash equilibrium are introduced in Section 2.2.3. Recall from Section 2.1.3 that the space bottlenecks of representing a Bayesian game are the type distribution and the utility function. Representing them as tables, the Bayesian normal form requires $n \times \prod_{i=1}^{n}(|\Theta_i| \times |A_i|) + \prod_{i=1}^{n}|\Theta_i|$

numbers to specify.

We say a Bayesian game has *independent type distributions* if players' types are drawn independently, i.e. the type-profile distribution $P(\theta)$ is a product distribution: $P(\theta) = \prod_i P(\theta_i)$. In this case the distribution P can be represented compactly using $\sum_i |\Theta_i|$ numbers.

Given a permutation of players $\pi : N \to N$ and an action profile $a = (a_1, \ldots, a_n)$, let $a^\pi = (a_{\pi(1)}, \ldots, a_{\pi(n)})$. Similarly let $\theta^\pi = (\theta_{\pi(1)}, \ldots, \theta_{\pi(n)})$. We say the type distribution P is symmetric if $|\Theta_i| = |\Theta_j|$ for all $i, j \in N$, and if for all permutations $\pi : N \to N$, $P(\theta) = P(\theta^\pi)$. We say a Bayesian game has *symmetric utility functions* if $|A_i| = |A_j|$ and $|\Theta_i| = |\Theta_j|$ for all $i, j \in N$, and if for all permutations $\pi : N \to N$, we have $u_i(a, \theta) = u_{\pi(i)}(a^\pi, \theta^\pi)$ for all $i \in N$. A Bayesian game is symmetric if its type distribution and utility functions are symmetric. The utility functions of such a game range over at most $|\Theta_i||A_i|\binom{n-2+|\Theta_i||A_i|}{|\Theta_i||A_i|-1}$ unique utility values.

A Bayesian game exhibits *conditional utility independence* if each player i's utility depends on the action profile a and her own type θ_i, but does not depend on the other players' types. Then the utility function of each player i ranges over at most $|A||\Theta_i|$ unique utility values.

6.2.1 Complete-information interpretations

Harsanyi [Harsanyi, 1967] showed that any Bayesian game can be interpreted as one of two complete-information games, the Nash equilibria of each of which correspond to Bayes-Nash equilibria of the Bayesian game.

A Bayesian game can be converted to its *induced normal form*, which is a complete-information game with the same set of n players, in which each player's set of actions is her set of pure strategies in the Bayesian game. Each player's utility under an action profile is defined to be equal to the player's expected utility under the corresponding pure strategy profile in the Bayesian game.

Alternatively, a Bayesian game can be transformed to its *agent form*, where each type of each player in the Bayesian game is turned into one player in a complete-information game. Formally, given a Bayesian game $(N, \{A_i\}_{i \in N}, \Theta, P, \{u_i\}_{i \in N})$, we define its agent form as the complete-information game $(\tilde{N}, \{\tilde{A}_{j,\theta_j}\}_{(j,\theta_j) \in \tilde{N}}, \{\tilde{u}_{j,\theta_j}\}_{(j,\theta_j) \in \tilde{N}})$, where $\tilde{N}$ consists of $\sum_{j \in N} |\Theta_j|$ players, one for ev-

ery type of every player of the Bayesian game. We index the players by the tuple (j, θ_j) where $j \in N$ and $\theta_j \in \Theta_j$. For each player $(j, \theta_j) \in \tilde{N}$ of the agent form game, her action set $\tilde{A}_{(j,\theta_j)}$ is A_j, the action set of j in the Bayesian game. The set of action profiles is then $\tilde{A} = \prod_{j,\theta_j} A_{(j,\theta_j)}$. The utility function of player (j, θ_j) is $\tilde{u}_{j,\theta_j} : \tilde{A} \to \mathbb{R}$. For all $\tilde{a} \in \tilde{A}$, $\tilde{u}_{j,\theta_j}(\tilde{a})$ is equal to the expected utility of player j of the Bayesian game given type θ_j, under the pure strategy profile $s^{\tilde{a}}$, where for all i and all θ_i, $s_i^{\tilde{a}}(\theta_i) = \tilde{a}_{(i,\theta_i)}$. Observe that there is a one-to-one correspondence between action profiles in the agent form and pure strategies of the Bayesian game. A similar correspondence exists for mixed strategy profiles: each mixed strategy profile σ of the Bayesian game corresponds to a mixed strategy $\tilde{\sigma}$ of the agent form, with $\tilde{\sigma}_{(i,\theta_i)}(a_i) = \sigma_i(a_i|\theta_i)$ for all i, θ_i, a_i. It is straightforward to verify that $\tilde{u}_{i,\theta_i}(\tilde{\sigma}) = u_i(\sigma|\theta_i)$ for all i, θ_i. This implies a correspondence between Bayes Nash equilibria of a Bayesian game and Nash equilibria of its agent form.

Proposition 6.2.1. *σ is a Bayes-Nash equilibrium of a Bayesian game if and only if $\tilde{\sigma}$ is a Nash equilibrium of its agent form.*

6.3 Bayesian Action-Graph Games

In this section we introduce Bayesian Action-Graph Games (BAGGs), a compact representation of Bayesian games. First consider representing the type distributions. Specifically, the type distribution P is specified by a Bayesian network (BN) containing at least n random variables corresponding to the n players' types $\theta_1, \ldots, \theta_n$. For example, when the types are independently distributed, then P can be specified by the simple BN with n variables $\theta_1, \ldots, \theta_n$ and no edges.

Now consider representing the utility functions. Our approach is to adapt concepts from the AGG representation (see Chapter 3) to the Bayesian game setting. At a high level, a BAGG is a Bayesian game on an *action graph*, a directed graph on a set of *action nodes* $\mathscr{A}$. To play the game, each player i, given her type θ_i, simultaneously chooses an action node from her *type-action set* $A_{i,\theta_i} \subseteq \mathscr{A}$. Each action node thus corresponds to an action choice that is available to one or more of the players. Once the players have made their choices, an *action count* is tallied for each action node $\alpha \in \mathscr{A}$, which is the number of agents that have chosen α. A player's utility depends only on the action node she chose and the action counts

on the neighbors of the chosen node. We observe that the main difference between the AGG and BAGG representations is that whereas in an AGG each player's set of available actions is specified by her action set, in a BAGG we have type-action sets, meaning each player's set of available actions can depend on her instantiated type.

We now turn to a formal description of BAGGs' utility function representation. Central to our model is the *action graph*.[2] An *action graph* $G = (\mathscr{A}, E)$ is a directed graph where $\mathscr{A}$ is the set of action nodes, and E is a set of directed edges, with self edges allowed. We say α' is a *neighbor* of α if there is an edge from α' to α, i.e., if $(\alpha', \alpha) \in E$. Let the *neighborhood* of α, denoted $\nu(\alpha)$, be the set of neighbors of α.

For each player i and each instantiation of her type $\theta_i \in \Theta_i$, her *type-action set* $A_{i,\theta_i} \subseteq \mathscr{A}$ is the set of possible action choices of i given θ_i. These subsets are unrestricted: different type-action sets may (partially or completely) overlap. Define player i's *total action set* to be $A_i^{\cup} = \bigcup_{\theta_i \in \Theta_i} A_{i,\theta_i}$. We denote by $A = \prod_i A_i^{\cup}$ the set of *action profiles*, and by $a \in A$ an action profile. Observe that the action profile a provides sufficient information about the type profile to be able to determine the outcome of the game; there is no need to additionally encode the realized type distribution. We note that for different types $\theta_i, \theta_i' \in \Theta_i$, A_{i,θ_i} and $A_{i,\theta_i'}$ may have different sizes; i.e., i may have different numbers of available action choices depending on her realized type.

A *configuration* c is a vector of $|\mathscr{A}|$ non-negative integers, specifying for each action node the numbers of players choosing that action. Let $c(\alpha)$ be the element of c corresponding to the action α. Let $\mathscr{C} : A \mapsto C$ be the function that maps from an action profile a to the corresponding configuration c. Formally, if $c = \mathscr{C}(a)$ then $c(\alpha) = |\{i \in N : a_i = \alpha\}|$ for all $\alpha \in \mathscr{A}$. Define $C = \{c : \exists a \in A \text{ such that } c = \mathscr{C}(a)\}$. In other words, C is the set of all possible configurations in the BAGG. Observe that the concept of configurations in BAGGs is related to the concept of configurations in AGGs in the following way: C in a BAGG is isomorphic to the set of configurations in an AGG-$\emptyset$ with the same action graph $G = (\mathscr{A}, E)$ but with action sets corresponding to total action sets of the BAGG, i.e., $A_i \equiv A_i^{\cup}$.

[2]The definition of action graph coincides with the corresponding concept in AGGs. We repeat the definition here in order to give a complete description of BAGGs.

We can also define a configuration over a subset of nodes. In particular, we will be interested in configurations over a node's neighborhood. Given a configuration $c \in C$ and a node $\alpha \in \mathscr{A}$, let the *configuration over the neighborhood* of α, denoted $c^{(\alpha)}$, be the restriction of c to $\nu(\alpha)$, i.e., $c^{(\alpha)} = (c(\alpha'))_{\alpha' \in \nu(\alpha)}$. Similarly, let $C^{(\alpha)}$ denote the set of configurations over $\nu(\alpha)$ in which at least one player plays α. Let $\mathscr{C}^{(\alpha)} : A \mapsto C^{(\alpha)}$ be the function that maps from an action profile to the corresponding configuration over $\nu(\alpha)$.

Definition 6.3.1. *A Bayesian action-graph game (BAGG) is a tuple* $(N, \Theta, P, \{A_{i,\theta_i}\}_{i \in N, \theta_i \in \Theta_i}, G, \{u^\alpha\}_{\alpha \in \mathscr{A}})$ *where N is the set of agents;* $\Theta = \prod_i \Theta_i$ *is the set of type profiles; P is the type distribution, represented as a Bayesian network;* $A_{i,\theta_i} \subseteq \mathscr{A}$ *is the type-action set of i given* θ_i*;* $G = (\mathscr{A}, E)$ *is the action graph; and for each* $\alpha \in \mathscr{A}$*, the utility function is* $u^\alpha : C^{(\alpha)} \to \mathbb{R}$.

As in the case of AGGs, shared actions in a BAGG capture the game's *anonymity* structure. Furthermore, the (lack of) edges between nodes in the action graph of a BAGG expresses *action- and type-specific independencies* of utilities of the game: depending on player i's chosen action node (which also encodes information about her type), her utility depends on configurations over different sets of nodes.

Lemma 6.3.2. *An arbitrary Bayesian game given in Bayesian normal form can be encoded as a BAGG storing the same number of utility values.*

Proof. Given an arbitrary Bayesian game $(N, \{A_i\}_{i \in N}, \Theta, P, \{u_i\}_{i \in N})$ represented in Bayesian normal form, we construct the BAGG $(N, \Theta, P, \{A'_{i,\theta_i}\}_{i \in N, \theta_i \in \Theta_i}, G, \{u^\alpha\}_{\alpha \in \mathscr{A}})$ as follows. The Bayesian normal form's tabular representation of type profile distribution P can be straightforwardly represented as a BN, e.g. by creating a random variable representing θ as the only parent of the random variables $\theta_1, \ldots, \theta_n$. To represent utility functions, we create an action graph G with $\sum_i |\Theta_i||A_i|$ action nodes; in other words, all type-action sets A'_{i,θ_i} are disjoint. Each action $a_i \in A_i$ of the Bayesian normal form corresponds to $|\Theta_i|$ action nodes in the BAGG, one for each type instantiation θ_i. For each player i and each type $\theta_i \in \Theta_i$, each action node $\alpha \in A'_{i,\theta_i}$ has incoming edges from all action nodes from type-action sets A'_{j,θ_j} for all $j \neq i$, $\theta_j \in \Theta_j$, i.e. all action nodes of the other players. For each action node $\alpha \in A'_{i,\theta_i}$ corresponding to $a_i \in A_i$, the utility function u^α is defined as follows: given configuration $c^{(\alpha)}$ we can infer the action profile $a'_{-i} \in A'_{-i}$

of the BAGG, which then tells us the corresponding a_{-i} and θ_{-i} of the Bayesian normal form, which gives us the utility $u_i(a,\theta)$. The number of utility values stored in this BAGG is the same as the Bayesian normal form. □

Bayesian games with symmetric utility functions exhibit anonymity structure, which can be expressed in BAGGs by sharing action nodes. Specifically, we label each Θ_i as $\{1,\ldots,T\}$, so that each $t \in \{1,\ldots,T\}$ corresponds to a class of equivalent types. Then for each $t \in \{1,\ldots,T\}$, we have $A_{i,t} = A_{j,t}$ for all $i,j \in N$, i.e. type-action sets for equivalent types are identical. Figure 6.1 shows the action graph for a symmetric Bayesian game with two types and two actions per type.

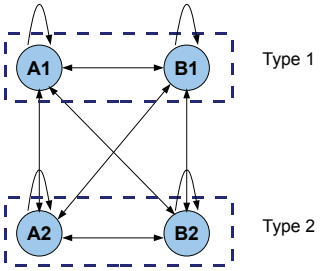

Figure 6.1: Action graph for a symmetric Bayesian game with n players, 2 types, 2 actions per type.

6.3.1 BAGGs with Function Nodes

In this section we extend the basic BAGG representation by introducing *function nodes* to the action graph, as we did for AGG-FNs in Chapter 3. Function nodes allow us to exploit a much wider variety of utility structures in BAGGs.

In this extended representation,[3] the action graph G's vertices consist of both the set of action nodes $\mathscr{A}$ and the set of function nodes $\mathscr{P}$. We require that no function node $p \in \mathscr{P}$ can be in any player's action set. Each function node $p \in \mathscr{P}$ is associated with a function $f^p : C^{(p)} \to \mathbb{R}$. We extend c by defining $c(p)$ to be the result of applying f^p to the configuration over p's neighbors, $f^p(c^{(p)})$. Intuitively, $c(p)$ can be used to describe intermediate parameters that players' utilities depend

[3]The definitions of function nodes and contribution-independent function nodes coincides with the corresponding concepts in AGGs. We repeat them here for completeness.

on. To ensure that the BAGG is meaningful, the graph restricted to nodes in $\mathscr{P}$ is required to be a directed acyclic graph. As before, for each action node α we define a utility function $u^\alpha : C^{(\alpha)} \to \mathbb{R}$.

Of particular computational interest is the subclass of *contribution-independent function nodes*. A function node p in a BAGG is *contribution-independent* if $\nu(p) \subseteq \mathscr{A}$, there exists a commutative and associative operator $*$, and for each $\alpha \in \nu(p)$ an integer w_α, such that given an action profile $a = (a_1, \ldots, a_n)$, $c(p) = *_{i \in N : a_i \in \nu(p)} w_{a_i}$. A BAGG is contribution-independent if all its function nodes are contribution-independent. Intuitively, if function node p is contribution-independent, each player's strategy affects $c(p)$ independently.

A very useful kind of contribution-independent function nodes are *simple aggregator function nodes*, which set $*$ to the summation operator $+$ and the weights to 1. Such a function node p simply counts the number of players that chose any action in $\nu(p)$.

Let us consider the size of a BAGG representation. The representation size of the Bayesian network for P is exponential only in the in-degree of the BN. The utility functions store $\sum_\alpha |C^{(\alpha)}|$ values. Recall that C and thus $C^{(\alpha)}$ correspond to configurations in an related AGG. We can thus apply the same analysis for the representation size of AGGs in Chapter 3. As in Chapter 3, estimations of this size generally depend on what types of function nodes are included. We state only the following (relatively straightforward) result since in this chapter we are mostly concerned with BAGGs with simple aggregator function nodes.

Theorem 6.3.3. *Consider BAGGs whose only function nodes, if any, are simple aggregator function nodes. If the in-degrees of the action nodes as well as the in-degrees of the Bayesian networks for P are bounded by a constant, then the sizes of the BAGGs are bounded by a polynomial in n, $|\mathscr{A}|$, $|\mathscr{P}|$, $\sum_i |\Theta_i|$ and the sizes of domains of variables in the BN.*

The proof is by a direct application of Corollary 3.2.11. This theorem shows a nice property of simple aggregator function nodes: representation size does not grow exponentially in the in-degrees of these function nodes. The next example (an extension of Example 3.2.7) illustrates the usefulness of simple aggregator function nodes, including for expressing conditional utility independence.

Example 6.3.4 (Coffee Shop game)**.** *Consider a symmetric Bayesian game involving n players; each player plans to open a new coffee shop in a downtown area, but has to decide on the location. The downtown area is represented by a $r \times k$ grid. Each player can choose to open a shop located within any of the $B \equiv rk$ blocks or decide not to enter the market. Each player has one of T types, representing her private information about her cost of opening a coffee shop. Players' types are independently distributed. Conditioned on player i choosing some location, her utility depends on: (a) her own type; (b) the number of players that chose the same block; (c) the number of players that chose any of the surrounding blocks; and (d) the number of players that chose any other location.*

The Bayesian normal form representation of this game has size $n[T(B+1)]^n$. The game can be expressed as a BAGG as follows. Since the game is symmetric, we label the types as $\{1,\ldots,T\}$. $\mathscr{A}$ contains one action O corresponding to not entering and TB other action nodes, with each location corresponding to a set of T action nodes, each representing the choice of that location by a player with a different type. For each $t \in \{1,\ldots,T\}$, the type-action sets $A_{i,t} = A_{j,t}$ for all $i, j \in N$ and each consists of the action O and B actions corresponding to locations for type t. For each location (x,y) we create three function nodes: p_{xy} representing the number of players choosing this location, p'_{xy} representing the number of players choosing any surrounding blocks, and p''_{xy} representing the number of players choosing any other block. Each of these function nodes is a simple aggregator function node, whose neighbors are action nodes corresponding to the appropriate locations (for all types). Each action node for location (x,y) has three neighbors, p_{xy}, p'_{xy}, and p''_{xy}. Figure 6.2 shows the action graph for the game with $T = 2$ on an $1 \times k$ grid. Since the BAGG action graph has maximum in-degree 3, by Theorem 6.3.3 the representation size is polynomial in n, B and T.

6.4 Computing a Bayes-Nash Equilibrium

In this section we consider the problem of finding a sample Bayes-Nash equilibrium given a BAGG. Our overall approach is to interpret the Bayesian game as a complete-information game, and then to apply existing algorithms for finding Nash equilibria of complete-information games. We consider two state-of-the-art

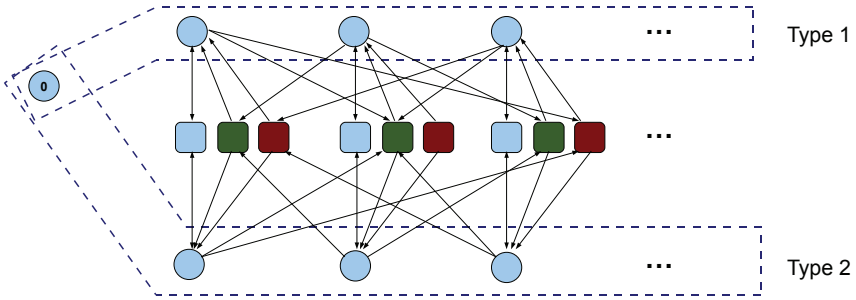

Figure 6.2: BAGG representation for a Coffee Shop game with 2 types per player on an $1 \times k$ grid.

Nash equilibrium algorithms, van der Laan et al's simplicial subdivision [1987] and Govindan and Wilson's global Newton method [2003].

Recall from Section 6.2.1 that a Bayesian game can be transformed into its induced normal form or its agent form. In the induced normal form, each player i has $|A_i|^{|\Theta_i|}$ actions (corresponding to her pure strategies of the Bayesian game). Solving such a game would be infeasible for large $|\Theta_i|$; just to represent an Nash equilibrium requires space exponential in $|\Theta_i|$.

A more promising approach is to consider the agent form. Note that we can straightforwardly adapt the agent-form transformation described in Section 6.2.1 to the setting of BAGGs: now the action set of player (i, θ_i) of the agent form corresponds to the type-action set A_{i,θ_i} of the BAGG. The resulting complete-information game has $\sum_{i \in N} |\Theta_i|$ players and $|A_{i,\theta_i}|$ actions for each player (i, θ_i); a Nash equilibrium can be represented using just $\sum_i \sum_{\theta_i} |A_{i,\theta_i}|$ numbers. However, the normal form representation of the agent form has size $\sum_{j \in N} |\Theta_j| \prod_{i,\theta_i} |A_{i,\theta_i}|$, which grows exponentially in n and $|\Theta_i|$. Applying the Nash equilibrium algorithms to this normal form would be infeasible for large games. Fortunately, we do not have to explicitly represent the agent form as a normal form game. Instead, we treat a BAGG as a compact representation of its agent form, and carry out any required computation on the agent form by operating directly on the BAGG. Recall from

Section 2.2.1 that a key computational task required by both Nash equilibrium algorithms in their inner loops is the computation of expected utility of the agent form. Recall from Section 6.2.1 that for all (i, θ_i) the expected utility $\tilde{u}_{i,\theta_i}(\tilde{\sigma})$ of the agent form is equal to the expected utility $u_i(\sigma|\theta_i)$ of the Bayesian game. Thus in the remainder of this section we focus on the problem of computing expected utility in BAGGs.

6.4.1 Computing Expected Utility in BAGGs

Recall from Section 2.2.3 that $\sigma^{\theta_i \to a_i}$ is the mixed strategy profile that is identical to σ except that i plays a_i given θ_i. The main quantity we are interested in is $u_i(\sigma^{\theta_i \to a_i}|\theta_i)$, player i's expected utility given θ_i under the strategy profile $\sigma^{\theta_i \to a_i}$. Note that the expected utility $u_i(\sigma|\theta_i)$ can then be computed as the sum $u_i(\sigma|\theta_i) = \sum_{a_i} u_i(\sigma^{\theta_i \to a_i}|\theta_i)\sigma_i(a_i|\theta_i)$.

One approach is to directly apply Equation (2.2.2), which has $(|\Theta_{-i}| \times |A|)$ terms in the summation. For games represented in Bayesian normal form, this algorithm runs in time polynomial in the representation size. Since BAGGs can be exponentially more compact than their equivalent Bayesian normal form representations, this algorithm runs in exponential time for BAGGs.

In this section we present a more efficient algorithm that exploits BAGG structure. We first formulate the expected utility problem as a Bayesian network inference problem. Given a BAGG and a mixed strategy profile $\sigma^{\theta_i \to a_i}$, we construct the *induced Bayesian network (IBN)* as follows.

We start with the BN representing the type distribution P, which includes (at least) the random variables $\theta_1, \ldots, \theta_n$. The conditional probability distributions (CPDs) for the network are unchanged. We add the following random variables: one strategy variable D_j for each player j; one action count variable for each action node $\alpha \in \mathscr{A}$, representing its action count, denoted $c(\alpha)$; one function variable for each function node $p \in \mathscr{P}$, representing its configuration value, denoted $c(p)$; and one utility variable U^α for each action node α. We then add the following edges: an edge from θ_j to D_j for each player j; for each player j and each $\alpha \in A_j^{\cup}$, an edge from D_j to $c(\alpha)$; for each function variable $c(p)$, all incoming edges corresponding to those in the action graph G; and for each $\alpha \in \mathscr{A}$, for each action or function node

$m \in \nu(\alpha)$ in G, an edge from $c(m)$ to U^α in the IBN.

The CPDs of the newly added random variables are defined as follows. Each strategy variable D_j has domain $A_j^\cup$, and given its parent θ_j, its CPD chooses an action from $A_j^\cup$ according to the mixed strategy $\sigma_j^{\theta_i \to a_i}$. In other words, if $j \neq i$ then $\Pr(D_j = a_j|\theta_j)$ is equal to $\sigma_j(a_j|\theta_j)$ for all $a_j \in A_{j,\theta_j}$ and 0 for all $a_j \in A_j^\cup \setminus A_{j,\theta_j}$; and if $j = i$ we have $\Pr(D_j = a_i|\theta_j) = 1$. For each action node α, the parents of its action-count variable $c(\alpha)$ are strategy variables that have α in their domains. The CPD is a deterministic function that returns the number of its parents that take value α; i.e., it calculates the action count of α. For each function variable $c(p)$, its CPD is the deterministic function f^p. The CPD for each utility variable U^α is a deterministic function specified by u^α.

Remark 6.4.1. *Observe that our construction of IBN here is similar to the construction of induced BN from a TAGG in Chapter 5. One difference is that in a BAGG, type affects utility indirectly through type-action sets, resulting in a different construction of CPDs at the strategy variables D_j from the TAGG case. Also, each strategy variable in a BAGG has in-degree 1, whereas in a perfect-recall TAGG the in-degree of a decision of player i grows linearly in the number of i's previous decisions.*

It is straightforward to verify that the IBN is a directed acyclic graph (DAG) and thus represents a valid joint distribution. Furthermore, the expected utility $u_i(\sigma^{t_i \to a_i}|\theta_i)$ is exactly the expected value of the variable U^{a_i} conditioned on the instantiated type θ_i.

Lemma 6.4.2. *For all $i \in N$, all $\theta_i \in \Theta_i$ and all $a_i \in A_{i,\theta_i}$, we have $u_i(\sigma^{\theta_i \to a_i}|\theta_i) = E[U^{a_i}|\theta_i]$.*

Standard BN inference methods could be used to compute $E[U^{a_i}|\theta_i]$. However, such standard algorithms do not take advantage of structure that is inherent in BAGGs. In particular, recall that in the induced network, each action count variable $c(\alpha)$'s parents are all strategy variables that have α in their domains, implying large in-degrees for action count variables. As in the TAGG case, the CPDs of action count variables exhibit *causal independence*, and we can apply a version of Heckerman and Breese's method [Heckerman and Breese, 1996] to transform the IBN

into an equivalent BN small in-degree. Given an action count variable $c(\alpha)$ with parents (say) $\{D_1 \ldots D_n\}$, for each $i \in \{1 \ldots n-1\}$ we create a node $M_{\alpha,i}$, representing the count induced by $D_1 \ldots D_i$. Then, instead of having $D_1 \ldots D_n$ as parents of $c(\alpha)$, its parents become D_n and $M_{\alpha,n-1}$, and each $M_{\alpha,i}$'s parents are D_i and $M_{\alpha,i-1}$. The resulting graph has in-degree at most 2 for $c(\alpha)$ and the $M_{\alpha,i}$'s. The CPDs of function variables corresponding to contribution-independent function nodes also exhibit causal independence, and thus we can use a similar transformation to reduce their in-degree to 2. We call the resulting Bayesian network the *transformed Bayesian network (TBN)* of the BAGG.

As in Chapter 5, it is straightforward to verify that the representation size of the TBN is polynomial in the size of the BAGG. We can then use standard inference algorithms to compute $E[U^\alpha|\theta_i]$ on the TBN. For classes of BNs with bounded treewidths, this can be computed in polynomial time. Since the graph structure (and thus the treewidth) of the TBN does not depend on the strategy profile (but, rather, only on the BAGG itself), we have the following result.

Theorem 6.4.3. *For BAGGs whose TBNs have bounded treewidths, expected utility can be computed in time polynomial in n, $|\mathscr{A}|$, $|\mathscr{P}|$ and $|\sum_i \Theta_i|$.*

Bayesian games with independent type distributions are an important class of games and have many applications, such as independent-private-value auctions. When contribution-independent BAGGs have independent type distributions, expected utility can be efficiently computed.

Theorem 6.4.4. *For contribution-independent BAGGs with independent type distributions, expected utility can be computed in time polynomial in the size of the BAGG.*

Note that this result is stronger than that of Theorem 6.4.3, which only guarantees efficient computation when TBNs have constant treewidth.

Proof. We reduce the problem of computing expected utility $u_i(\sigma^{\theta_i \to a_i}|\theta_i)$ for BAGGs with independent type distributions to the problem of computing expected utility for AGGs.

Given a BAGG $(N, G, \{u^\alpha\}_{\alpha \in \mathscr{A}})$, we consider the AGG Γ specified by $(N, \{A_i^\cup\}_{i \in N}, G, \{u^\alpha\}_{\alpha \in \mathscr{A}})$, i.e., an AGG with the same set of players, the same action

graph and the same utility functions, but with action sets corresponding to total action sets of the BAGG. The representation size of the AGG Γ is proportional to the size of the BAGG. Furthermore, since the BAGG is contribution-independent, all function nodes in the AGG Γ are contribution-independent.

Given i, θ_i and $\sigma^{\theta_i \to a_i}$, for each player $j \neq i$ we can calculate $\Pr(D_j)$ by summing out θ_j: $\Pr(D_j = a_j) = \sum_{\theta_j} \sigma_j(a_j|\theta_j)$. Observe that this distribution of the strategy variable D_j can be interpreted as a (complete-information) mixed strategy σ'_j of the AGG Γ's player j. Similarly for player i, the distribution $\Pr(D_i|\theta_i)$ can be interpreted as a mixed strategy σ'_i for Γ's player i. Furthermore these distributions are independent, so they induce the same distribution over configurations of the BAGG as the distribution over configurations of the AGG Γ induced by the mixed-strategy profile $\sigma' = (\sigma'_1, \ldots, \sigma'_n)$.

Therefore the expected utility $u_i(\sigma^{\theta_i \to a_i}|\theta_i)$ for the BAGG is equal to the expected utility of i in the AGG Γ under the mixed strategy profile σ'. Expected utility for contribution-independent AGGs can be computed in polynomial time by running the algorithm described in Section 3.4.2. □

An alternative approach for proving Theorem 6.4.4 is to work on the TBN of the BAGG, which can be shown to have treewidth as most $|\nu(a_i)|$. Although $|\nu(a_i)|$ is not necessarily a constant, meaning that Theorem 8 cannot be directly applied, it can be shown that a variable elimination algorithm needs to store at most $|C^{(a_i)}|$ numbers in each of its tables, which is polynomial in the size of the BAGG. These two proof approaches can be thought of as two interpretations of the same expected utility algorithm.

6.5 Experiments

We have implemented our approach for computing a Bayes-Nash equilibrium given a BAGG by applying Nash equilibrium algorithms on the agent form of the BAGG. We adapted two algorithms, GAMBIT's [McKelvey et al., 2006] implementation of simplicial subdivision and GameTracer's [Blum et al., 2002] implementation of Govindan and Wilson's global Newton method, by replacing calls to expected utility computations of the complete-information game with corresponding expected utility computations of the BAGG. Recall from Section 3.5 that we have adapted

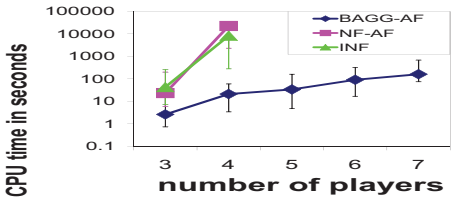

Figure 6.3: GW, varying players.

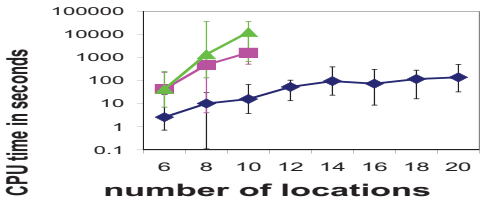

Figure 6.4: GW, varying locations.

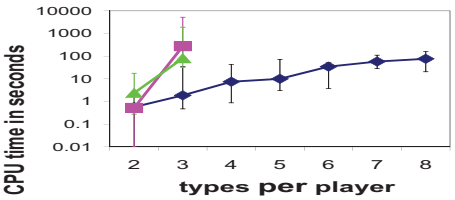

Figure 6.5: GW, varying types.

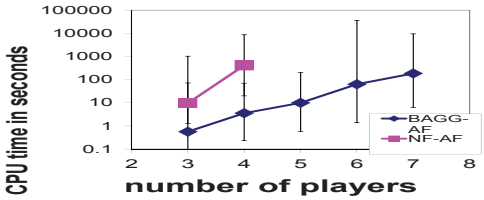

Figure 6.6: Simplicial subdivision.

GAMBIT's implementation of simplicial subdivision to a black-box implementation, and that Gametracer's implementation is already black-box, thus further adaptation of the algorithms to the BAGG case was relatively straightforward to implement once we have the expected utility subroutine. We ran experiments that tested the performance of our approach (denoted by BAGG-AF) against two approaches that compute a Bayes-Nash equilibrium for arbitrary Bayesian games. The first (denoted INF) computes a Nash equilibrium on the induced normal form; the second (denoted NF-AF) computes a Nash equilibrium on the normal form representation of the agent form. Both were implemented using the original, normal-form-based implementations of simplicial subdivision and global Newton method. We thus studied six concrete algorithms, two for each game representation.

We tested these algorithms on instances of the Coffee Shop Bayesian game described in Example 6.3.4. We created games of different sizes by varying the number of players, the number of types per player and the number of locations. For each size we generated 10 game instances with random integer payoffs, and measured the running (CPU) times. Each run was cut off after 10 hours if it had not yet finished. All our experiments were performed using a computer cluster consisting of 55 machines with dual Intel Xeon 3.2GHz CPUs, 2MB cache and 2GB RAM, running Suse Linux 11.1.

We first tested the three approaches based on the Govindan-Wilson (GW) algo-

rithm. Figure 6.3 shows running time results for Coffee Shop games with n players, 2 types per player on a 2×3 grid, with n varying from 3 to 7. Figure 6.4 shows running time results for Coffee Shop games with 3 players, 2 types per player on a $2 \times x$ grid, with x varying from 3 to 10. Figure 6.5 shows results for Coffee Shop games with 3 players, T types per player on a 1×3 grid, with T varying from 2 to 8. The data points represent the median running time of 10 game instances, with the error bars indicating the maximum and minimum running times. All results show that our BAGG-based approach (BAGG-AF) significantly outperformed the two normal-form-based approaches (INF and NF-AF). Furthermore, as we increased the dimensions of the games the normal-form based approaches quickly ran out of memory (hence the missing data points), whereas BAGG-NF did not.

We also did experiments on BAGG-AF and NF-AF running the simplicial subdivision algorithm. Figure 6.6 shows running time results for Coffee Shop games with n players, 2 types per player on a 1×3 grid, with n varying from 3 to 7. Again, BAGG-AF significantly outperformed NF-AF, and NF-AF ran out of memory for game instances with more than 4 players.

Chapter 7

Polynomial-time Computation of Exact Correlated Equilibrium in Compact Games

7.1 Introduction

So far we have focused on the AGG representation and its extensions. For the remaining two technical chapters (this chapter and Chapter 8) we switch our attention to algorithms that work for a wide class of compact representations including AGGs. Specifically, we consider problems regarding correlated equilibrium (CE) [Aumann, 1974, 1987]. In this chapter we consider the problem of computing a sample correlated equilibrium. In Section 2.2.7 we gave an overview of literature on this problem; in order to motivate our results in this chapter we first take a more in-depth look at some of the relevant papers. The "Ellipsoid Against Hope" algorithm [Papadimitriou, 2005, Papadimitriou and Roughgarden, 2008] is a polynomial-time method for identifying (a polynomial-size representation of) a CE, given a game representation satisfying two properties: polynomial type and the *polynomial expectation property*, which requires access to a polynomial-time algorithm that computes the expected utility of any player under any mixed-strategy profile. Recall that most existing compact game representations discussed in Sec-

tion 2.1.1 (including graphical games, symmetric games, congestion games, polymatrix games and action-graph games) satisfy these properties. At a high level, the Ellipsoid Against Hope algorithm works by solving an infeasible dual LP (D) using the ellipsoid method (exploiting the existence of a separation oracle), and arguing that the LP (D') formed by the generated cutting planes must also be infeasible. Solving the dual of this latter LP (which has polynomial size) yields a CE, which is represented as a mixture of the product distributions generated by the separation oracle. The Ellipsoid Against Hope algorithm is an instance of the black-box approach: it calls the expected utility subroutine as part of its separation oracle computation, but does not access the internal details of the representation.

7.1.1 Recent Uncertainty About the Complexity of Exact CE

In a recent paper, Stein, Parrilo and Ozdaglar [2010] raised two interrelated concerns about the Ellipsoid Against Hope algorithm. First, they identified a symmetric 3-player, 2-action game with rational[1] utilities on which the algorithm can fail to compute an exact CE. Indeed, they showed that the same problem arises on this game for a whole class of related algorithms. Specifically, if an algorithm (a) outputs a rational solution, (b) outputs a convex combination of product distributions, and (c) outputs a convex combination of symmetric product distributions when the game is symmetric, then that algorithm fails to find an exact CE on their game, because the only CE of their game that satisfies properties (b) and (c) has irrational probabilities. This implies that any algorithm for exact rational CE must violate (b) or (c).

Second, Stein, Parrilo and Ozdaglar also showed that the original analysis by Papadimitriou and Roughgarden [2008] incorrectly handles certain numerical precision issues, which we now briefly describe. Recall that a run of the ellipsoid method requires as inputs an initial bounding ball with radius R and a volume bound v such that the algorithm stops when the ellipsoid's volume is smaller than v. To correctly certify the (in)feasibility of an LP using the ellipsoid method, R and v need to be set to appropriate values, which depend on the maximum encoding size of a constraint in the LP. However (as pointed out by Papadimitriou and

[1] Throughout this chapter, by "rational" we mean rational numbers (ratios of integers) rather than rationality of players.

Roughgarden [2008]), each cut returned by the separation oracle is a convex combination of the constraints of the original dual LP (D) and thus may require more bits to represent than any of the constraints in (D); as a result, the infeasibility of the LP (D') formed by these cuts is not guaranteed. Papadimitriou and Roughgarden [2008] proposed a method to overcome this difficulty, but Stein et al. showed that this method is insufficient for finding an exact CE. For the related problem of finding an approximate correlated equilibrium (ε-CE), Stein et al. gave a slightly modified version of the Ellipsoid Against Hope algorithm that runs in time polynomial in $\log \frac{1}{\varepsilon}$ and the game representation size.[2] For problems that can have necessarily irrational solutions, it is typical to consider such approximations as efficient; however, the computation of a sample CE is not such a problem, as there always exists a rational CE in a game with rational utilities, since CE are defined by linear constraints. It remains an open problem to determine whether the Ellipsoid Against Hope algorithm can be modified to compute an exact, rational correlated equilibrium.[3]

7.1.2 Our Results

In this chapter, we use an alternate approach—completely sidestepping the issues just discussed—to derive a polynomial-time algorithm for computing an exact (and rational) correlated equilibrium given a game representation that has polynomial type and satisfies the polynomial expectation property. Specifically, our approach is based on the observation that if we use a separation oracle (for the same dual LP formulation proposed by Papadimitriou and Roughgarden [2008]) that generates cuts corresponding to pure-strategy profiles (instead of Papadimitriou and Roughgarden's separation oracle that generates nontrivial product distributions), then these cuts are actual constraints in the dual LP, as opposed to convex combinations of constraints. As a result we no longer encounter the numerical accuracy issues that prevented the previous approaches from finding exact correlated equilibria. Both the resulting algorithm and its analysis are also considerably simpler

[2] An ε-CE is defined to be a distribution that violates the CE incentive constraints by at most ε.

[3] In a recent addendum to their original paper, Papadimitriou and Roughgarden [2010] acknowledged the flaw in the original algorithm. We note also that Stein et al. subsequently withdrew their paper from arXiv. It is our belief that their results are nevertheless correct; we discuss them here because they help to motivate our alternate approach.

than the original: standard techniques from the theory of the ellipsoid method are sufficient to show that our algorithm computes an exact CE using a polynomial number of oracle queries.

The key issue is the identification of pure-strategy-profile cuts. It is relatively straightforward to show that such cuts always exist: since the product distribution generated by the Ellipsoid Against Hope algorithm ensures the nonnegativity of a certain expected value, then by a simple application of the probabilistic method there must exist a pure-strategy profile that also ensures the nonnegativity of that expected value. The key is to go beyond this nonconstructive proof of existence to also *compute* pure-strategy-profile cuts in polynomial time. We show how to do this by applying the method of conditional probabilities [Erdős and Selfridge, 1973, Raghavan, 1988, Spencer, 1994], an approach for derandomizing probabilistic proofs of existence. At a high level, our new separation oracle begins with the product distribution generated by Papadimitriou and Roughgarden's separation oracle, then sequentially fixes a pure strategy for each player in a way that guarantees that the corresponding conditional expectation given the choices so far remains nonnegative. Since our separation oracle goes though players sequentially, the cuts generated can be asymmetric even for symmetric games. Indeed, we can confirm (see Section 7.4.2) that it makes such asymmetric cuts on Stein, Parrilo and Ozdaglar's symmetric game—thus violating their condition (c)—because our algorithm always identifies a rational CE. As with the Ellipsoid Against Hope algorithm and Stein et al.'s modified algorithm, our algorithm is also a black-box algorithm that calls the expected utility subroutine.

Another effect of our use of pure-strategy-profile cuts is that the correlated equilibria generated by our algorithm are guaranteed to have polynomial-sized supports; i.e., they are mixtures over a polynomial number of pure strategy profiles. Correlated equilibria with polynomial-sized supports are known to exist in every game (e.g., [Germano and Lugosi, 2007]); intuitively this is because CE are defined by a polynomial number of linear constraints, so a basic feasible solution of the linear feasibility program would have a polynomial number of non-zero entries. Such small-support correlated equilibria are more natural solutions than the mixtures of product distributions produced by the Ellipsoid Against Hope algorithm: because of their simpler form they require fewer bits to represent and fewer random bits to

sample from; furthermore, verifying whether a given polynomial-support distribution is a CE only requires evaluating the utilities of a polynomial number of pure strategy profiles, whereas verifying whether a mixture of product distributions is a CE would require evaluating expected utilities under product distributions, which is generally more expensive. No tractable algorithm has previously been proposed for identifying such a CE, thus our algorithm is the first algorithm that computes in polynomial time a CE with polynomial support given a compactly-represented game. In fact, we show that any CE computed by our algorithm corresponds to a basic feasible solution of the linear feasibility program that defines CE, and is thus an extreme point of the set of CE of the game.

Since Papadimitriou and Roughgarden [2008] proposed the Ellipsoid Against Hope algorithm for computing a CE, researchers have proposed algorithms for related problems that used a similar approach (which we call the Ellipsoid Against Hope approach): first solving an infeasible LP using the ellipsoid method with some separation oracle, then arguing that the LP formed by the cutting planes is also infeasible, and finally solving the dual of the latter polynomial-sized LP. For example, Hart and Mansour [2010] considered the setting where each player initially knows only her own utility function, and proposed a communication procedure that finds a CE with polynomial communication complexity using a straightforward adaptation of the Ellipsoid Against Hope algorithm. Huang and Von Stengel [2008] proposed a polynomial-time algorithm for computing a extensive-form correlated equilibrium (EFCE) [von Stengel and Forges, 2008], a solution concept for extensive-form games, by applying the Ellipsoid Against Hope approach to the LP formulation of EFCE. For both algorithms, the separation oracle outputs a mixture of the original constraints, and hence the flaws of the Ellipsoid Against Hope algorithm pointed out by Stein et al. [2010] also apply. We show that our techniques can be adapted to these two algorithms, yielding in both cases exact solutions with polynomial-sized supports. In particular, we replace the original separation oracles with "purified" versions that output cutting planes corresponding to the original constraints, which ensures that the resulting algorithms avoid the numerical issues.

The rest of the chapter is organized as follows. We start with basic definitions and notation in Section 7.2. In Section 7.3 we summarize Papadimitriou and Roughgarden's Ellipsoid Against Hope algorithm. In Section 7.4 we describe our

algorithm and prove its correctness. In Sections 7.5 and 7.6 we describe our fixes to Hart and Mansour's [2010] and Huang and Von Stengel's [2008] algorithms respectively, and Section 7.7 concludes.

This chapter is based on published joint work with Kevin Leyton-Brown [2011]. New material that does not appear in [Jiang and Leyton-Brown, 2011] includes Sections 7.5 and 7.6.

7.2 Preliminaries

In this chapter and Chapter 8 we largely follow the notation of Papadimitriou [2005] and Papadimitriou and Roughgarden [2008], which has become standard notation for the literature on CE computation. The notation is slightly different from the one we used in the previous (AGG-specific) chapters. Consider a simultaneous-move game with n players. Denote a player p, and player p's set of pure strategies (i.e., actions) S_p. Let $m = \max_p |S_p|$. Denote a pure strategy profile $s = (s_1, \ldots, s_n) \in S$, with s_p being player p's pure strategy. Denote by S_{-p} the set of partial pure strategy profiles of the players other than p. Player p's utility under pure strategy profile s is u_s^p. We assume that utilities are nonnegative integers (but results in this chapter can be straightforwardly adapted to rational utilities). Denote the largest utility of the game as u.

A *correlated distribution* is a probability distribution over pure strategy profiles, represented by a vector $x \in \mathbb{R}^M$, where $M = \prod_p |S_p|$. Then x_s is the probability of pure strategy profile s under the distribution x. A correlated distribution x is a *product distribution* when it can be achieved by each player p randomizing independently over her actions according to some distribution x^p, i.e., $x_s = \prod_p x_{s_p}^p$. Such a product distribution is also known as a mixed-strategy profile, with each player p playing the mixed strategy x^p.

Throughout the paper we assume that a game is given in a representation satisfying two properties, following Papadimitriou and Roughgarden [2008]:

- *polynomial type*: recall from Section 2.1.1 that this means the number of players and the number of actions for each player are bounded by polynomials in the size of the representation.
- *the polynomial expectation property*: we have access to an algorithm that

computes the expected utility of any player p under any product distribution x, i.e., $\sum_{s\in S} u_s^p x_s$, in time polynomial in the size of the representation.

Definition 7.2.1. *A correlated distribution x is a* correlated equilibrium *(CE) if it satisfies the following* incentive constraints*: for each player p and each pair of her actions $i, j \in S_p$,*

$$\sum_{s\in S_{-p}} [u_{is}^p - u_{js}^p] x_{is} \geq 0, \tag{7.2.1}$$

where the subscript "is" (respectively "js") denotes the pure strategy profile in which player p plays i (respectively j) and the other players play according to the partial profile $s \in S_{-p}$.

We write these incentive constraints in matrix form as $Ux \geq 0$. Thus U is an $N \times M$ matrix, where $N = \sum_p |S_p|^2$. The rows of U, corresponding to the left-hand sides of the constraints (7.2.1), are indexed by (p,i,j) where p is a player and $i, j \in S_p$ are a pair of p's actions. Denote by U_s the column of U corresponding to pure strategy profile s. These incentive constraints, together with the constraints

$$x \geq 0, \quad \sum_{s\in S} x_s = 1, \tag{7.2.2}$$

which ensure that x is a probability distribution, form a linear feasibility program that defines the set of CE. The largest value in U is at most u.

We define the *support* of a correlated equilibrium x as the set of pure strategy profiles assigned positive probability by x. Germano and Lugosi [2007] showed that for any n-player game, there always exists a correlated equilibrium with support size at most $1+\sum_p |S_p|(|S_p|-1) = N+1-\sum_p |S_p|$. Intuitively, such correlated equilibria are basic feasible solutions of the linear feasibility program for CE, i.e., vertices of the polyhedron defining the feasible region. Furthermore, these basic feasible solutions involve only rational numbers for games with rational payoffs (see e.g. Lemma 6.2.4 of [Grötschel et al., 1988]).

7.3 The Ellipsoid Against Hope Algorithm

In this section, we summarize Papadimitriou and Roughgarden's [2008] Ellipsoid Against Hope algorithm for finding a sample CE, which can be seen as an effi-

ciently constructive version of earlier proofs [Hart and Schmeidler, 1989, Myerson, 1997, Nau and McCardle, 1990] of the existence of CE. We will concentrate on the main algorithm and only briefly point out the numerical issues discussed at length by both Papadimitriou and Roughgarden [2008] and Stein et al. [2010], as our analysis will ultimately sidestep these issues.

Papadimitriou and Roughgarden's approach considers the linear program

$$\begin{aligned} &\max \sum_{s \in S} x_s && (P) \\ Ux \geq 0, \qquad & x \geq 0, \end{aligned}$$

which is modified from the linear feasibility program for CE by replacing the constraint $\sum_{s \in S} x_s = 1$ from (7.2.2) with the maximization objective. (P) either has $x = 0$ as its optimal solution or is unbounded; in the latter case, taking a feasible solution and scaling it to be a distribution yields a correlated equilibrium. Thus one way to prove the existence of CE is to show the infeasibility of the dual problem

$$U^T y \leq -1, \qquad y \geq 0. \tag{D}$$

The Ellipsoid Against Hope algorithm uses the following lemma, versions of which were also used by Nau and McCardle [1990] and Myerson [1997].

Lemma 7.3.1 ([Papadimitriou and Roughgarden, 2008])**.** *For every dual vector $y \geq 0$, there exists a product distribution x such that $xU^T y = 0$. Furthermore there exists an algorithm that given any $y \geq 0$, computes the corresponding x (represented by $x^1, \ldots, x^n$) in time polynomial in n and m.*

We will not discuss the details of this algorithm; we will only need the facts that the resulting x is a product distribution and can be computed in polynomial time. Note also that the resulting x is symmetric if y is symmetric. Lemma 7.3.1 implies that the dual problem (D) is infeasible (and therefore a CE must exist): $xU^T y$ is a convex combination of the left hand sides of the rows of the dual, and for any feasible y the result must be less than or equal to -1.

The Ellipsoid Against Hope algorithm runs the ellipsoid algorithm on the dual (D), with the algorithm from Lemma 7.3.1 as separation oracle, which we call the

the Product Separation Oracle. At each step of the ellipsoid algorithm, the separation oracle is given a dual vector $y^{(i)}$. The oracle then generates the corresponding product distribution $x^{(i)}$ and indicates to the ellipsoid algorithm that $(x^{(i)}U^T)y \leq -1$ is violated by $y^{(i)}$. The ellipsoid algorithm will stop after a polynomial number of steps and determine that the program is infeasible. Let X be the matrix whose rows are the generated product distributions $x^{(1)}, \ldots, x^{(L)}$.

Consider the linear program

$$[XU^T]y \leq -1, \qquad y \geq 0, \tag{D'}$$

and observe that the rows of $[XU^T]y \leq -1$ are the cuts generated by the ellipsoid method. If we apply the same ellipsoid method to (D') and use a separation oracle that returns the cut $x^{(i)}U^T y \leq -1$ given query $y^{(i)}$, the ellipsoid algorithm would go through the same sequence of queries $y^{(i)}$ and cutting planes $x^{(i)}U^T y \leq -1$ and return infeasible. Presuming that numerical problems do not arise,[4] we will find that (D') is infeasible. This implies that its dual $[UX^T]\alpha \geq 0$, $\alpha \geq 0$ is unbounded and has polynomial size, and thus can be solved for a nonzero feasible α. We can thus scale α to obtain a probability distribution. We then observe that $X^T\alpha$ satisfies the incentive constraints (7.2.1) and the probability distribution constraints (7.2.2) and is therefore a correlated equilibrium. The distribution $X^T\alpha$ is the mixture of product distributions $x^{(1)}, \ldots, x^{(L)}$ with weights α, and thus can be represented in polynomial space and can be efficiently sampled from.

One issue remains. Although the matrix XU^T is polynomial sized, computing it using matrix multiplication would involve an exponential number of operations. On the other hand, entries of XU^T are differences between expected utilities that arise under product distributions. Since we have assumed that the game represen-

[4] Since each row of (D')'s constraint matrix XU^T may require more bits to represent than any row of the constraint matrix U^T for (D), running the ellipsoid algorithm on (D') with the original bounding ball and volume lower bound for (D) would not be sound, and as a result (D') is not guaranteed to be infeasible. Indeed, Stein et al. [2010] showed that when running the algorithm on their symmetric game example, (D') would remain feasible, and thus the output of the algorithm would not be an exact CE. Furthermore, since the only CE of that game that is a mixture of symmetric product distributions is irrational, there is no way to resolve this issue without breaking at least one of the symmetry and product distribution properties of the Ellipsoid Against Hope algorithm. For more on these issues and possible ways to address them, please see Papadimitriou and Roughgarden [2008, 2010], Stein et al. [2010].

tation admits a polynomial-time algorithm for computing such expected utilities, XU^T can be computed in polynomial time.

Lemma 7.3.2 ([Papadimitriou and Roughgarden, 2008])**.** *There exists an algorithm that given a game representation with polynomial type and satisfying the polynomial expectation property, and given an arbitrary product distribution x, computes xU^T in polynomial time. As a result, XU^T can be computed in polynomial time.*

7.4 Our Algorithm

In this section we present our modification of the Ellipsoid Against Hope algorithm, and prove that it computes exact CE. There are two key differences between our approach and the original algorithm for computing approximate CE.

1. Our modified separation oracle produces pure-strategy-profile cuts;
2. The algorithm is simplified, no longer requiring a special mechanism to deal with numerical issues (because pure-strategy-profile cuts can be represented directly as rows of (D)'s constraint matrix).

7.4.1 The Purified Separation Oracle

We start with a "purified" version of Lemma 7.3.1.

Lemma 7.4.1. *Given any dual vector $y \geq 0$, there exists a pure strategy profile s such that $(U_s)^T y \geq 0$.*

Proof. Recall that Lemma 7.3.1 states that given dual vector $y \geq 0$, a product distribution x can be computed in polynomial time such that $xU^T y = 0$. Since $x[U^T y]$ is a convex combination of the entries of the vector $U^T y$, there must exist some nonnegative entry of $U^T y$. In other words, there exists a pure strategy profile s such that $(U_s)^T y \geq xU^T y = 0$. □

The proof of Lemma 7.4.1 is a straightforward application of the probabilistic method: since $xU^T y$ is the expected value of $(U_s)^T y$ under distribution x, which we denote $E_{s\sim x}[(U_s)^T y]$, the nonnegativity of this expectation implies the existence of

some s such that $(U_s)^T y \geq 0$. Like many other probabilistic proofs, this proof is not efficiently constructive; note that there are an exponential number of possible pure strategy profiles.

It turns out that for game representations with polynomial type and satisfying the polynomial expectation property, an appropriate s can indeed be identified in polynomial time. Our approach can be seen as derandomizing the probabilistic proof using the method of conditional probabilities [Erdős and Selfridge, 1973, Raghavan, 1988, Spencer, 1994]. At a high level, for each player p our algorithm picks a pure strategy s_p, such that the conditional expectation of $(U_s)^T y$ given the choices so far remains nonnegative. This requires us to compute the conditional expectations, but this can be done efficiently using the expected utility subroutine guaranteed by the polynomial expectation property.

Lemma 7.4.2. *There exists a polynomial-time algorithm that given*

- *an instance of a game in a representation satisfying polynomial type and the polynomial expectation property,*
- *a polynomial-time subroutine for computing expected utility under any product distribution (as guaranteed by the polynomial expectation property), and*
- *a dual vector $y \geq 0$,*

finds a pure strategy profile $s \in S$ such that $(U_s)^T y \geq 0$.

Proof. Given a product distribution x, let $x_{(p \to s_p)}$ be the product distribution in which player p plays s_p and all other players play according to x. Since x is a product distribution, $x_{(p \to s_p)} U^T y$ is the conditional expectation of $(U_s)^T y$ given that p plays s_p, and furthermore we have for any p,

$$x U^T y = \sum_{s_p} \left[x_{(p \to s_p)} U^T y \right] x^p_{s_p}. \tag{7.4.1}$$

Since x^p is a distribution, the right hand side of (7.4.1) is a convex combination and thus there must exist an action $s_p \in S_p$ such that $x_{(p \to s_p)} U^T y \geq x U^T y \geq 0$. Since $x_{(p \to s_p)}$ is a product distribution, this process can be repeated for each player

Algorithm 5 Computes a pure strategy profile s such that $(U_s)^T y \geq 0$.

1. Given $y \geq 0$, identify a product distribution x satisfying $xU^T y = 0$, using the algorithm described in Lemma 7.3.1.
2. Sequentially for each player $p \in \{1, \ldots, n\}$,
   (a) iterate through actions $s_p \in S_p$, and compute $x_{(p \to s_p)} U^T$ using the algorithm described in Lemma 7.3.2, until we find an action $s_p^* \in S_p$ such that $\left[x_{(p \to s_p^*)} U^T\right] y \geq 0$.
   (b) set x to be $x_{(p \to s_p^*)}$.
3. The resulting x corresponds to a pure strategy profile s. Output s.

to yield a pure strategy profile s such that $(U_s)^T y \geq xU^T y \geq 0$. This is formalized in Algorithm 5.

We now consider the running time of Algorithm 5. We observe that x remains a product distribution throughout the algorithm and can thus be represented by its marginals $x^1, \ldots, x^n$, requiring only polynomial space. Due to the polynomial expectation property, the algorithm described in Lemma 7.3.2 is polynomial, which implies that in Step 2a, for each $s_p \in S_p$, $x_{(p \to s_p)} U^T$ can be computed in polynomial time. Since Step 2a requires at most $|S_p|$ such computations, and since polynomial type implies that n and $|S_p|$ are polynomial in the input size, the algorithm runs in polynomial time. □

A straightforward corollary is the following:

Corollary 7.4.3. *Algorithm 5 can be used as a separation oracle for the dual LP* (*D*) *in the Ellipsoid Against Hope algorithm: for each query point y, the oracle computes the corresponding pure-strategy profile s according to Algorithm 5 and returns the half space $(U_s)^T y \leq -1$. We call this the Purified Separation Oracle. This separation oracle has the following properties:*

- *Each returned half space is one of the constraints of* (*D*).
- *Since Algorithm 5 iterates through the players sequentially, the generated*

pure-strategy profiles can be asymmetric even for symmetric games and symmetric y.

- *Since a pure-strategy profile is a special case of a product distribution, the resulting pure-strategy profile s also satisfies Lemma 7.3.1, with x being the unit vector corresponding to s.*

7.4.2 The Simplified Ellipsoid Against Hope Algorithm

We now modify the Ellipsoid Against Hope Algorithm by replacing the Product Separation Oracle with our Purified Separation Oracle. The rows of X in (D') become unit vectors corresponding to the pure-strategy profiles generated by the oracle. Thus, we can write (D') as

$$(U')^T y \leq -1, \qquad y \geq 0, \tag{D''}$$

where the matrix $U' \equiv UX^T$ consists of the columns $U_{s^{(i)}}$ that correspond to pure-strategy profiles $s^{(i)}$ generated by the separation oracle. Note that each constraint of (D'') is also one of the constraints of (D), and as a result neither the maximum value of the coefficients nor the right-hand sides of (D'') are greater than in (D). Therefore, a starting ball and volume lower bound that are valid for a run of the ellipsoid method on (D) is also valid for (D''). We thus avoid the precision issues faced by the Ellipsoid Against Hope algorithm, and it is sufficient to use standard values for the initial radius and volume lower bound, and standard perturbation methods for dealing with non-full-dimensional solutions. The resulting CE is a mixture over a polynomial number of pure strategy profiles. We can make a further conceptual simplification of the algorithm: instead of using X as in the Ellipsoid Against Hope algorithm, we can directly treat the generated pure-strategy profiles as columns of U, and use U' in place of UX^T.

We now formally state and prove our result. Note that although we only briefly discussed the way numerical issues are addressed in the original Ellipsoid Against Hope algorithm in Section 7.3, we do go into detail about how our algorithm ensures its own numerical accuracy. Nevertheless that task is comparatively easy, as it is sufficient for us to apply standard techniques from the theory of the ellip-

Algorithm 6 Computes an exact rational CE given a game representation satisfying polynomial type and the polynomial expectation property.

1. Apply the ellipsoid method to (D), using the Purified Separation Oracle, a starting ball with radius of $R = u^{5N^3}$ centered at 0, and stopping when the volume of the ellipsoid is below $v = \alpha_N u^{-7N^5}$, where α_N is the volume of the N-dimensional unit ball.

2. Form the matrix U' whose columns are the $U_{s^{(1)}}, \ldots, U_{s^{(L)}}$ generated by the separation oracle during the run of the ellipsoid method.

3. Compute a basic feasible solution x' of the linear feasibility program

$$U'x' \geq 0, \qquad x' \geq 0, \qquad \mathbf{1}^T x' = 1, \tag{P^*}$$

   by applying the ellipsoid method on the explicitly represented (P^*) and recovering a basis using, e.g., Algorithm 4.2 of Dantzig and Thapa [2003].

4. Output x' and $s^{(1)}, \ldots, s^{(L)}$, interpreted as a distribution over pure-strategy profiles $s^{(1)}, \ldots, s^{(L)}$ with probabilities x'.

soid method. Our analysis makes use of the following lemma from Grötschel et al. [1988].

Lemma 7.4.4 (Lemma 6.2.6, [Grötschel et al., 1988])**.** *Let $P = \{y \in \mathbb{R}^N | Ay \leq b\}$ be a full-dimensional polyhedron defined by the system of inequalities, with the encoding length of each inequality at most φ. Then P contains a ball with radius $2^{-7N^3\varphi}$. Moreover, this ball is contained in the ball with radius $2^{5N^2\varphi}$ centered at 0.*

We note that the only restriction on P is full dimensionality; we do not need to assume that P is bounded, or that A has full row rank.

Theorem 7.4.5. *Given a game representation with polynomial type and satisfying the polynomial expectation property, Algorithm 6 computes an exact and rational CE with support size at most $1 + \sum_p |S_p|(|S_p| - 1)$ in polynomial time.*

Proof. We begin by proving the correctness of the algorithm. First, we will show that the ellipsoid method in Step 1 is a valid run for (D), which certifies that the

feasible set of (D) is either empty or not full dimensional.[5] Suppose the contrary, i.e., the feasible set of (D) is feasible and full dimensional. Since the encoding length of each constraint of (D) is at most $N\log_2 u$, then by Lemma 7.4.4, the feasible set must contain a ball with radius u^{-7N^4}, and thus volume $\alpha_N u^{-7N^5}$, and furthermore this ball must be contained in the ball with radius u^{5N^3} centered at 0, which is the initial ball of our ellipsoid method in Step 1. Since at the end of Step 1 the ellipsoid method certifies that the intersection of the initial ball and the feasible set has volume less than $v = \alpha_N u^{-7N^5}$, we reach a contradiction and therefore either the LP (D) must be infeasible or the feasible set must not be full dimensional. Since the largest magnitude of the coefficients in (D'') is also u, Step 1 is also a valid run for (D'') and therefore either (D'') must be infeasible or the feasible set of (D'') must not be full dimensional.

Of course a non-full-dimensional feasible set is not sufficient for our purpose; we now perturb (D'') to get an infeasible LP. Fix $\rho > 1$. Perturbing the constraints $(U')^T y \leq -1$ of (D'') by multiplying the RHS by ρ, we get the LP:

$$\begin{aligned} &\min 0 \\ &(U')^T y \leq -\rho \mathbf{1} \\ &y \geq 0. \end{aligned} \tag{7.4.2}$$

We claim that (7.4.2) is infeasible. Suppose otherwise: then there exists a $y \in \mathbb{R}^N$ such that $y \geq 0$ and $(U')^T y \leq -\rho\mathbf{1}$. Let $y' \in \mathbb{R}^N$ be a vector such that $0 \leq y'_j - y_j \leq \frac{\rho-1}{Nu}$ for all j. Then $y' \geq 0$, and each component s of $U'^T y'$ satisfies

$$\begin{aligned} (U'_s)^T y &\leq (U'_s)^T y + \frac{\rho-1}{Nu}\sum_j |U'^j_s| \\ &\leq -\rho + \rho - 1 \\ &\leq -1. \end{aligned}$$

Thus, any such y' is feasible for (D''). However, the set of all such vectors y' is a

[5] Since the ellipsoid method relies on shrinking the volume of the candidate set, it is not able to distinguish between non-full-dimensional feasible sets and infeasibility. We overcome this by perturbing the LP after the ellipsoid method has been applied; an alternate method perturbs the LP in advance to ensure the feasible set is either empty or full dimensional.

full-dimensional cube. This contradicts the fact that (D'') is either infeasible or not full dimensional, and therefore (7.4.2) is infeasible. This means that (7.4.2)'s dual

$$\begin{aligned} \max \rho \mathbf{1}^T x' \qquad & (7.4.3) \\ U'x' &\geq 0 \\ x' &\geq 0 \end{aligned}$$

is unbounded (since it is feasible, e.g. $x' = 0$). Then a nonzero feasible vector x' is (after normalization) a distribution over the pure strategy profiles corresponding to columns of U'. Treating it as a sparse representation of a correlated distribution x, it satisfies the feasibility program for CE and is therefore an exact CE.

This CE is exact but its support size could be greater than $1 + \sum_p |S_p|(|S_p| - 1)$ (although as we argue below it is still polynomial). To get a CE with the required support size, we notice that since (7.4.3) is unbounded, a feasible solution of the bounded linear feasibility program (P^*) is a CE. Note that (P^*) has the same set of constraints as the feasibility program for CE defined by (7.2.1) and (7.2.2), and that for each player p and action $i \in S_p$, the incentive constraint (p, i, i) corresponds to deviating from action i to itself and is therefore redundant. Thus the number of bounding constraints of (P^*) is at most $1 + \sum_p |S_p|(|S_p| - 1)$ and therefore a basic feasible solution x' of (P^*) will have the required support size. Since the coefficients and right-hand sides of (P^*) are rational, then (by e.g. Lemma 6.2.4 of Grötschel et al. [1988]) its basic feasible solution x' is also rational and can be represented using at most $4N^3u$ bits.

We now consider the running time of the algorithm. Since Step 1 is a standard run of the ellipsoid method, it terminates in a polynomial number of iterations. For example if we use the ellipsoid algorithm presented in Theorem 3.2.1 of Grötschel et al. [1988], then by Lemma 3.2.10 of Grötschel et al. [1988] the ratio between volumes of successive ellipsoids $\text{vol}(E_{k+1})/\text{vol}(E_k) \leq e^{-1/(5N)}$. With the volume of the initial ellipsoid at most $\alpha_N R^N$ and stopping when volume is below v, the

number of iterations L is at most

$$\begin{aligned}
&5N\left[\ln(\alpha_N R^N) - \ln v\right] \\
&= 5N\left[5N^4 \ln u + 7N^5 \ln u\right] \\
&= O(N^6 \ln u),
\end{aligned}$$

which is polynomial in the input size since $N \equiv \sum_p |S_p|^2$ is polynomial. Since each call to the separation oracle takes polynomial time by Lemma 7.4.2, Step 1 takes polynomial time. L being polynomial also ensures that (P^*) has polynomial size, and thus a basic feasible solution can be found in polynomial time. □

We note that the estimates on R and v (and thus L) can be improved, but our main goal here is to prove that the running time of our algorithm is polynomial.

The reader may wonder how our algorithm would deal with Stein et al. [2010]'s counterexample, a symmetric game in which the only CE that is a convex combination of symmetric product distributions has irrational probabilities. Since we have proved that our algorithm computes a rational CE as a convex combination of product distributions, it must violate the symmetry property. Indeed as we discussed in Section 7.4.1, our Purified Separation Oracle can return asymmetric cuts for symmetric games and symmetric queries, and thus for this game it must return at least one asymmetric cut.

7.5 Uncoupled Dynamics with Polynomial Communication Complexity

Hart and Mansour [2010] considered the setting where each player initially knows only her own utility function, and analyzed the communication complexity for such *uncoupled* dynamics to reach various equilibrium concepts. They use a straightforward adaptation of Papadimitriou and Roughgarden's Ellipsoid Against Hope algorithm to show that a CE can be reached using polynomial communication. The recent discovery by Stein et al. [2010] of flaws of the Ellipsoid Against Hope algorithm imply that Hart and Mansour's procedure as proposed would not reach an exact CE. We show that our modified version of the Ellipsoid Against Hope

algorithm can be straightforwardly adapted into a polynomial communication procedure for exact CE.

Formally, in Hart and Mansour's setting, each player p initially knows only her utility function u^p. No assumption is made on how the game is represented and the cost of computation is of no concern; instead, we focus on the amount of communication required to reach a CE. Hart and Mansour's approach used the following property of the Product Separation Oracle (Lemma 7.3.1): given $y \geq 0$, the corresponding product distribution x depends only on y and not on the utilities of the game. Although generating the cutting plane requires computing xU^T which does depend on the utilities, each entry (p,i,j) of the vector xU^T depends only on the utilities of player p.

We now describe Hart and Masour's procedure. A center runs the Ellipsoid Against Hope algorithm; when the Product Separation Oracle generates a product distribution x, the center sends it to all players, and asks each player p to compute her segment of the vector xU^T, i.e., entries (p,i,j) for all $i,j \in S_p$, to send back to the center. This exactly simulates the Ellipsoid Against Hope algorithm, and its communication costs are those of sending the product distributions to players and each player sending back her part of xU^T.

This procedure can be modified to use the Purified Separation Oracle instead. At Step 2a of the Purified Separation Oracle (Algorithm 5), for each $s_p \in S_p$ the center sends $x_{(p \to s_p)}$ to all players and asks each to compute her segment of $x_{(p \to s_p)}U^T$. After assembling the vector $x_{(p \to s_p)}U^T$ from the segments, the center checks whether $\left[x_{(p \to s_p)}U^T\right] y \geq 0$. We call the resulting modified version of Algorithm 5 the Uncoupled Purified Separation Oracle. It is straightforward to see that this exactly simulates the Purified Separation Oracle. The communication costs are those of the center sending the product distributions and the players sending back segments of $x_{(p \to s_p)}U^T$. At most $\sum_p |S_p|$ rounds of such exchange are required for each call to the Purified Separation Oracle, therefore the total amount of communication is polynomially bounded.

Corollary 7.5.1. *Modify Hart and Mansour's procedure by replacing its separation oracle with the Uncoupled Purified Separation Oracle. The resulting communication procedure reaches an exact CE while both the number of bits of communi-*

cation required and the size of the support are polynomial in n and $\sum_p |S_p|$.

7.6 Computing Extensive-form Correlated Equilibria

Recently, von Stengel and Forges [2008] proposed *extensive-form correlated equilibrium* (EFCE), a solution concept for extensive-form games that is closely related to correlated equilibrium. Here we focus on the computational problem of finding an EFCE and refer interested readers to von Stengel and Forges [2008] for details on EFCE as a solution concept. Huang and Von Stengel [2008] described a polynomial-time algorithm for computing sample extensive-form correlated equilibria. Their algorithm follows a very similar structure as Papadimitriou and Roughgarden's Ellipsoid Against Hope algorithm, and the problems pointed out by Stein et al. [2010] carry over. As a result, the algorithm can fail to find an exact EFCE.

We extend our fix for Papadimitriou and Roughgarden's Ellipsoid Against Hope algorithm to Huang and Von Stengel's algorithm, allowing it to compute an exact EFCE with polynomial-sized support. We first give a high-level description of Huang and Von Stengel's algorithm, following Huang [2011]. [6] The input of the problem is an n-player extensive-form game with perfect recall. Each nonterminal node of the game tree is a decision node for either one of the players or Chance. H denotes the set of information sets, and C_h denotes the set of moves available from $h \in H$, and T denotes the set of terminal nodes. Due to the tree structure of the extensive form, for each node there exists a unique path from the root of the tree to that node. Let s be a pure-strategy profile; $s(h)$ denotes the move at information set $h \in H$. Let z be a distribution over the set of pure-strategy profiles. The size of z is exponential. Huang and Von Stengel [2008] showed that z is an EFCE if it satisfies a polynomial number of linear constraints, which can be written as $Az + Bv \geq 0$ where v is an auxiliary vector of polynomial size. They considered the

[6] We assume that readers are familiar with the standard concepts of extensive form games, information sets, perfect recall, and behavior strategies.

exponential-sized primal LP

$$\begin{aligned} \max \sum_s z_s \qquad & (7.6.1) \\ Az + Bv &\geq 0 \\ z &\geq 0, \end{aligned}$$

and its dual

$$\begin{aligned} A^T y &\leq -1 \qquad (7.6.2) \\ B^T y &= 0 \\ y &\geq 0 \end{aligned}$$

which has a polynomial number of variables and exponential number of constraints. The following is a key lemma:

Lemma 7.6.1. *[Huang and Von Stengel, 2008] For all $y \geq 0$ such that $B^T y = 0$, there exists a product distribution z such that $z^T A^T y = 0$.*

Unlike the simultaneous-move game case, z being a product distribution (mixed-strategy profile) does not imply that it can be concisely represented, as the number of pure strategies for each player can be exponential. Fortunately the z constructed by Lemma 7.6.1 corresponds to a *behavior strategy profile*, which specifies a distribution (denoted z^h) over moves for each information set h. Formally, given z^h for all $h \in H$, the resulting distribution over pure-strategy profiles is given by

$$\forall s, \qquad z_s = \sum_{t \in T : t \text{ agrees with } s} p(t) x_t,$$

where we say *t agrees* with pure-strategy profile s if all the moves by the players on the path from the root to t are given by s, $p(t)$ is the product of probabilities of moves by Chance along the path from the root to t, and $x_t = \prod_{h \text{ precedes } t} z^h_{s(h)}$ is the product of probabilities of moves by the players along the path from the root to t. Here by "h precedes t" we mean that h is an information set on the path from the root to t. Note that perfect recall ensures that an information set h appears at most once along the path from the root to t. Such a behavior strategy profile requires

only a polynomial number of values to specify. Given y, the corresponding z can be computed in polynomial time.

By the same argument as for the Ellipsoid Against Hope algorithm, Lemma 7.6.1 implies the infeasibility of (7.6.2), and can be used as a separation oracle for a ellipsoid method on (7.6.2). In order to generate the cutting plane $[zA^T]y \leq -1$, the oracle needs to compute zA^T whose inner dimensions are exponential. It turned out that zA^T can be formulated as expected utility computations which can be carried out in polynomial time. Huang and Von Stengel's algorithm thus proceeds similarly as in the Ellipsoid Against Hope algorithm to produce a feasible solution to (7.6.1), which can be scaled to be an EFCE.

By the same argument as our fix of the Ellipsoid Against Hope algorithm, in order to overcome the problems pointed out by Stein et al. [2010] it is sufficient to construct a Purified Separation Oracle that given a $y \geq 0$ such that $B^T y = 0$, computes a pure-strategy profile s such that $(A_s)^T y \geq 0$. We construct such an oracle using a similar application of the method of conditional probabilities. For a behavior strategy profile z, an information set h, and a move $d \in C_h$, define $z_{(h \to d)}$ to be the behavior strategy profile that is identical to z except at information set d, where the corresponding player deterministically chooses d instead. Our Purified Separation Oracle starts with the behavior strategy profile constructed by Lemma 7.6.1, and uses the same algorithm as Algorithm 5, except that instead of going through players in step 2a, we go through information sets sequentially, and for each information set h we iterate through $z_{(h \to d)}$ until we find a d^* such that $[z_{(h \to d^*)} A^T]y \geq 0$.

To show that our algorithm is correct, we use the following lemma:

Lemma 7.6.2. *Given a behavior strategy profile z, for each information set h,*

$$z = \sum_{d \in C_h} z_{(h \to d)} z^h_d,$$

where z^h_d is the probability of choosing d at h prescribed by z.

Proof. Recall that

$$z_s = \sum_{t \in T : t \text{ agrees with } s} p(t) x_t,$$

where $x_t = \prod_{h \text{ precedes } t} z^h_{s(h)}$. Since the moves along the path to t are uniquely deter-

mined by t, x_t is fully specified by the behavior strategies and does not depend on s. We can write this in matrix form as $z = Fx$, with $x \in \mathbb{R}^{|T|}$. Let $x_{(h \to d)} \in \mathbb{R}^{|T|}$ be the vector induced by behavior strategy profile $z_{(h \to d)}$. We then have $z_{(h \to d)} = F x_{(h \to d)}$. Furthermore, we observe that for all h,

$$x = \sum_{d \in C_h} x_{(h \to d)} z^h_d.$$

(It is straightforward to verify the above by considering the terminal nodes t for which h precedes t and then the other terminal nodes.) We thus have

$$z = Fx = F \sum_{d \in C_h} x_{(h \to d)} z^h_d = \sum_{d \in C_h} z_{(h \to d)} z^h_d,$$

which is the required equality. □

The correctness and the polynomial running time of our algorithm for Purified Separation Oracle then follow by the same argument as in the proof of Lemma 7.4.2. After modifying Huang and Von Stengel's algorithm by replacing their separation oracle with our Purified Separation Oracle, the resulting algorithm computes in polynomial time an exact EFCE that is a mixture of a polynomial number of pure-strategy profiles.

Corollary 7.6.3. *Given a game in extensive form, an exact EFCE with polynomial-sized support can be computed in polynomial time.*

7.7 Conclusion

We have proposed a polynomial-time algorithm, a variant of Papadimitriou and Roughgarden's Ellipsoid Against Hope approach, for computing an exact CE given a game representation with polynomial type and satisfying the polynomial expectation property. A key component of our approach is a derandomization of Papadimitriou and Roughgarden's separation oracle using the method of conditional probabilities, yielding a polynomial-time separation oracle that outputs cuts corresponding to pure-strategy profiles. Our approach is then spared from dealing with the numerical precision issues that were a major focus of previous approaches, and the

algorithm is considerably simplified as a result. Furthermore, the correlated equilibria returned by our algorithm have polynomial-sized supports. We expect these properties of our algorithm to be independently interesting, beyond its usefulness in resolving the recent uncertainty about the computational complexity of identifying exact CE. For example, we show that our techniques can be adapted to two existing algorithms that are based on the Ellipsoid Against Hope approach, Hart and Mansour's [2010] CE procedure with polynomial communication complexity and Huang and Von Stengel's [2008] polynomial-time algorithm for extensive-form correlated equilibria, yielding in both cases exact solutions with polynomial-sized supports.

Our algorithm has additional practical benefits: the resulting cutting planes are deeper cuts than those produced by the original oracle, resulting in a smaller number of iterations required to reach convergence, albeit at the cost of more work per iteration. It is also possible to return cuts corresponding to pure strategy profiles with (e.g.) good social welfare, yielding a heuristic method for generating correlated equilibria with good social welfare. However, recall from Section 2.2.7 that finding a CE with optimal social welfare is generally NP-hard for many game representations [Papadimitriou and Roughgarden, 2008]. In Chapter 8 we analyze the optimal CE problem using a somewhat different approach.

Chapter 8

A General Framework for Computing Optimal Correlated Equilibria in Compact Games

8.1 Introduction

In this chapter we[1] continue to focus on correlated equilibrium (CE). We have seen from the previous chapter and its related literature [Jiang and Leyton-Brown, 2011, Papadimitriou and Roughgarden, 2008] that finding a sample CE is tractable, even for compactly represented games. However, since in general there can be an infinite number of CE even in a generic game, finding an arbitrary one is of limited value. Instead, here we focus on the problem of computing a correlated equilibrium that optimizes some objective. In particular we consider two kinds of objectives: (1) A linear function of players' expected utilities. For example, computing the best (or worst) social welfare corresponds to maximizing (or minimizing) the sum of players' utilities, respectively. (2) Max-min welfare: maximizing the utility of the worst-off player. (More generally, maximizing the minimum of a set of linear functions of players' expected utilities.) We are also interested in comput-

[1] This chapter is based on joint work with Kevin Leyton-Brown. A shorter version is published in the Proceedings of the Seventh Workshop on Internet and Network Economics (WINE), 2011.

ing optimal coarse correlated equilibrium (CCE) [Hannan, 1957]. Recall from Section 2.2.7 that the empirical distribution of any no-external-regret learning dynamic converges to the set of CCE, while the empirical distribution of no-internal-regret learning dynamics converges to the set of CE. Thus, optimal CE / CCE provide useful bounds on the social welfare of the empirical distributions of these dynamics. Optimal CE / CCE can also be used as bounds on optimal NE since CE and CCE are both relaxations of NE. Hence they are also useful for computing (bounds on) the price of anarchy and price of stability of a game. The problems of computing optimal CE / CCE can be formulated as linear programs with sizes polynomial in the size of normal form. However, as with the rest of the thesis, we are interested in the case when the input is a compactly-represented game.

We are particularly interested in the relationship between the optimal CE / CCE problems and the problem of computing the optimal social welfare outcome (i.e. strategy profile) of the game, which is exactly the optimal social welfare CE problem without the incentive constraints. This is an instance of a line of questions that has received much interest from the algorithmic game theory community: "How does adding incentive constraints to an optimization problem affect its complexity?" This question in the mechanism design setting is perhaps one of the central questions of algorithmic mechanism design [Nisan and Ronen, 2001]. Of course, a more constrained problem can in general be computationally easier than the relaxed version of the problem. Nevertheless, results from complexity of Nash equilibria and algorithmic mechanism design suggest that adding *incentive constraints* to a problem is unlikely to decrease its computational difficulty. That is, when the optimal social welfare problem is hard, we tend also to expect that the optimal CE problem will be hard as well. On the other hand, we are interested in the other direction: when it is the case for a class of games that the optimal social welfare problem can be efficiently computed, can the same structure be exploited to efficiently compute the optimal CE?

As mentioned in Section 2.2.7, Papadimitriou and Roughgarden [2008] considered the optimal linear objective CE problem and proved that the problem is NP-hard for many representations, while tractable for a couple of representations. We now take a more in-depth look at this paper. In particular, the representations shown to be NP-hard include graphical games, polymatrix games, and congestion games.

These hardness results, although nontrivial, are not surprising: the optimal social welfare problem is already NP-hard for these representations. On the tractability side, Papadimitriou and Roughgarden [2008] focused on so-called "reduced form" representations, meaning representations for which there exist player-specific partitions of the strategy profile space into payoff-equivalent outcomes. They showed that if a particular *separation problem* is polynomial-time solvable, the optimal CE problem is polynomial-time solvable as well. Finally, they showed that this separation problem is polynomial-time solvable for bounded-treewidth graphical games, symmetric games and anonymous games.

Perhaps most surprising and interesting is the *form* of Papadimitriou and Roughgarden's sufficient condition for tractability: their separation problem for an instance of a reduced-form-based representation is essentially equivalent to solving the optimal social welfare problem for an instance of that representation with the same reduced form but possibly different payoffs. In other words, if we have a polynomial-time algorithm for the optimal social welfare problem for a reduced-form-based representation, we can turn that into a polynomial-time algorithm for the optimal social welfare CE problem. However, Papadimitriou and Roughgarden's sufficient condition for tractability only applies to reduced-form-based representations. Their definition of reduced forms is unable to handle representations that exploit linearity of utility, and in which the structure of player p's utility function may depend on the action she chose. As a result, many representations do not fall into this characterization, such as polymatrix games, congestion games, and action-graph games. Although the optimal CE problems for these representations are NP-hard in general, we are interested in identifying tractable subclasses of games, and a sufficient condition that applies to all representations would be helpful.

In this chapter, we propose a different algorithmic approach for the optimal CE problem that applies to *all* compact representations. By applying the ellipsoid method to the dual of the LP for optimal CE, we show that the polynomial-time solvability of what we call the *deviation-adjusted social welfare problem* is a sufficient condition for the tractability of the optimal CE problem. We also give a sufficient condition for tractability of the optimal CCE problem: the polynomial-time solvability of the *coarse deviation-adjusted social welfare problem*, which we

show reduces to the deviation-adjusted social welfare problem. Our algorithms are instances of the black-box approach, with the required subroutines being the computations of the deviation-adjusted social welfare problem and the coarse deviation-adjusted social welfare problem, respectively. We show that for reduced-form-based representations, the deviation-adjusted social welfare problem can be reduced to the separation problem of Papadimitriou and Roughgarden [2008]. Thus the class of reduced forms for which our problem is polynomial-time solvable contains the class for which the separation problem is polynomial-time solvable. More generally, we show that if a representation can be characterized by "linear reduced forms", i.e. player-specific linear functions over partitions, then for that representation, the deviation-adjusted social welfare problem can be reduced to the optimal social welfare problem. As an example, we show that for graphical polymatrix games on trees, optimal CE can be computed in polynomial time. Such games are not captured by the reduced-form framework.[2] The key feature of these representations upon which our argument relies is that the partitions for player p (which characterize the structure of the utility function for p) do not depend on the action chosen by p.

On the other hand, representations like action-graph games and congestion games have *action-specific* structure, and as a result the deviation-adjusted social welfare problems and coarse deviation-adjusted social welfare problems on these representations are structured differently from the corresponding optimal social welfare problems. Nevertheless, we are able to show a polynomial-time algorithm for the optimal CCE problem on *singleton congestion games* [Ieong et al., 2005], a subclass of congestion games. We use a symmetrization argument to reduce the optimal CCE problem to the coarse deviation-adjusted social welfare problem with player-symmetric deviations, which can be solved using a dynamic-programming algorithm. This is an example where the optimal CCE problem is tractable while the complexity of the optimal CE problem is not yet known.

[2] In a recent paper Kamisetty et al. [2011] has independently proposed an algorithm for optimal CE in graphical polymatrix games on trees. They used a different approach that is specific to graphical games and graphical polymatrix games, and it is not obvious whether their approach can be extended to other classes of games.

8.2 Problem Formulation

We follow the notation of Chapter 7. Furthermore, let $\mathcal{N} = \{1, \ldots, n\}$ be the set of players. Let w be the vector of social welfare for each pure profile, that is $w = \sum_{p \in \mathcal{N}} u^p$, with w_s denoting the social welfare for pure profile s.

Throughout the chapter we assume that the game is given in a representation with polynomial type. Unlike in Chapter 7, here we do not assume the existence of a polynomial-time algorithm for expected utility.

8.2.1 Correlated Equilibrium

Correlated equilibrium (CE) is defined in Definition 7.2.1. The problem of computing a maximum social welfare CE can be formulated as the LP

$$\begin{aligned} \max\, & w^T x && (P) \\ & Ux \geq 0 \\ & x \geq 0 \\ & \sum_{s \in S} x_s = 1 \end{aligned}$$

Another objective of interest is the max-min welfare CE problem: computing a CE that maximizes the utility of the worst-off player.

$$\begin{aligned} \max\, & r && (8.2.1) \\ & \sum_s x_s u_s^p \geq r \qquad \forall p && (8.2.2) \\ & Ux \geq 0 \\ & x \geq 0 \\ & \sum_{s \in S} x_s = 1 \end{aligned}$$

Another solution concept of interest is *coarse correlated equilibrium* (CCE). Whereas CE requires that each player has no profitable deviation even if she takes into account the signal she receives from the intermediary, CCE only requires that each player has no profitable *unconditional deviation*.

Definition 8.2.1. *A correlated distribution x is a* coarse correlated equilibrium *(CCE) if it satisfies the following incentive constraints: for each player p and each of his actions $j \in S_p$,*

$$\sum_{(i,s_{-p}) \in S} [u^p_{is_{-p}} - u^p_{js_{-p}}] x_{is_{-p}} \geq 0. \tag{8.2.3}$$

We write these incentive constraints in matrix form as $Cx \geq 0$. Thus C is an $(\sum_p |S_p|) \times M$ matrix. By definition, a CE is also a CCE.

The problem of computing a maximum social welfare CCE can be formulated as the LP

$$\begin{aligned} \max\, & w^T x && (CP) \\ Cx &\geq 0 \\ x &\geq 0 \\ \sum_{s \in S} x_s &= 1. \end{aligned}$$

8.3 The Deviation-Adjusted Social Welfare Problem

Consider the dual of (*P*),

$$\begin{aligned} \min\, & t && (D) \\ U^T y + w &\leq t\mathbf{1} \\ y &\geq 0. \end{aligned}$$

We label the (p,i,j)-th element of $y \in \mathbb{R}^N$ (corresponding to row (p,i,j) of U) as $y^p_{i,j}$. This is an LP with a polynomial number of variables and an exponential number of constraints. Given a separation oracle, we can solve it in polynomial time using the ellipsoid method. A separation oracle needs to determine whether a given (y,t) is feasible, and if not output a hyperplane that separates (y,t) from the feasible set. We focus on a restricted form of separation oracles, which outputs a violated constraint for infeasible points.[3] Such a separation oracle needs to solve

[3]This is a restriction because in general there exist separating hyperplanes other than the violated constraints. For example as we saw in Chapter 7, Papadimitriou and Roughgarden [2008]'s algo-

the following problem:

Problem 8.3.1. *Given (y,t) with $y \geq 0$, determine if there exists an s such that $(U_s)^T y + w_s > t$; if so output such an s.*

The left-hand-side expression $(U_s)^T y + w_s$ is the social welfare at s plus the term $(U_s)^T y$. Observe that the (p,i,j)-th entry of U_s is $u_s^p - u_{js_{-p}}^p$ if $s_p = i$ and is zero otherwise. Thus $(U_s)^T y = \sum_p \sum_{j \in S_p} y_{s_p,j}^p \left(u_s^p - u_{js_{-p}}^p\right)$. We now reexpress $(U_s)^T y + w_s$ in terms of *deviation-adjusted utilities* and *deviation-adjusted social welfare*.

Definition 8.3.2. *Given a game, and a vector $y \in \mathbb{R}^N$ such that $y \geq 0$, the* deviation-adjusted utility *for player p under pure profile s is*

$$\hat{u}_s^p(y) = u_s^p + \sum_{j \in S_p} y_{s_p,j}^p \left(u_s^p - u_{js_{-p}}^p\right).$$

The deviation-adjusted social welfare is $\hat{w}_s(y) = \sum_p \hat{u}_s^p(y)$.

By construction, the deviation-adjusted social welfare $\hat{w}_s(y) = \sum_p u_s^p + \sum_p \sum_{j \in S_p} y_{s_p,j}^p \left(u_s^p - u_{js_{-p}}^p\right) = (U_s)^T y + w_s$. Therefore, Problem 8.3.1 is equivalent to the following *deviation-adjusted social welfare problem*.

Definition 8.3.3. *For a game representation, the* deviation-adjusted social welfare problem *is the following: given an instance of the representation and rational vector $(y,t) \in \mathbb{Q}^{N+1}$ such that $y \geq 0$, determine if there exists an s such that the deviation-adjusted social welfare $\hat{w}_s(y) > t$; if so output such an s.*

Proposition 8.3.4. *If the deviation-adjusted social welfare problem can be solved in polynomial time for a game representation, then so can the problem of computing the maximum social welfare CE.*

Proof. Recall that an algorithm for Problem 8.3.1 can be used as a separation oracle for (*D*). Then we can apply the ellipsoid method using the given algorithm for the deviation-adjusted social welfare problem as a separation oracle. This solves

rithm for computing a sample CE uses a separation oracle that outputs a convex combination of the constraints as a separating hyperplane.

(D) in polynomial time. By LP duality, the optimal objective of (D) is the social welfare of the optimal CE. The cutting planes generated during the ellipsoid method can then be used to compute such a CE with polynomial-sized support. □

We observe that our approach has certain similarities to the Ellipsoid Against Hope algorithm and its variants discussed in Chapter 7: both approaches are black-box approaches based on LP duality formulations of the respective problems, and both make use of the ellipsoid method to overcome the exponential size of the LPs. On the other hand, due to the different LP formulations of the sample CE problem and the optimal CE problem respectively, the two approaches require different separation oracles, which leads to the different requirements on the subroutines provided by the representation.

Let us consider interpretations of the dual variables y and the deviation-adjusted social welfare of a game. The dual (D) can be rewritten as $\min_{y\geq 0} \max_s \tilde{w}_s(y)$. By weak duality, for a given $y \geq 0$ the maximum deviation-adjusted social welfare $\max_s \tilde{w}_s(y)$ is an upper bound on the maximum social welfare CE. So the task of the dual (D) is to find y such that the resulting maximum deviation-adjusted social welfare gives the tightest bound.[4] At optimum, y corresponds to the concept of "shadow prices" from optimization theory; that is, y^p_{ij} equals the rate of change in the social welfare objective when the constraint (p,i,j) is relaxed infinitesimally. Compared to the maximum social welfare CE problem, the maximum deviation-adjusted social welfare problem replaces the incentive constraints with a set of additional penalties or rewards. Specifically, we can interpret y as a set of nonnegative prices, one for each incentive constraint (p,i,j) of (P). At strategy profile s, for each incentive constraint (p,i,j) we impose a penalty equal to y^p_{ij} times the amount the constraint (p,i,j) is violated by s. Note that the penalty can be negative, and is zero if $s_p \neq i$. Then $\tilde{w}_s(y)$ is equal to the social welfare of the modified game.

Practical computation. We have seen from Chapters 2, 3 and 7 that the problem of computing the expected utility given a mixed strategy profile has been established as an important subproblem for both the sample NASH problem and

[4] An equivalent perspective is to view y as Lagrange multipliers, and the optimal deviation-adjusted SW problem as the Lagrangian relaxation of (P) given the multipliers y.

the sample CE problem, both in theory and in practice. Our results in this chapter suggest that the deviation-adjusted social welfare problem is of similar importance to the optimal CE problem. This connection is more than theoretical: our algorithmic approach can be turned into a practical method for computing optimal CE. In particular, although it makes use of the ellipsoid method, we can easily substitute a more practical method, such as simplex with column generation. In contrast, Papadimitriou and Roughgarden [2008]'s algorithmic approach for reduced forms makes two nested applications of the ellipsoid method, and is less likely to be practical. Furthermore, even for representations without a polynomial-time algorithm for the deviation-adjusted social welfare problem, a promising direction would be to formulate the deviation-adjusted social welfare problem as a integer program or constraint program and solve using e.g. CPLEX.

8.3.1 The Weighted Deviation-Adjusted Social Welfare Problem

For the max-min welfare CE problem, we can form the dual of (8.2.1),

$$\min t \tag{8.3.1}$$

$$U^T y + \sum_p v_p u^p \leq t\mathbf{1} \tag{8.3.2}$$

$$y \geq 0,\ v \geq 0$$

$$\sum_p v_p = 1.$$

This is again an LP with polynomial number of variables and exponential number of constraints; specifically, block (8.3.2) is exponential. We observe that (8.3.2) is similar to the corresponding block in (D), except for the weighted sum $\sum_p v_p u^p$ instead of the social welfare w. Thus, in order to express the left-hand side of (8.3.2) we need notions slightly different from those given in Definition 8.3.2, which we call *weighted deviation-adjusted utility* and *weighted deviation-adjusted social welfare*.

Definition 8.3.5. *Given a game, a vector $y \in \mathbb{R}^N$ such that $y \geq 0$, and a vector $v \in \mathbb{R}^n$ such that $v \geq 0$ and $\sum_p v_p = 1$, the* weighted deviation-adjusted utility *for*

player p under pure profile s is

$$\hat{u}_s^p(y,v) = v_p u_s^p + \sum_{j \in S_p} y^p_{s_p,j}(u_s^p - u^p_{js_{-p}}).$$

The weighted deviation-adjusted social welfare is $\hat{w}_s(y,v) = \sum_p \hat{u}_s^p(y,v)$.

Following analysis similar to that given above, the following problem serves as a separation oracle of LP (8.3.1).

Definition 8.3.6. *For a game representation, the* weighted deviation-adjusted social welfare problem *is the following: given an instance of the representation, and rational vector $(y,v,t) \in \mathbb{Q}^{N+n+1}$ such that $y \geq 0$, $v \geq 0$ and $\sum_p v_p = 1$, determine if there exists an s such that the deviation-adjusted social welfare $\hat{w}_s(y) > t$; if so output such an s.*

Proposition 8.3.7. *If the weighted deviation-adjusted social welfare problem can be solved in polynomial time for a game representation, then the problem of computing the max-min welfare CE is in polynomial time for this representation.*

It is straightforward to see that the deviation-adjusted social welfare problem reduces to the weighted deviation-adjusted social welfare problem. In all representations that we consider in this chapter, the weighted and unweighted versions have the same structure and thus the same complexity.

8.3.2 The Coarse Deviation-Adjusted Social Welfare Problem

For the optimal social welfare CCE problem, we can form the dual of (*CP*)

$$\begin{aligned} &\min t \\ C^T y + w &\leq t\mathbf{1} \\ y &\geq 0 \end{aligned} \tag{8.3.3}$$

Definition 8.3.8. *We label the (p,j)-th element of y as y^p_j. Given a game, and a vector $y \in \mathbb{R}^{\sum_p |S_p|}$ such that $y \geq 0$, the* coarse deviation-adjusted utility *for player*

p under pure profile s is

$$\tilde{u}_s^p(y) = u_s^p + \sum_{j \in S_p} y_j^p (u_s^p - u_{js_{-p}}^p).$$

The coarse deviation-adjusted social welfare is $\tilde{w}_s(y) = \sum_p \tilde{u}_s^p(y)$.

Proposition 8.3.9. *If the coarse deviation-adjusted social welfare problem can be solved in polynomial time for a game representation, then the problem of computing the maximum social welfare CCE is in polynomial time for this representation.*

The coarse deviation-adjusted social welfare problem reduces to the deviation-adjusted social welfare problem. To see this, given an input vector y for the coarse deviation-adjusted social welfare problem, we can construct an input vector $y' \in \mathbb{Q}^N$ for the deviation-adjusted social welfare problem with $y'^p_{ij} = y_j^p$ for all $p \in \mathcal{N}$ and $i, j \in S_p$.

8.4 The Deviation-Adjusted Social Welfare Problem for Specific Representations

In this section we study the deviation-adjusted social welfare problem and its variants on specific representations. Depending on the representation, the deviation-adjusted social welfare problem is not always solvable in polynomial time. Indeed, Papadimitriou and Roughgarden [2008] showed that for many representations the problem of optimal CE is NP-hard. Nevertheless, for such representations we can often identify tractable subclasses of games. We will argue that the deviation-adjusted social welfare problem is a more useful formulation for identifying tractable classes of games than the separation problem formulation of Papadimitriou and Roughgarden [2008], as the latter only applies to reduced-form-based representations.

8.4.1 Reduced Forms

Papadimitriou and Roughgarden [2008] gave the following reduced form characterization of representations.

Definition 8.4.1 ([Papadimitriou and Roughgarden, 2008])**.** *Consider a game* $G = (\mathcal{N}, \{S_p\}_{p\in\mathcal{N}}, \{u^p\}_{p\in\mathcal{N}})$*. For* $p = 1,\ldots,n$*, let* $P_p = \{C_p^1 \ldots C_p^{r_p}\}$ *be a partition of* S_{-p} *into* r_p *classes. The set* $\mathscr{P} = \{P_1,\ldots,P_n\}$ *of partitions is a* reduced form *of* G *if* $u_s^p = u_{s'}^p$ *whenever (1)* $s_p = s'_p$ *and (2) both* s_{-p} *and* s'_{-p} *belong to the same class in* P_p*. The* size *of a reduced form is the number of classes in the partitions plus the bits required to specify a payoff value for each tuple* (p,k,ℓ) *where* $1 \le p \le n$*,* $1 \le k \le r_p$ *and* $\ell \in S_p$*.*

Intuitively, the reduced form imposes the condition that p's utility for choosing an action s_p depends only on which *class* in the partition P_p the profile of the others' actions belongs to. Papadimitriou and Roughgarden [2008] showed that several compact representations such as graphical games and anonymous games have natural reduced forms whose sizes are (roughly) equal to the sizes of the representation. We say such a compact representation has a *concise reduced form*. Intuitively, such a reduced form describes the structure of the game's utility functions.

Example 8.4.2. *Recall from Section 2.1.1 that a graphical game [Kearns et al., 2001] is associated with a graph* $(\mathcal{N}, E)$*, such that player* p*'s utility depends only on her action and the actions of her neighbors in the graph. The sizes of the utility functions are exponential only in the degrees of the graph. Such a game has a natural reduced form where the classes in* P_p *are identified with the pure profiles of* p*'s neighbors, i.e.,* s_{-p} *and* s'_{-p} *belong to the same class if and only if they agree on the actions of* p*'s neighbors. The size of the reduced form is exactly the number of utility values required to specify the graphical game's utility functions.* □

Let $\mathscr{S}_p(k,\ell)$ denote the set of pure strategy profiles s such that $s_p = \ell$ and s_{-p} is in the k-th class C_p^k of P_p, and let $u^p_{(k,\ell)}$ denote the utility of p for that set of strategy profiles. Papadimitriou and Roughgarden [2008] defined the following *Separation Problem* for a reduced form.

Definition 8.4.3 ([Papadimitriou and Roughgarden, 2008])**.** *Let* $\mathscr{P}$ *be a reduced form for game* G*. The* Separation Problem *for* $\mathscr{P}$ *is the following: Given rational numbers* $\gamma_p(k,\ell)$ *for all* $p \in \{1,\ldots,n\}$*,* $k \in \{1,\ldots,r_p\}$*, and* $\ell \in S_p$*, is there a pure strategy profile* s *such that* $\sum_{p,k,\ell:s\in\mathscr{S}_p(k,\ell)} \gamma_p(k,\ell) < 0$*? If so, find such an* s*.*

Since $s \in \mathscr{S}_p(k,\ell)$ implies $s_p = \ell$, the left-hand side of the above expression is equivalent to $\sum_p \sum_{k:s\in\mathscr{S}_p(k,s_p)} \gamma_p(k,s_p)$. Furthermore, since s belongs to exactly one class in P_p, the expression is a sum of exactly n summands, one for each player.

Papadimitriou and Roughgarden [2008] proved that if the separation problem can be solved in polynomial time, then a CE that maximizes a given linear objective in the players' utilities can be computed in time polynomial in the size of the reduced form. How does Papadimitriou and Roughgarden [2008]'s sufficient condition relate to ours, provided that the game has a concise reduced form? We show that the class of reduced form games for which our weighted deviation-adjusted social welfare problem is polynomial-time solvable contains the class for which the separation problem is polynomial-time solvable.

Proposition 8.4.4. *Let $\mathscr{P}$ be a reduced form for game G. Suppose the separation problem can be solved in polynomial time. Then the weighted deviation-adjusted social welfare problem can be solved in time polynomial in the size of the reduced form.*

Proof. First we observe that if a game G has a reduced form $\mathscr{P}$, then its deviation-adjusted utilities (and weighted deviation-adjusted utilities) also satisfy the partition structure specified by $\mathscr{P}$, i.e., given y and v, the weighted deviation-adjusted utility $\hat{u}^p_s(y,v)$ depends only on a player's action s_p and the class in P_p that s_{-p} belongs to. To see why, suppose $s_{-p} \in C^k_p$. Then

$$\begin{aligned}\hat{u}^p_{\ell s_{-p}}(y,v) &= v_p u^p_{\ell s_{-p}} + \sum_{j\in S_p} y^p_{\ell,j}(u^p_{\ell s_{-p}} - u^p_{j s_{-p}}) \\ &= v_p u^p_{(k,\ell)} + \sum_{j\in S_p} y^p_{\ell,j}(u^p_{(k,\ell)} - u^p_{(k,j)}),\end{aligned}$$

which depends only on ℓ and k. This proves the following, which will be useful later.

Lemma 8.4.5. *Let $\mathscr{P}$ be a reduced form for game G.*

1. *For all $y \in \mathbb{R}^N$, $v \in \mathbb{R}^n$, for all players p, $s_p \in S_p$, and for all $s_{-p}, s'_{-p} \in S_{-p}$, if s_{-p} and s'_{-p} are in the same class in P_p then the weighted deviation-adjusted utilities $\hat{u}^p_{s_p,s_{-p}}(y,v) = \hat{u}^p_{s_p,s'_{-p}}(y,v)$.*

2. *Write the weighted deviation-adjusted utility for player p, given her pure strategy $\ell \in S_p$ and class C_p^k, as $\hat{u}^p_{(k,\ell)}(y,v)$ (well defined by the above). We have*

$$\hat{u}^p_{(k,\ell)}(y,v) \equiv v_p u^p_{(k,\ell)} + \sum_{j \in S_p} y^p_{\ell,j}(u^p_{(k,\ell)} - u^p_{(k,j)}).$$

Given an instance of the weighted deviation-adjusted social welfare problem with a game with reduced form $\mathscr{P}$ and rational vectors $y \in \mathbb{R}^N$, $v \in \mathbb{R}^n$ and $t \in \mathbb{R}$, we construct an instance of the separation problem by letting $\gamma_p(k,\ell) = t/n - \hat{u}^p_{(k,\ell)}(y,v)$, where $\hat{u}^p_{(k,\ell)}(y,v)$ is as defined in Lemma 8.4.5 and can be efficiently computed given the reduced form. Recall that the separation problem asks for pure profile s such that $\sum_{p,k,\ell:s\in\mathscr{S}_p(k,\ell)} \gamma_p(k,\ell) < 0$, the left hand side of which is a sum of n terms. By construction, for all s, $\sum_{p,k,\ell:s\in\mathscr{S}_p(k,\ell)} \gamma_p(k,\ell) < 0$ if and only if $\sum_p \sum_{k:s\in\mathscr{S}_p(k,s_p)} \left(t/n - \hat{u}^p_{(k,s_p)}(y,v)\right) < 0$, and since the left hand side is a sum of n terms, this holds if and only if $\hat{w}^p_s(y,v) > t$. Therefore the weighted deviation-adjusted social welfare problem instance has a solution s if and only if the corresponding separation problem instance has a solution s, and a polynomial-time algorithm for the separation problem can be used to solve the weighted deviation-adjusted social welfare problem in polynomial time. □

We now compare the the weighted deviation-adjusted social welfare problem with the optimal social welfare problem for these representations. We observe from Lemma 8.4.5 that the weighted deviation-adjusted social welfare problem can be formulated as an instance of the optimal social welfare problem on another game with the same reduced form but different payoffs. Can we claim that the existence of a polynomial-time algorithm for the optimal social welfare problem for a representation implies the existence of a polynomial-time algorithm for the weighted social welfare problem (and thus the optimal CE problem)? This is not necessarily the case, because the representation might impose certain structure on the utility functions that are not captured by the reduced forms, and the polynomial-time algorithm for the optimal social welfare problem could depend on the existence of such structure. The weighted deviation-adjusted social welfare problem might no longer exhibit such structure and thus might not be solvable using the given algorithm.

Nevertheless, if we consider a game representation that is "completely charac-

terized" by its reduced forms, the weighted deviation-adjusted social welfare problem is equivalent to the decision version of the optimal social welfare outcome problem for that representation. To make this more precise, we say a game representation is a *reduced-form-based representation* if there exists a mapping from instances of the representation to reduced forms such that it maps each instance to a concise reduced form of that instance, and if we take such a reduced form and change its payoff values arbitrarily, the resulting reduced form is a concise reduced form of another instance of the representation.

Corollary 8.4.6. *For a reduced-form-based representation, if there exists a polynomial-time algorithm for the optimal social welfare problem, then the optimal social welfare CE problem and the max-min welfare CE problem can be solved in polynomial time.*

Of course, this can be derived using the separation problem for reduced forms without the deviation-adjusted social welfare formulation. On the other hand, the deviation-adjusted social welfare formulation can be applied to representations without concise reduced forms. In fact, we will use it to show below that the connection between the optimal social welfare problem and the optimal CE problem applies to a wider classes of representations than just reduced-form-based representations.

8.4.2 Linear Reduced Forms

One class of representations that does not have concise reduced forms are those that represent utility functions as sums of other functions, such as polymatrix games and the hypergraph games of Papadimitriou and Roughgarden [2008]. In this section we characterize these representations using linear reduced forms, showing that linear-reduced-form-based representations satisfy a property similar to Corollary 8.4.6.

Roughly speaking, a linear reduced form has multiple partitions for each agent, rather than just one; an agent's overall utility is a sum over utility functions defined on each of that agent's partitions.

Definition 8.4.7. *Consider a game* $G = (\mathcal{N}, \{S_p\}_{p\in\mathcal{N}}, \{u^p\}_{p\in\mathcal{N}})$. *For* $p = 1,\ldots,n$, *let* $P_p = \{P_{p,1},\ldots,P_{p,t_p}\}$, *where* $P_{p,q} = \{C^1_{p,q}\ldots C^{r_{pq}}_{p,q}\}$ *is a partition of* S_{-p} *into* r_{pq}

classes. The set $\mathscr{P} = \{P_1, \ldots, P_n\}$ *is a* linear reduced form *of* G *if for each* p *there exist* $u^{p,1}, \ldots, u^{p,t_p} \in \mathbb{R}^M$ *such that for all* s, $u^p_s = \sum_q u^{p,q}_s$, *and for each* $q \leq t_p$, $u^{p,q}_s = u^{p,q}_{s'}$ *whenever (1)* $s_p = s'_p$ *and (2) both* s_{-p} *and* s'_{-p} *belong to the same class in* $P_{p,q}$. *The* size *of a reduced form is the number of classes in the partitions plus the bits required to specify a number for each tuple* (p,q,k,ℓ) *where* $1 \leq p \leq n$, $1 \leq q \leq t_p$, $1 \leq k \leq r_{pq}$ *and* $\ell \in S_p$.

We write $u^{p,q}_{(k,\ell)}$ for the value corresponding to tuple (p,q,k,ℓ), and for $\mathbf{k} = (k_1, \ldots, k_{t_p})$ we write $u^p_{(\mathbf{k},\ell)} \equiv \sum_q u^{p,q}_{(k_q,\ell)}$.

Example 8.4.8 (polymatrix games)**.** *Recall from Section 2.1.1 that in a polymatrix game, each player's utility is the sum of utilities resulting from her bilateral interactions with each of the* $n-1$ *other players:* $u^p_s = \sum_{p' \neq p} e^T_{s_p} A^{pp'} e_{s_{p'}}$ *where* $A^{pp'} \in \mathbb{R}^{|S_p| \times |S_{p'}|}$ *and* $e_{s_p} \in \mathbb{R}^{|S_p|}$ *is the unit vector corresponding to* s_p. *The utility functions of such a representation require only* $\sum_{p,p' \in \mathscr{N}} |S_p| \times |S_{p'}|$ *values to specify. Polymatrix games do not have a concise reduced-form encoding, but can easily be written as linear-reduced-form games. Essentially, we create one partition for every matrix game that an agent plays, with each class differing in the action played by the other agent who participates in that matrix game, and containing all the strategy profiles that can be adopted by all of the other players. Formally, given a polymatrix game, we construct its linear reduced form with* $P_p = \{P_{p,q}\}_{q \in \mathscr{N} \setminus \{p\}}$, *and* $P_{p,q} = \{C^\ell_{p,q}\}_{\ell \in S_q}$ *with* $C^\ell_{p,q} = \{s_{-p} | s_q = \ell\}$. □

Most of the results in Section 8.4.1 straightforwardly translate to linear reduced forms.

Lemma 8.4.9. *Let* $\mathscr{P}$ *be a linear reduced form for game* G. *Then for all* $y \in \mathbb{R}^N$, $v \in \mathbb{R}^n$, *for all players* p, *there exist* $\hat{u}^{p,1}(y,v), \ldots, \hat{u}^{p,t_p}(y,v) \in \mathbb{R}^M$ *such that the weighted deviation-adjusted utilities* $\hat{u}^p(y,v) = \sum_q \hat{u}^{p,q}(y,v)$, *and for all* $q \leq t_p$, $s_p \in S_p$ *and* $s_{-p}, s'_{-p} \in S_{-p}$, *if* s_{-p} *and* s'_{-p} *are in the same class in* $P_{p,q}$, *then* $\hat{u}^{p,q}_{s_p,s_{-p}}(y,v) = \hat{u}^{p,q}_{s_p,s'_{-p}}(y,v)$.

Write the weighted deviation-adjusted utility for player p, *her pure strategy* $\ell \in S_p$ *and classes* $C^{k_1}_{p,1}, \ldots, C^{k_{t_p}}_{p,t_p}$ *as* $\hat{u}^p_{(\mathbf{k},\ell)}(y,v)$ *where* $\mathbf{k} = (k_1, \ldots, k_{t_p})$. *Furthermore, we have*

$$\hat{u}^p_{(\mathbf{k},\ell)}(y,v) \equiv v_p u^p_{(\mathbf{k},\ell)} + \sum_{j \in S_p} y^p_{\ell,j} (u^p_{(\mathbf{k},\ell)} - u^p_{(\mathbf{k},j)}).$$

Corollary 8.4.10. *For a linear-reduced-form-based representation, if there exists a polynomial-time algorithm for the optimal social welfare problem, then the optimal social welfare CE problem and the max-min welfare CE problem can be solved in polynomial time.*

Graphical Polymatrix Games

A polymatrix game may have graphical-game-like structure: player p's utility may depend only on a subset of the other player's actions. In terms of utility functions, this corresponds to $A^{pp'} = 0$ for certain pairs of players p, p'. As with graphical games, we can construct the (undirected) graph $G = (\mathcal{N}, E)$ where there is an edge $\{p, p'\} \in E$ if $A^{pp'} \neq 0$ or$A^{p'p} \neq 0$. We call such a game a graphical polymatrix game. This can also be understood as a graphical game where each player p's utility is the sum of bilateral interactions with her neighbors.

A tree polymatrix game is a graphical polymatrix game whose corresponding graph is a tree. Consider the optimal CE problem on tree polymatrix games. Since such a game is also a tree graphical game, Papadimitriou and Roughgarden [2008]'s optimal CE algorithm for tree graphical games can be applied. However, this algorithm does not run in polynomial time, because the representation size of tree polymatrix games can be exponentially smaller than that of the corresponding graphical game (which grows exponentially in the degree of the graph). However, we can give a different polynomial-time algorithm for this problem.

Theorem 8.4.11. *Optimal CE in tree polymatrix games can be computed in polynomial time.*

Proof. It is sufficient to give an algorithm for the deviation-adjusted social welfare problem. Using an argument similar to that given in Example 8.4.8, tree polymatrix games have a natural linear reduced form, and it is straightforward to verify that tree polymatrix games are a linear-reduced-form-based representation. By Corollary 8.4.10 it is sufficient to construct an algorithm for the optimal social welfare problem.

Let N_p be the set of players in the subtree rooted at p. Suppose p's parent in the tree is q. Let the *social welfare contribution* of N_p be the social welfare of players in N_p minus $e_{s_p}^T A^{pq} e_{s_q}$. Let the social welfare contribution of the root player be the

social welfare of $\mathscr{N}$. Then the social welfare contribution of N_p depends solely on the pure strategy profile restricted to N_p.

The following dynamic programming algorithm solves the optimal social welfare problem in polynomial time. We go from the leaves to the root of the tree. Each child q of p passes to its parent the message $\{w^{N_q,s_q}\}_{s_q \in S_q}$, where w^{N_q,s_q} is the optimal social welfare contribution of N_q provided that q plays s_q. Given the messages from all of $p's$ children $q_1,\ldots,q_k$, we can compute the message of p as follows: for each $s_p \in S_p$,

$$\begin{aligned} w^{N_p,s_p} &= \max_{s_{q_1},\ldots,s_{q_k}} \sum_{j=1}^{k} \left[w^{N_{q_j},s_{q_j}} + e_{s_p}^T A^{p,q_j} e_{s_{q_j}} \right] \\ &= \sum_{j=1}^{k} \max_{s_{q_j}} \left[w^{N_{q_j},s_{q_j}} + e_{s_p}^T A^{p,q_j} e_{s_{q_j}} \right]. \end{aligned}$$

The second equality is due to the fact that the j-th summand depends only on s_{q_j}. It is straightforward to verify that the optimal social welfare is $\max_{s_r} w^{N_r,s_r}$ where r is the root player, and that the algorithm runs in polynomial time. The corresponding optimal pure strategy profile can be constructed by going from the root to the leaves. □

This algorithm can be straightforwardly extended to yield a polynomial-time algorithm for optimal CE in graphical polymatrix games with constant treewidth, for hypergraphical games [Papadimitriou and Roughgarden, 2008] on acyclic hypergraphs, and more generally for hypergraphs with constant hypertree-width.

8.4.3 Representations with Action-Specific Structure

The above results for reduced forms and linear reduced forms crucially depend on the fact that the partitions (i.e., the structure of the utility functions) depend on p but do not depend on the action chosen by player p. There are representations whose utility functions have action-dependent structure, including congestion games [Rosenthal, 1973], local effect games [Leyton-Brown and Tennenholtz, 2003], and action-graph games [Jiang et al., 2011]. For such representations, we can define a variant of the reduced form that has action-dependent partitions. For

example:

Definition 8.4.12. *Consider a game* $G = (\mathcal{N}, \{S_p\}_{p\in\mathcal{N}}, \{u^p\}_{p\in\mathcal{N}})$. *For* $p = 1,\ldots,n$, $\ell \in S_p$, *let* $P_{p,\ell} = \{P_{p,\ell,1},\ldots,P_{p,\ell,t_{p\ell}}\}$, *where* $P_{p,\ell,q} = \{C^1_{p,\ell,q}\ldots C^{r_{p\ell q}}_{p,\ell,q}\}$ *is a partition of* S_{-p} *into* $r_{p\ell q}$ *classes. The set* $\mathscr{P} = \{P_{p,\ell}\}_{p\in\mathcal{N},\ell\in S_p}$ *is a* action-specific linear reduced form *of G if for each* p,ℓ *there exist* $u^{p,\ell,1},\ldots,u^{p,\ell,t_{p\ell}} \in \mathbb{R}^M$ *such that for each* $p \in \mathcal{N}$, $\ell \in S_p$, *and* $q \le t_p$,

1. *for all* $s_{-p} \in S_{-p}$, $u^p_{\ell s_{-p}} = \sum_q u^{p,\ell,q}_{\ell s_{-p}}$;
2. $u^{p,\ell,q}_{\ell s_{-p}} = u^{p,\ell,q}_{\ell s'_{-p}}$ *whenever both* s_{-p} *and* s'_{-p} *belong to the same class in* $P_{p,\ell,q}$.

The size *of a reduced form is the number of classes in the partitions plus the bits required to specify a number for each tuple* (p,q,k,ℓ) *where* $1 \le p \le n$, $1 \le q \le t_{p\ell}$, $1 \le k \le r_{p\ell q}$ *and* $\ell \in S_p$.

However, unlike both the reduced form and linear reduced form, the weighted deviation-adjusted utilities no longer satisfy the same partition structure as the utilities. Intuitively, the weighted deviation-adjusted utility at s has contributions from the utilities of the strategy profiles when player p deviates to different actions. Whereas for linear reduced forms these deviated strategy profiles correspond to the same class as s in the partition, we now consider different partitions for each action to which p deviates. As a result the weighted deviation-adjusted social welfare problem has a more complex form that the optimal social welfare problem.

Singleton Congestion Games

As mentioned in Chapters 2 and 4, Ieong et al. [2005] studies a class of games called singleton congestion games and showed that the optimal PSNE can be computed in polynomial time. Such a game can be formulated as an instance of congestion games where each action contains a single resource, or an instance of symmetric AGGs where the only edges are self edges.

Formally, a singleton congestion game is specified by $(\mathcal{N}, \mathscr{A}, \{f^\alpha\}_{\alpha\in\mathscr{A}})$ where $\mathcal{N} = 1,\ldots,n$ is the set of players, $\mathscr{A}$ the set of actions, and for each action $\alpha \in \mathscr{A}$, $f^\alpha : [n] \to \mathbb{R}$. The game is symmetric; each player's set of actions $S_p \equiv \mathscr{A}$. Each strategy profile s induces an action count $c(\alpha) = |\{p|s_p = \alpha\}|$ on each α: the

number of players playing action α. Then the utility of a player that chose α is $f^{\alpha}(c(\alpha))$. The representation requires $O(|\mathscr{A}|n)$ numbers to specify.

We now show that the optimal social welfare CCE problem can be computed in polynomial time for singleton congestion games. Before attacking the problem, we first note that the optimal social welfare problem can be solved in polynomial time by a relatively straightforward dynamic-programming algorithm which is a simplified version of Ieong et al. [2005]'s algorithm for optimal PSNE in singleton congestion games. First observe that the social welfare of a strategy profile can be written in terms of the action counts:

$$w_s = \sum_{\alpha} c(\alpha) f^{\alpha}(c(\alpha)).$$

The optimal social welfare problem is equivalent to finding a vector of action counts that sums to n and maximizes the above expression. The social welfare can be further decomposed into contributions from each action α. The dynamic-programming algorithm starts with a single action and adds one action at a time until all actions are added. At each iteration, it maintains a set of tuples $\{(n', w^{n'})\}_{1 \leq n' \leq n}$, specifying that the best social welfare contribution from the current set of actions is $w^{n'}$ when exactly n' players chose actions in the current set.

Consider the optimal social welfare CCE problem. Can we leverage the algorithm for the optimal social welfare problem to solve the coarse deviation-adjusted social welfare problem? Our task here is slightly more complicated: in general the coarse deviation-adjusted social welfare problem no longer has the same symmetric structure due to the fact that y can be asymmetric. However, when y is player-symmetric (that is, $y_j^p = y_j^{p'}$ for all pairs of players (p, p')), then we recover symmetric structure.

Lemma 8.4.13. *Given a singleton congestion game and player-symmetric input y, the coarse deviation-adjusted social welfare problem can be solved in polynomial time.*

Proof. The coarse deviation-adjusted social welfare can be written as

$$\begin{aligned}\tilde{w}_s(y) &= \sum_p u_s^p(1+\sum_{j\neq s_p} y_j^p) - \sum_p \sum_{j\neq s_p} y_j^p u_{js_{-p}}^p \\ &= \sum_{\alpha\in\mathscr{A}} \left[c(\alpha) f^\alpha(c(\alpha)) \left(1+\sum_{j\neq\alpha} y_j^p\right) - (n-c(\alpha)) f^\alpha(c(\alpha)+1) y_\alpha^p \right].\end{aligned}$$

The contribution from each action α depends only on $c(\alpha)$. Therefore, using a similar dynamic-programming algorithm as above we can solve the coarse deviation-adjusted social welfare problem in polynomial time. □

Therefore if we can guarantee that during a run of ellipsoid method for (8.3.3) all input queries y to the separation oracle are symmetric, then we can apply Lemma 8.4.13 to solve the problem in polynomial time. We observe that for any symmetric game, there must exist a *symmetric* CE that optimizes the social welfare. This is because given an optimal CE we can create a mixture of permuted versions of this CE, which must itself be a CE by convexity, and must also achieve the same social welfare by symmetry. However, this argument in itself does not guarantee that the y we obtain by the method above will be symmetric. Instead, we observe that if we solve (8.3.3) using a ellipsoid method with a player-symmetric initial ball, and use a separation oracle that returns a player-symmetric cutting plane, then the query points y will be player-symmetric. We are able to construct such a separation oracle using a symmetrization argument.

Theorem 8.4.14. *Given a singleton congestion game, the optimal social welfare CCE can be computed in polynomial time.*

Proof. As argued in Section 8.4.3, it is sufficient to construct a separation oracle for (8.3.3) that returns a player-symmetric cutting plane. The cutting plane corresponding to a pure strategy profile solution s of the coarse deviation-adjusted social welfare problem is not player-symmetric in general; but we can symmetrize it by constructing a mixture of permutations of s. Since by symmetry each permuted version of s correspond to a violated constraint, the resulting cutting plane is still correct and is symmetric. Enumerating all permutations over players would be exponential, but it turns out that for our purposes it is sufficient to use a small set of

permutations.

Formally, let π_i be the permutation over the set of players $\mathscr{N}$ that maps each p to $p+i \mod n$. Then the set of permutations $\{\pi_i\}_{0\le i\le n-1}$ corresponds to the cyclic group.

Suppose s is a solution of the coarse deviation-adjusted social welfare problem with symmetric input y. The corresponding cut (violated constraint) is $(C_s)^T y + w_s \le t$. Recall that the (p,j)-th entry of C_s is $C_s^{p,j} = (u_s^p - u^p_{js_{-p}})$. For a permutation π over $\mathscr{N}$, write s^π the permuted profile induced by π, i.e. $s^\pi = (s_{\pi(1)}, \ldots, s_{\pi(n)})$. Then s^π is also a solution of the coarse deviation-adjusted social welfare problem. Form the following convex combination of n of the constraints of (8.3.3):

$$\frac{1}{n}\sum_{i=0}^{n-1}\left[(C_{s^{\pi_i}})^T y + w_{s^{\pi_i}}\right] \le t$$

The left-hand side can be simplified to $w_s + (\overline{C}_s)^T y$ where $\overline{C}_s = \frac{1}{n}\sum_{i=0}^{n-1} C_{s^{\pi_i}}$. We claim that this cutting plane is player-symmetric, meaning $\overline{C}_s^{p,j} = \overline{C}_s^{p',j}$ for all pairs of players p, p' and all $j \in \mathscr{A}$. This is because

$$\begin{aligned}\overline{C}_s^{p,j} &= \frac{1}{n}\sum_{i=0}^{n-1} C_{s^{\pi_i}}^{p,j} = \frac{1}{n}\sum_{i=0}^{n-1}(u^p_{s^{\pi_i}} - u^p_{js^{\pi_i}_{-p}}) \\ &= \frac{1}{n}\left[\sum_{\alpha\ne j} c(\alpha) f^\alpha(c(\alpha)) - (n-c(j)) f^j(c(j)+1)\right] = \overline{C}_s^{p',j}.\end{aligned}$$

This concludes the proof. □

Our approach for singleton congestion games crucially depends on the fact that the coarse deviation profile y_j^p does not care which action it is deviating from. This allowed us to (in the proof of Lemma 8.4.13) decompose the coarse deviation-adjusted social welfare into terms that only depend on the action count on one action. The same approach cannot be directly applied to solve the optimal CE problem, because then the deviation profile would give a different y_{ij}^p for each action i that p deviates from, and the resulting expression for deviation-adjusted social welfare would involve summands that depend on the action counts on pairs of actions.

An interesting future direction is to explore whether our approach for singleton congestion games can be generalized to other classes of symmetric games, such as symmetric AGGs with bounded treewidth.

8.5 Conclusion and Open Problems

We have proposed an algorithmic approach for solving the optimal correlated equilibrium problem in succinctly represented games, substantially extending a previous approach due to Papadimitriou and Roughgarden [2008]. In particular, we showed that the optimal CE problem is tractable when the *deviation-adjusted social welfare problem* can be solved in polynomial time. We generalized the reduced forms of Papadimitriou and Roughgarden [2008] to show that if a representation can be characterized by "linear reduced forms", i.e. player-specific linear functions over partitions, then for that representation, the deviation-adjusted social welfare problem can be reduced to the optimal social welfare problem. Leveraging this result, we showed that the optimal CE problem is tractable in graphical polymatrix games on tree graphs. We also considered the problem of computing the optimal *coarse correlated equilibrium*, and derived a similar sufficient condition. We used this condition to prove that the optimal CCE problem is tractable for singleton congestion games.

Our work points the way to a variety of open problems, which we briefly summarize here.

Price of Anarchy. Our results imply that for compactly represented games with polynomial-time algorithms for the optimal social welfare problem and the weighted deviation-adjusted social welfare problem, the Price of Anarchy (POA) for correlated equilibria (i.e., the ratio of social welfare under the best outcome and the worst correlated equilibrium) can be computed in polynomial time. Similarly for the Price of Total Anarchy (i.e., the ratio of social welfare under the best outcome and the worst coarse correlated equilibrium). There is an extensive literature on proving bounds on the POA for various solution concepts and for various classes of games. One line of research that is particularly relevant to our work is the "smoothness bounds" method pioneered by Roughgarden [2009]. In particular, that work showed that if a certain smoothness relation can be shown to hold for a

class of games, then it can be used to prove an upper bound on POA for these games that holds for many solution concepts including pure and mixed NE, CE and CCE. More recently, Nadav and Roughgarden [2010] gave a primal-dual LP formulation for proving POA bounds and showed that finding the best smoothness coefficients corresponds to the dual of the LP for the POA for average coarse correlated equilibrium (ACCE), a weaker solution concept than CCE. The primal-dual LP formulation of Nadav and Roughgarden [2010] and our LPs (P) and (D) are equivalent up to scaling; however whereas Nadav and Roughgarden [2010] focused on the task of proving POA upper bounds for classes of games, here we focus on computing the optimal CE / CCE and POA for individual games. One interesting direction is to use our algorithms together with an game instance generator to automatically find game instances with large POA, thus improving the lower bounds on POA for given classes of games.

Complexity separations. We have shown that for singleton congestion games, the optimal social welfare problem and the optimal CCE problem are tractable while the complexity of the optimal CE problem is unknown. An open problem is to prove a separation of the complexities of these problems for singleton congestion games or for another class. Another related problem is the optimal PSNE problem, which can be thought of as the optimal CE problem plus integer constraints on x. We do not know the exact relationship between the optimal PSNE problem and the other problems. For example the optimal PSNE problem is known to be tractable for singleton congestion games [Ieong et al., 2005] while we do not know how to solve the optimal CE problem. On the other hand for tree polymatrix games we showed the CE problem is in polynomial time, while the complexity of the PSNE problem is unknown.

Necessary condition for tractability. Another open question is the following: is tractability of the deviation-adjusted social welfare problem a *necessary* condition for tractability of the optimal CE problem? We know (e.g., from Grötschel et al. [1988]) that the separation oracle problem for the dual LP (D) is equivalent to the problem of optimizing an arbitrary linear objective on the feasible set of (D). However this in itself is not enough to prove equivalence of the deviation-adjusted social welfare problem and the optimal CE problem. First of all the separation oracle problem is more general: it allows cutting planes other than constraints cor-

responding to pure strategy profiles. Furthermore, (D) has a particular objective, but optimizing an arbitrary linear objective means allowing the objective to depend on y as well as t. If we take the dual of such an LP with (e.g.) objective $r^T y + t$ for some vector $r \in \mathbb{R}^N$, we get a generalized version of the optimal CE problem, with constraints $Ux \geq r$ instead of $Ux \geq 0$.

Relaxations and approximations. Another interesting direction worth exploring is relaxations of the incentive constraints of these problems, either as hard bounds or as soft constraints that add penalties to the objective, as well as the problem of approximating the optimal CE. For these problems we can define corresponding variants of the deviation-adjusted social welfare problem as sufficient conditions, but it remains to be seen whether one can prove concrete results, e.g., for approximating optimal CE for specific representations for which the exact optimal CE problem is hard.

Communication complexity of uncoupled dynamics. Hart and Mansour [2010] considered a setting in which each player is informed only about her own utility function, and analyzed the communication complexity for so-called *uncoupled* dynamics to reach various kinds of equilibrium. They used a straightforward adaptation of Papadimitriou and Roughgarden [2008]'s algorithm for a sample CE to show that a CE can be reached using polynomial amount of communication. We can consider the question of reaching an optimal CE by uncoupled dynamics. Our approach can be straightforwardly adapted to this setting, reducing the problem to finding a communication protocol for the uncoupled version of the deviation-adjusted social welfare problem in which each player knows only her own utility function.

Proposition 8.5.1. *If there is a polynomial communication protocol for the uncoupled deviation-adjusted social welfare problem, then there is a polynomial communication protocol for the optimal CE problem.*

At a high level, the protocol has a center running the ellipsoid method on (D), using the communication protocol for the uncoupled deviation-adjusted social welfare problem as a separation oracle. An open problem is whether there exist more "natural" types of dynamics that converge to optimal CE. For example, there is extensive literature on no-internal-regret learning dynamics that converges to the

set of approximate CE in a polynomial number of steps. Can such dynamics be modified to yield optimal CE?

Bibliography

S. Adlakha, R. Johari, and G. Y. Weintraub. Equilibria of dynamic games with many players: Existence, approximation, and market structure. *CoRR*, abs/1011.5537, 2010.

B. Adsul, J. Garg, R. Mehta, and M. A. Sohoni. Rank-1 bimatrix games: a homeomorphism and a polynomial time algorithm. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 195–204, 2011.

C. Alvarez, J. Gabarro, and M. Serna. Pure Nash equilibria in a game with large number of actions. In *Mathematical Foundations of Computer Science*, 2005.

R. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67–96, 1974.

R. Aumann. Correlated equilibrium as an expression of Bayesian rationality. *Econometrica: Journal of the Econometric Society*, pages 1–18, 1987.

D. Avis, G. Rosenberg, R. Savani, and B. von Stengel. Enumeration of nash equilibria for two-player games. *Economic Theory*, 42:9–37, 2010. ISSN 0938-2259. URL http://dx.doi.org/10.1007/s00199-009-0449-x. 10.1007/s00199-009-0449-x.

E. Ben-Sasson, A. Kalai, and E. Kalai. An approach to bounded rationality. In *NIPS: Proceedings of the Neural Information Processing Systems Conference*, pages 145–152, 2006.

N. Bhat and K. Leyton-Brown. Computing Nash equilibria of action-graph games. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 35–42, 2004.

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

B. Blum, C. Shelton, and D. Koller. A continuation method for Nash equilibria in structured games. *JAIR: Journal of Artificial Intelligence Research*, 25: 457–502, 2006.

H. Bodlaender. Treewidth: Algorithmic techniques and results. In *Mathematical Foundations of Computer Science*, pages 19–36. Springer Berlin / Heidelberg, 1997. ISBN 978-3-540-63437-9. URL http://dx.doi.org/10.1007/BFb0029946.

H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. 25(6):1305–1317, 1996. ISSN 00975397. URL http://dx.doi.org/doi/10.1137/S0097539793251219.

H. L. Bodlaender. Treewidth: structure and algorithms. In *Proceedings of the 14th international conference on Structural information and communication complexity*, SIROCCO'07, pages 11–25, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72918-1.

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *UAI*, pages 115–123, 1996.

F. Brandt, F. Fischer, and M. Holzer. Symmetries and the complexity of pure Nash equilibrium. *Journal of Computer and System Sciences*, 75(3):163–177, 2009.

G. Brown. Iterative solutions of games by fictitious play. In T. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, New York, 1951.

X. Chen and X. Deng. Settling the complexity of 2-player Nash-equilibrium. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 261–272, 2006.

V. Conitzer and T. Sandholm. Complexity of (iterated) dominance. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, 2005.

V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, 2006.

V. Conitzer and T. Sandholm. New complexity results about nash equilibria. *Games and Economic Behavior*, 63(2):621 – 641, 2008. ISSN 0899-8256. Second World Congress of the Game Theory Society.

G. Dantzig and M. Thapa. *Linear Programming 2: Theory and Extensions*. Springer, 2003.

A. Darwiche. Constant-space reasoning in dynamic Bayesian networks. *International Journal of Approximate Reasoning*, 26(3):161–178, 2001.

C. Daskalakis and C. Papadimitriou. The complexity of games on highly regular graphs. In *the 13th Annual European Symposium on Algorithms*, 2005.

C. Daskalakis and C. Papadimitriou. Computing pure Nash equilibria via Markov random fields. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 91–99, 2006.

C. Daskalakis and C. Papadimitriou. Computing equilibria in anonymous games. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 83–93, 2007.

C. Daskalakis and C. Papadimitriou. Discretized multinomial distributions and nash equilibria in anonymous games. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, 2008.

C. Daskalakis and C. Papadimitriou. On oblivious PTAS's for Nash equilibrium. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 75–84. ACM New York, NY, USA, 2009.

C. Daskalakis, A. Fabrikant, and C. Papadimitriou. The game world is flat: The complexity of Nash equilibria in succinct games. In *ICALP: Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 513–524, 2006a.

C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 71–78, 2006b.

C. Daskalakis, G. Schoenebeck, G. Valiant, and P. Valiant. On the complexity of Nash equilibria of Action-Graph Games. In *SODA: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 710–719, 2009.

T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

E. Elkind, L. Goldberg, and P. Goldberg. Nash equilibria in graphical games on trees revisited. *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 100–109, 2006.

P. Erdős and J. L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14(3):298 – 301, 1973. ISSN 0097-3165.

K. Etessami and M. Yannakakis. On the Complexity of Nash Equilibria and Other Fixed Points (Extended Abstract). In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 113–123, 2007.

A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 604–612. ACM New York, NY, USA, 2004.

E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–499, 1962.

D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

D. Gale, H. Kuhn, and A. Tucker. On symmetric games. *Contributions to the Theory of Games*, pages 81–87, 1950.

F. Germano and G. Lugosi. Existence of sparsely supported correlated equilibria. *Economic Theory*, 32(3):575–578, 2007.

M. Goemans, V. Mirrokni, and A. Vetta. Sink equilibria and convergence. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 142–154, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2468-0. URL http://dx.doi.org/10.1109/SFCS.2005.68.

P. W. Goldberg and C. H. Papadimitriou. Reducibility among equilibrium problems. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 61–70, 2006.

G. Gottlob, G. Greco, and F. Scarcello. Pure Nash equilibria: Hard and easy games. *Journal of Artificial Intelligence Research*, 24:357–406, 2005.

G. Gottlob, G. Greco, and T. Mancini. Complexity of pure equilibria in Bayesian games. In *IJCAI: Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1294–1299, 2007.

S. Govindan and R. Wilson. Structure theorems for game trees. *Proceedings of the National Academy of Sciences*, 99(13):9077–9080, 2002.

S. Govindan and R. Wilson. A global Newton method to compute Nash equilibria. *Journal of Economic Theory*, 110:65–86, 2003.

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

M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, New York, NY, 1988.

J. Hannan. Approximation to Bayes risk in repeated plays. In M. Dresher, A. Tucker, and P. Wolfe, editors, *Contributions to the Theory of Games*, volume 3, pages 97–139. Princeton University Press, 1957.

J. Harsanyi. Games with incomplete information played by "Bayesian" players, i-iii. part i. the basic model. *Management science*, 14(3):159–182, 1967.

S. Hart and Y. Mansour. How long to equilibrium? the communication complexity of uncoupled equilibrium procedures. *Games and Economic Behavior*, 69(1):107–126, 2010. ISSN 0899-8256.

S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5), 2000.

S. Hart and D. Schmeidler. Existence of correlated equilibria. *Mathematics of Operations Research*, 14(1):18–25, 1989.

D. Heckerman and J. S. Breese. Causal independence for probability assessment and inference using Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics*, 26(6):826–831, 1996.

P. Herings and R. Peeters. A globally convergent algorithm to compute all Nash equilibria for n-Person games. *Annals of Operations Research*, 137(1): 349–368, 2005.

P. Herings and R. Peeters. Homotopy methods to compute equilibria in game theory. *Economic Theory*, pages 1–38, 2009.

H. Hotelling. Stability in competition. *Economic Journal*, 39:41–57, 1929.

J. Howson Jr. Equilibria of polymatrix games. *Management Science*, pages 312–318, 1972.

J. Howson Jr and R. Rosenthal. Bayesian equilibria of finite two-person games with incomplete information. *Management Science*, pages 313–315, 1974.

W. Huang. *Equilibrium Computation for ExtensiveGames*. PhD thesis, London School of Economics and Political Science, 2011.

W. Huang and B. Von Stengel. Computing an extensive-form correlated equilibrium in polynomial time. In *WINE: Proceedings of the Workshop on Internet and Network Economics*, pages 506–513, 2008.

S. Ieong, R. McGrew, E. Nudelman, Y. Shoham, and Q. Sun. Fast and compact: A simple class of congestion games. *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, pages 489–494, 2005.

K. Iyer, R. Johari, and M. Sundararajan. Mean field equilibria of dynamic auctions with learning. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 339–340, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0261-6. URL http://doi.acm.org/10.1145/1993574.1993631.

A. Jiang and K. Leyton-Brown. Polynomial computation of exact correlated equilibrium in compact games. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, 2011. `http://arxiv.org/abs/1011.0253`.

A. X. Jiang. Computational problems in multiagent systems. Master's thesis, University of British Columbia, 2006.

A. X. Jiang and K. Leyton-Brown. A polynomial-time algorithm for Action-Graph Games. In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, pages 679–684, 2006.

A. X. Jiang and K. Leyton-Brown. Computing pure Nash equilibria in symmetric Action-Graph Games. In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, pages 79–85, 2007a.

A. X. Jiang and K. Leyton-Brown. A tutorial on the proof of the existence of Nash equilibria. Technical Report TR-2007-25, University of British Columbia, Department of Computer Science, November 2007b.

A. X. Jiang and K. Leyton-Brown. Bayesian action-graph games. In *NIPS: Proceedings of the Neural Information Processing Systems Conference*, 2010.

A. X. Jiang and M. Safari. Pure Nash equilibria: Complete characterization of hard and easy graphial games. In *AAMAS: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2010.

A. X. Jiang, A. Pfeffer, and K. Leyton-Brown. Temporal Action-Graph Games: A new representation for dynamic games. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2009.

A. X. Jiang, K. Leyton-Brown, and N. Bhat. Action-graph games. *Games and Economic Behavior*, 71(1):141–173, January 2011.

S. Kakade, M. Kearns, J. Langford, and L. Ortiz. Correlated equilibria in graphical games. In *EC: Proceedings of the ACM Conference on Electronic*

Commerce, pages 42–47, New York, NY, USA, 2003. ACM. ISBN 1-58113-679-X. URL http://doi.acm.org/10.1145/779928.779934.

E. Kalai. Large robust games. *Econometrica*, 72(6):1631–1665, 2004.

E. Kalai. Partially-specified large games. In *WINE: Proceedings of the Workshop on Internet and Network Economics*, pages 3–13, 2005.

H. Kamisetty, E. P. Xing, and C. J. Langmead. Approximating correlated equilibria using relaxations on the marginal polytope. In *ICML*, 2011.

R. Kannan and T. Theobald. Games of fixed rank: A hierarchy of bimatrix games. *Economic Theory*, pages 1–17, 2009.

M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

P. Klingsberg. A Gray code for compositions. *Journal of Algorithms*, 3:41–44, 1982.

T. Kloks. *Treewidth: Computations and Approximations*. Springer-Verlag, Berlin, 1994.

D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. In *IJCAI: Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221, 2003.

D. Koller, N. Megiddo, and B. Von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2): 247–259, 1996.

H. Kuhn. Extensive games and the problem of information. In H. Kuhn and A. Tucker, editors, *Contributions to the Theory of Games*, volume II, pages 193–216, 1953.

C. Lemke and J. Howson. Equilibrium points of bimatrix games. *Society for Industrial and Applied Mathematics Journal of Applied Mathematics*, 12: 413–423, 1964.

C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11(7):681–689, May 1965.

K. Leyton-Brown and M. Tennenholtz. Local-effect games. In *IJCAI: Proceedings of the International Joint Conference on Artificial Intelligence*, pages 772–780, 2003.

R. Lipton, E. Markakis, and A. Mehta. Playing large games using simple strategies. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 36–41. ACM New York, NY, USA, 2003.

G. B. D. M. Benisch and T. Sandholm. Algorithms for closed under rational behavior (curb) sets. *Journal of Artificial Intelligence Research*, 38:513–534, 2010.

O. Mangasarian. Equilibrium points in bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):778–780, 1964.

R. McKelvey and A. McLennan. Computation of equilibria in finite games. *Handbook of Computational Economics*, 1:87–142, 1996.

R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software tools for game theory, 2006. http://econweb.tamu.edu/gambit.

B. Milch and D. Koller. Ignorable information in multi-agent scenarios. Technical Report MIT-CSAIL-TR-2008-029, MIT, 2008.

I. Milchtaich. Congestion games with player-specific payoff functions. *Games and Economic Behavior*, 13:111–124, 1996.

D. Monderer. Multipotential games. In *IJCAI: Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1422–1427, 2007.

D. Monderer and L. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2002.

K. Murphy. Bayes Net Toolbox for Matlab. `http://bnt.sourceforge.net`, 2007.

R. Myerson. Dual reduction and elementary games. *Games and Economic Behavior*, 21(1-2):183–202, 1997.

U. Nadav and T. Roughgarden. The limits of smoothness: A primal-dual framework for Price of Anarchy bounds. In *WINE: Proceedings of the Workshop on Internet and Network Economics*, 2010.

J. F. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

R. Nau and K. McCardle. Coherent behavior in noncooperative games. *Journal of Economic Theory*, 50(2):424–444, 1990.

D. Nilsson and S. Lauritzen. Evaluating influence diagrams using LIMIDs. In *UAI*, pages 436–445, 2000.

N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, Cambridge, UK, 2007.

E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *AAMAS: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 880–887, 2004.

F. A. Oliehoek, M. T. J. Spaan, J. Dibangoye, and C. Amato. Heuristic search for identical payoff bayesian games. In *AAMAS: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1115–1122, May 2010.

L. Ortiz and M. Kearns. Nash propagation for loopy graphical games. In *NIPS: Proceedings of the Neural Information Processing Systems Conference*, pages 817–824, 2003.

C. Papadimitriou. Computing correlated equilibria in multiplayer games. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 49–56, 2005.

C. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3):14, July 2008.

C. Papadimitriou and T. Roughgarden. Comment on "computing correlated equilibria in multi-player games", 2010. http://theory.stanford.edu/~tim/papers/comment.pdf, accessed Jan. 10, 2011.

C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48 (3):498 – 532, 1994. ISSN 0022-0000.

C. H. Papadimitriou and T. Roughgarden. Computing equilibria in multi-player games. In *SODA: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 82–91, 2005.

C. Papdimitriou. The complexity of finding Nash equilibria. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, Cambridge, UK, 2007.

P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, CA, 1988.

A. Pfeffer. *Probabilistic reasoning for complex systems*. PhD thesis, Computer Science Department, Stanford University, 2000.

D. Poole and N. Zhang. Exploiting contextual independence in probabilistic inference. *Journal of Artificial Intelligence Research*, 18:263–313, 2003.

R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

Z. Rabinovich, E. Gerding, M. Polukarov, and N. R. Jennings. Generalised fictitious play for a continuum of anonymous players. In *IJCAI: Proceedings of the International Joint Conference on Artificial Intelligence*, pages 245–250, 2009.

P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130 – 143, 1988. ISSN 0022-0000.

D. M. Reeves and M. P. Wellman. Computing best-response strategies in infinite games of incomplete information. In *UAI*, pages 470–478, 2004. ISBN 0-9749039-0-6.

N. Robertson and P. Seymour. Algorithmic aspects of tree-width. *J. Algorithms*, 7: 309–322, 1986.

R. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

T. Roughgarden. Intrinsic robustness of the Price of Anarchy. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, 2009.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd edition*. Prentice Hall, Englewood Cliffs, NJ, 2003.

C. T. Ryan, A. X. Jiang, and K. Leyton-Brown. Computing pure strategy Nash equilibria in compact symmetric games. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 63–72, 2010.

T. Sandholm, A. Gilpin, and V. Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, pages 495–501, 2005.

R. Savani and B. von Stengel. Exponentially many steps for finding a Nash equilibrium in a bimatrix game. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 258–267, 2004.

H. Scarf. The approximation of fixed points of a continuous mapping. *SIAM Journal of Applied Mathematics*, 15:1328–1343, 1967.

G. Schoenebeck and S. Vadhan. The computational complexity of Nash equilibria in concisely represented games. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 270–279, 2006.

Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2009.

S. Singh, V. Soni, and M. Wellman. Computing approximate Bayes-Nash equilibria in tree-games of incomplete information. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 81–90. ACM, 2004.

J. Spencer. *Ten lectures on the probabilistic method*. CBMS-NSF regional conference series in applied mathematics. Society for Industrial and Applied Mathematics, 1994. ISBN 9780898713251.

N. D. Stein, P. A. Parrilo, and A. Ozdaglar. Exchangeable equilibria contradict exactness of the Papadimitriou-Roughgarden algorithm, October 2010. http://arxiv.org/abs/1010.2871v1.

D. Thompson, S. Leong, and K. Leyton-Brown. Computing Nash equilibria of action-graph games via support enumeration. Working paper, 2011.

D. R. Thompson and K. Leyton-Brown. Computational analysis of perfect-information position auctions. In *EC: Proceedings of the ACM Conference on Electronic Commerce*, 2009.

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

D. Vickrey and D. Koller. Multi-agent algorithms for solving graphical games. In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, pages 345–351, 2002.

J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

B. von Stengel. Computing equilibria for two-person games. volume 3 of *Handbook of Game Theory with Economic Applications*, pages 1723 – 1759. Elsevier, 2002.

B. von Stengel and F. Forges. Extensive-form correlated equilibrium: Definition and computational complexity. *Mathematics of Operations Research*, 33(4): 1002–1022, 2008. URL http://mor.journal.informs.org/cgi/content/abstract/33/4/1002.

K. Waugh, M. Zinkevich, M. Johanson, M. Kan, D. Schnizlein, and M. Bowling. A practical use of imperfect recall. In *AAAI: Proceedings of the AAAI Conference on Artificial Intelligence*, 2009.

E. Yanovskaya. Equilibrium points in polymatrix games (in Russian). *Litovskii Matematicheskii Sbornik*, 8:381–384, 1968.

N. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *JAIR: Journal of Artificial Intelligence Research*, 5:301–328, 1996.

Appendix

Appendix A

Software

In this chapter I describe software packages implemented as part of my thesis research. Overall they can be characterized as tools for computational analysis of games using the AGG and BAGG representations. Source codes of these packages are available for download at the AGG Project website (http://agg.cs.ubc.ca).

In Section A.1 I introduce file formats used by all of these packages for describing AGG and BAGG game instances. Section A.2 describes command-line programs for finding sample (Bayes) Nash equilibria in AGGs and BAGGs. Section A.3 describes a graphical user interface for creating, editing and visualizing AGGs, and Section A.4 describes extensions of GAMUT that generate AGG instances. Finally in Section **??** I discuss software projects that are currently under development.

A.1 File Formats

These software packages can read and write a description of a game as a text file. There are two formats, one for AGGs and one for BAGGs. All packages work with the AGG format; additionally, the solvers in Section A.2 also work with the BAGG format.

A.1.1 The AGG File Format

Each representation of an AGG consists of 8 sections, separated by whitespaces. Lines with a starting "#" are treated as comments and are allowed between sections.

1. The number of players, n.
2. The number of action nodes, $|\mathscr{A}|$.
3. The number of function nodes, $|\mathscr{P}|$.
4. Size of action set for each player. This is a row of n integers: $|A_1|, |A_2|, \ldots, |A_n|$
5. Each player's action set. We have n rows; row i has $|A_i|$ integers in ascending order, which are indices of action nodes. Action nodes are indexed from 0 to $|\mathscr{A}| - 1$.
6. The Action Graph. We have $|\mathscr{A}| + |\mathscr{P}|$ nodes, indexed from 0 to $|\mathscr{A}| + |\mathscr{P}| - 1$. The function nodes are indexed after the action nodes. The graph is represented as $(|\mathscr{A}| + |\mathscr{P}|)$ neighbor lists, one list per row. Rows 0 to $|\mathscr{A}| - 1$ are for action nodes; rows $|\mathscr{A}|$ to $|\mathscr{A}| + |\mathscr{P}| - 1$ are for function nodes. In each row, the first number $|\nu|$ specifies the number of neighbors of the node. Then follows $|\nu|$ numbers, corresponding to the indices of the neighbors.

   We require that each function node has at least one neighbor, and the neighors of function nodes are action nodes. The action graph restricted to the function nodes has to be a directed acyclic graph (DAG).
7. Signatures of functions. This is $|\mathscr{P}|$ rows, each specifying the mapping f_p that maps from the configuration of the function node p's neighbors to an integer for p's "action count". Each function is specified by its "signature" consisting of an integer type, possibly followed by further parameters. Several types of mapping are implemented:

   - Types 0 to 3 require no further input:

     Type 0: Sum. The action count of a function node p is the sum of the action counts of p's neighbors.

Type 1: Existence: boolean for whether the sum of the counts of neighbors are positive.

Type 2: The index of the neighbor with the highest index that has non-zero counts, or $|\mathscr{A}| + |\mathscr{P}|$ if none applies.

Type 3: The index of the neighbor with the lowest index that has non-zero counts, or $|\mathscr{A}| + |\mathscr{P}|$ if none applies.

- Types 10 to 13 are extended versions of type 0 to 3, each requiring further parameters of an integer default value and a list of *weights*, $|\mathscr{A}|$ integers enclosed in square brackets. Each action node is thus associated with an integer weight.

  Type 10: Extended Sum. Each instance of an action in *p*'s neighborhood being chosen contributes the weight of that action to the sum. These are added to the default value.

  Type 11: Extended Existence: boolean for whether the extended sum is positive. The input default value and weights are required to be nonnegative.

  Type 12: The weight of the neighbor with the highest index that has non-zero counts, or the default value if none applies.

  Type 13: The weight of the neighbor with the lowest index that has non-zero counts, or the default value if none applies.

The following is an example of the signatures for an AGG with three action nodes and two function nodes:

```
2
10 0 [2 3 4]
```

8. The payoff function for each action node. So we have $|\mathscr{A}|$ sub-blocks of numbers. Payoff function for action α is a mapping from configurations to real numbers. Configurations are represented as a tuple of integers; the size of the tuple is the size of the neighborhood of α. Each configuration specifies

the action counts for the neighbors of α, in the same order as the neighor list of α.

The first number of each subblock specifies the type of the payoff function. There are multiple ways of representing payoff functions; we (or other people) can extend the file format by defining new types of payoff functions. We define two basic types:

Type 0: The complete representation. The set of possible configurations can be derived from the action graph. This set of configurations can also be sorted in lexicographical order. So we can just specify the payoffs without explicitly giving the configurations. So we just need to give one row of real numbers, which correspond to payoffs for the ordered set of configurations.

If action α is in multiple players' action sets (say players i and j), then it is possible that the set of possible configurations given $a_i = \alpha$ is different from the set of possible configurations given $a_j = \alpha$. In such cases, we need to specify payoffs for the union of the sets of configurations (sorted in lexicographical order).

Type 1: The mapping representation, in which we specify the configurations and the corresponding payoffs. For the payoff function of action α, first give $|\mathscr{C}^{(\alpha)}|$, the number of elements in the mapping. Then follows $|\mathscr{C}^{(\alpha)}|$ rows. In each row, first specify the configuration, which is a tuple of integers, enclosed by a pair of brackets "[" and "]", then the payoff. For example, the following specifies a payoff function of type 1, with two configurations:

```
1 2
[1 0] 2.5
[1 1] -1.2
```

A.1.2 The BAGG File Format

Each representation of a BAGG consists of the following sections, separated by whitespaces. Lines with a starting "#" are treated as comments and are allowed

between sections.

1. The number of players, n.
2. The number of action nodes, $|\mathscr{A}|$.
3. The number of function nodes, $|\mathscr{P}|$.
4. A row of n integers, specifying the number of types $|\Theta_i|$ for each player i.
5. Type distribution for each player i. The distributions are assumed to be independent. Each player i's type distribution is represented as a row of $|\Theta_i|$ real numbers, one for each type $\theta_i \in \Theta_i$, specifying $\Pr(\theta_i)$, the probability of i having type θ_i. The following example block gives the type distributions for a BAGG with two players and two types for each player.

   ```
   0.5 0.5
   0.2 0.8
   ```

6. Size of type-action set for each player's each type.
7. Type-action set for each player's each type. Each type-action set is represented as a row of integers in ascending order, which are indices of action nodes. Action nodes are indexed from 0 to $|\mathscr{A}| - 1$.
8. The action graph: same as Block 6 in the AGG format.
9. Types of functions: same as Block 7 in the AGG format.
10. Utility function for each action node: same as Block 8 in the AGG format.

A.2 Solvers for finding Nash Equilibria

The AGGSolver package is a collection of solvers that computes (Bayes) Nash equilibria given a game represented in (B)AGG format. The package is written in C++, and makes use of the GameTracer package (which implements Govindan & Wilson's GNM and IPA algorithms), and GAMBIT's implementation of the simplicial subdivision algorithm. Our black-box algorithmic approach is described in Chapter 3 for AGGs and Chapter 6 for BAGGs.

The following solvers are included:

gnm_agg takes an AGG and computes one or more Nash equilibria using Govindan & Wilson's Global Newton Method (GNM).

gnm_bagg takes a BAGG and computes one or more Bayes-Nash equilibria using the GNM algorithm.

gnm_ksym_agg takes an AGG or a symmetric BAGG and computes *k*-symmetric Nash equilibria using a modified GNM algorithm.

gnm_tracing_agg takes an AGG/BAGG and a file containing initial mixed strategy profiles, for each initial mixed strategy profile sigma run a version of the GNM algorithm that simulates the linear tracing procedure starting from sigma. Good approximate equilibria can be used as "warm starts".

ipa_agg / ipa_bagg takes an AGG/BAGG and computes an approximate Nash equilibrium using Govindan & Wilson's Iterated Polymatrix Approximation algorithm.

simpdiv takes an AGG/BAGG and computes one or more Nash/Bayes-Nash equilibria using the simplicial subdivision algorithm as implemented in GAMBIT.

The source code is available for download at http://agg.cs.ubc.ca. Detailed instructions on installation and usage of these solvers can be found in the README file included in the package, which is also available at http://agg.cs.ubc.ca/AGGSolver_README.txt.

A.3 AGG Graphical User Interface

Together with Damien Bargiacchi, we developed the AGGUI package, a graphical user interface that allows users to create and edit AGGs, read in existing AGGs, and visualize strategy profiles (e.g. Nash equilibria) as a density map the action graph (see, e.g., Figures 3.17 and 3.18 in Chapter 3). It is written in Java and runs on any platform that supports Java. It is available for download at http://agg.cs.ubc.ca/aggui.jar.

A.4 AGG Generators in GAMUT

GAMUT [Nudelman et al., 2004] is a suite of generators of game instances. We have extended GAMUT with generators of AGG instances in the AGG format. We have implemented generators for three classes of AGGs:

RandomSymmetricAGG generates symmetric AGGs on random action graphs with random utilities.

CoffeeShopGame generates instances of the Coffee Shop Game described in Chapter 3.

IceCreamGame generates instances of the Ice Cream Game described in Chapter 3.

The extended GAMUT package and documentation on these AGG generators are available for download at htt://agg.cs.ubc.ca.

A.5 Software Projects Under Development

GAMBIT [McKelvey et al., 2006] is a collection of software tools for game theoretic analysis that includes implementations of many of the existing algorithms for the normal form and the extensive form. Together with Professor Theodore Turocy, who is the main author and maintainer of GAMBIT, we are working to incorporate the AGG and BAGG representations into GAMBIT.