Approximate Analysis of Large Simulation-Based Games

by

Bryce Wiedenbeck

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2015

Doctoral Committee:

Professor Michael P. Wellman, Chair
Professor Satinder Singh Baveja
Professor Tilman Borgers
Assistant Professor Grant Schoenebeck

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Game theory offers powerful tools for reasoning about agent behavior and incentives in multi-agent systems. Traditional approaches to game-theoretic analysis require enumeration of all possible strategies and outcomes. This often constrains game models to small numbers of agents and strategies or simple closed-form payoff descriptions. Simulation-based game theory extends the reach of game-theoretic analysis through the use of agent-based modeling. In the simulation-based approach, the analyst describes an environment procedurally and then computes payoffs by simulation of agent interactions in that environment.

I use simulation-based game theory to study a model of credit network formation. Credit networks represent trust relationships in a directed graph and have been proposed as a mechanism for distributed transactions without a central currency. I explore what information is important when agents make initial decisions of whom to trust, and what sorts of networks can result from their decisions. This setting demonstrates both the value of simulation-based game theory—extending game-theoretic analysis beyond analytically tractable models—and its limitations—simulations produce prodigious amounts of data, and the number of simulations grows exponentially in the number of agents and strategies.

I propose several techniques for approximate analysis of simulation-based games with large numbers of agents and large amounts of simulation data. First, I show how bootstrap-based statistics can be used to estimate confidence bounds on the results of simulation-based game analysis. I show that bootstrap confidence intervals for regret of approximate equilibria are well-calibrated. Next, I describe deviation-preserving reduction, which approximates an environment with a large number of agents using a game model with a small number of players, and demonstrate that it outperforms previous player reductions on several measures. Finally, I employ machine learning to construct game models from sparse data sets, and provide evidence that learned game models can produce even better approximate equilibria in large games than deviation-preserving reduction.

CHAPTER 1

Introduction

The task of science is to explain and make predictions about the world. This task is principally achieved by constructing models that abstract away most of the real world's details while preserving key features of a phenomenon of interest. Such abstractions necessarily trade off the model's fidelity with its analytical tractability, forcing scientists into difficult decisions about what features to prioritize. Advances in computing power through better hardware and algorithms can move the frontier of this tradeoff, allowing scientists to rethink previous modeling decisions.

Social scientists and computer scientists often study interactions among multiple independent decision makers. In various contexts, the decision-making agents can be people, organizations, computer programs, or other entities. Game theory provides a common mathematical language that aids with abstract reasoning about such systems. Game-theoretic models work by enumerating all decisions agents can make and all possible outcomes of their joint decisions. Agents' preferences over outcomes are expressed as numerical payoffs, and agent behavior is predicted by identifying equilibria, where no agent prefers to unilaterally change its decision. This method has proven to be a powerful tool for understanding incentives in multi-agent systems.

Like all mathematical abstractions, game-theoretic models require several simplifying assumptions. In addition to the requirements for describing a game—that agents' decisions be enumerable and result in outcomes with quantifiable payoffs—equilibrium analysis generally assumes expected-utility-maximizing agent behavior. Much research has explored the ways in which real-world decision-making differs from game-theoretic rationality, and incorporating richer decision models presents an exciting avenue for future computational research. Yet even within the constraints of standard game-theoretic tools, analysts often face strong fidelity/tractability tradeoffs. Enumerating the full set of outcomes when many agents have many available options is often impractical, and even in relatively simple environments, describing payoffs for those outcomes that accurately capture agent incentives may be challenging.

This thesis contributes to a line of work that brings computational power to bear on game-theoretic problems through the use of agent-based simulations. *Simulation-based game theory* allows the modeler to describe the environment and agents' actions procedurally and rely on simulation to determine payoffs for each outcome. Sections 1.1 and 1.4 in the present chapter provide background on simulation-based game theory and lay out terminology and notation employed throughout the thesis. In Chapter 2, I introduce the credit network formation problem—a multi-agent interaction where model completeness seems strongly at odds with analytical tractability. This domain highlights some of the relative strengths of analytical and simulation-based approaches and motivates my methodological contributions. The remainder of the thesis covers new methods that improve the accuracy and rigor with which large simulation-based games are analyzed. In Chapter 3, I show how the large amounts of data in simulation-based games can be leveraged to assess statistical confidence in model predictions. In Chapters 4 and 5, I describe methods for analyzing simulation-based games with a very large number of players, first in Chapter 4 by constructing small games that approximate larger ones, and then in Chapter 5 through machine learning.

1.1 The Role-Symmetric Game Representation

I focus on games represented in *normal form*, which model each agent as selecting a strategy that governs its behavior throughout an interaction. I will often be interested in representing symmetries among agents, and therefore define as the basic unit of analysis a *role-symmetric game*, where agents are partitioned into roles, such as buyers and sellers in a market, within which individual agents are indistinguishable. This means that all members of the same role face identical incentives and have the same effects on their opponents, but roles may differ in their available strategies, their payoffs, and their influence on others' payoffs. Role-symmetric games can represent any level of player symmetry present in a normal-form game. A fully asymmetric game can be captured by a role-symmetric game where every role has one agent, while a fully symmetric game is a role-symmetric game with only one role.

Formally, a *role-symmetric game* $\Gamma = (R, \{N_r\}, \{S_r\}, u)$ consists of

- a set of *roles* R,
- the number of *players* $N_r \in \mathbb{Z}^+$ for each role $r \in R$,
- disjoint ordered sets of *strategies* S_r for each role, and

- a *utility function* u.

A *profile* is an assignment of one strategy to every player. I introduce the following notation to represent profiles in role-symmetric games. A *role profile* $\vec{s}_r = \langle c_s \rangle_{s \in S_r}$ is a vector with a count c_s of the number of players choosing each strategy $s \in S_r$, with $c_s \in \{0, \ldots, N_r\}$ and $\sum_s c_s = N_r$; the set of role profiles for role r is $\vec{S}_r$. A profile is then a vector of role profiles $\langle \vec{s}_r \rangle_{r \in R} = \left\langle \langle c_s \rangle_{s \in S_r} \right\rangle_{r \in R}$, and the set of all profiles in the game $\vec{S} = \times_{r \in R} \vec{S}_r$ is the cartesian product of the role-profile sets. The utility function $u : \vec{S} \times R \times S_r \to \mathbb{R}$ maps a profile, a role, and a strategy to a real-valued payoff. The standard tabular representation of the utility function is called a *payoff matrix*, and stores $\prod_{r \in R} |S_r| \binom{N_r + |S_r| - 2}{N_r - 1}$ payoff values.

A *mixed strategy* σ is a probability distribution over S_r that a role-r player uses to select among their strategies. To distinguish strategies from mixed strategies, I use the term *pure strategy* synonymously with the former. A *symmetric mixed strategy* $\vec{\sigma}_r$ is a mixed strategy that is played by all members of a role. A *role-symmetric mixed strategy* $\vec{\sigma} = \langle \vec{\sigma}_r \rangle_{r \in R}$ is a vector of symmetric mixed strategies, one for each role; $\vec{\Sigma}$ is the set of all such role-symmetric mixed strategies. When clear from context, I refer to a symmetric mixed strategy or role-symmetric mixed-strategy as a *mixture*. I use bracket notation to indicate indexing into vectors, so $\sigma[s]$ is the probability of strategy s in symmetric mixed strategy σ, while $\vec{\sigma}[r, s]$ denotes the probability with which role r plays strategy s in role-symmetric mixed strategy $\vec{\sigma}$. I overload u to accommodate role-symmetric mixed strategies as follows: $u(\vec{\sigma}, r, s)$ is the expected utility of a player in role r playing pure strategy s when all other players select their strategies according to $\vec{\sigma}$, and $u(\vec{\sigma}, r)$ is the expected utility to any of the players in role r following $\vec{\sigma}_r$ all when players jointly play $\vec{\sigma}$. When computing equilibria, I make use of a vector representation $\vec{u}(\vec{\sigma}) = \left\langle \langle u(\vec{\sigma}, r, s) \rangle_{s \in S_r} \right\rangle_{r \in R}$ in which the $r, s,$ component gives the expected utility to a single role-r player choosing strategy $s \in S_r$ when all opponents play $\vec{\sigma}$. I discuss the computation of $u(\vec{\sigma}, r, s)$ and $\vec{u}(\vec{\sigma})$ given $u(\vec{s})$ in Section 1.3.

Representing profiles as count vectors allows unilateral deviations to be described via arithmetic operations. Let $\hat{s}$ denote a unit vector with a 1 in the coordinate for strategy s and a 0 for all other strategies. The vector $\hat{s}$ can represent either a role profile or a game profile depending on context. The profile $\vec{s}_r - \hat{s}$ is then an $(N_r - 1)$-player role profile where one player using s has been removed from $\vec{s}_r$. Similarly, $\vec{s} - \hat{s} + \hat{t}$ gives a profile that differs from $\vec{s}$ by the deviation of one player from strategy s to strategy t. Another useful arithmetic operation on profiles is scaling by a constant the number of players choosing each strategy in a role profile: $k\vec{s}_r$. I use the $\circ$ operator to denote component-wise multiplication, so $\langle k_r \rangle_{r \in R} \circ \vec{s} = \langle k_r \vec{s}_r \rangle_{r \in R}$ is a game profile where the number of players in role $r \in R$ has been scaled by a linear factor k_r. The notation $s \in \vec{s}$ or $s \in \vec{s}_r$ indicates that strategy s

is played by at least one player in the (game or role) profile; that is, $c_s > 0$. Similarly, for mixtures, $s \in \vec{\sigma}$ or $s \in \vec{\sigma}_r$ indicates that strategy s has non-zero probability in the (symmetric or role-symmetric) mixture. I refer to a strategy s as being *in the support* of mixture $\vec{\sigma}$ when $s \in \vec{\sigma}$.

1.2 Game-Theoretic Analysis Concepts

An important concept employed throughout the thesis is *regret*, the maximum utility gain a player could achieve by deviating from a profile. Regret is always defined with respect to a game and a profile, but can optionally be restricted to a specific role, or a specific strategy; I overload the $\epsilon(\cdot)$ function to refer to all three variants. The regret of a strategy (played by role r in profile $\vec{s}$) is the maximum a player could gain by deviating from s to some strategy s', holding all other players' strategies fixed. The regret for a role (in profile $\vec{s}$) is the maximum strategy regret faced by any player in role r.[1] The regret of a profile is the maximum strategy regret faced by any player in any role. Formally,

$$\epsilon : \vec{S} \times R \times S_r \to \mathbb{R} \ldots \epsilon(\vec{s}, r, s) = \max_{s' \in S_r} u(\vec{s} - \hat{s} + \hat{s'}, r, s') - u(\vec{s}, r, s) \tag{1.1}$$

$$\epsilon : \vec{S} \times R \to \mathbb{R}. \ldots\ldots\ldots \epsilon(\vec{s}, r) = \max_{s \in \vec{s}} \epsilon(\vec{s}, r, s) \tag{1.2}$$

$$\epsilon : \vec{S} \to \mathbb{R}. \ldots\ldots\ldots\ldots\ldots \epsilon(\vec{s}) = \max_{r \in R} \epsilon(\vec{s}, r) \tag{1.3}$$

When the game is not clear from context, I will specify it as the first argument: $\epsilon(\Gamma, \vec{s}, \ldots)$. The *best response* correspondence gives the deviating strategy that produces the maximum gain. The formal definition of $\mathcal{BR}(\cdot)$ parallels $\epsilon(\cdot)$, but substitutes $\arg\max$ for $\max$:

$$\mathcal{BR} : \vec{S} \times R \times S_r \to 2^{S_r} \ldots \mathcal{BR}(\vec{s}, r, s) = \arg\max_{s' \in S_r} u(\vec{s} - \hat{s} + \hat{s'}, r, s') - u(\vec{s}, r, s) \tag{1.4}$$

$$\mathcal{BR} : \vec{S} \times R \to 2^{S_r} \times 2^{S_r} \ldots \mathcal{BR}(\vec{s}, r) = \arg\max_{s \in \vec{s}, s' \in S_r} u(\vec{s} - \hat{s} + \hat{s'}, r, s') - u(\vec{s}, r, s) \tag{1.5}$$

Note that regret can never be negative, because if $s \in \mathcal{BR}(\vec{s}, r, s)$ then $\epsilon(\vec{s}, r, s) = 0$, and if $s \notin \mathcal{BR}(\vec{s}, r, s)$ then there must exist s' such that $u(\vec{s} - \hat{s} + \hat{s'}, r, s') > u(\vec{s}, r, s)$, and therefore $\epsilon(\vec{s}, r, s) > 0$.

The definitions of regret and best responses for pure-strategy profiles above can also be extended to mixtures. The regret of a role in $\vec{\sigma}$ is the maximum a role-r player could gain by deviating from their role's mixture $\vec{\sigma}_r$ to some pure strategy, holding others fixed.

[1] Comparing utilities across roles is not necessarily a meaningful operation, but the definition of profile regret facilitates statements about all roles, particularly in the definition of ϵ-Nash equilibrium.

The regret of a role-symmetric mixed strategy is the maximum role regret over $r \in R$. Exploiting the overloading of u to expected utilities of mixtures allows for very similar definitions for regret:

$$\epsilon : \vec{\Sigma} \times R \to \mathbb{R} \ldots \epsilon(\vec{\sigma}, r) = \max_{s \in S_r} u(\vec{\sigma}, r, s) - u(\vec{\sigma}, r) \tag{1.6}$$

$$\epsilon : \vec{\Sigma} \to \mathbb{R} \ldots\ldots\ldots \epsilon(\vec{\sigma}) = \max_{r \in R} \max_{s' \in S_r} \epsilon(\vec{\sigma}, r) \tag{1.7}$$

and best responses:

$$\mathcal{BR} : \vec{\Sigma} \times R \to 2^{S_r} \ldots \mathcal{BR}(\vec{\sigma}, r) = \arg\max_{s \in S_r} u(\vec{\sigma}, r, s) - u(\vec{\sigma}, r) \tag{1.8}$$

Note that by linearity of expectation, mixed strategies are best responses if and only if every strategy in their support is a best response. I have therefore defined $\mathcal{BR}(\vec{\sigma}, r)$ to include only pure strategies, but it is understood that any mixture over best responses is also a best response.

The primary solution concept I employ is *Nash equilibrium*, a (pure- or mixed-strategy) profile from which no player can gain by deviating unilaterally. By definition, a profile $\vec{\sigma}$ is a *Nash equilibrium* if and only if $\epsilon(\vec{\sigma}) = 0$. While all finite games have at least one Nash equilibrium, and a finite role-symmetric game has a role-symmetric Nash equilibrium, computing exact Nash equilibria may be infeasible. Nash equilibrium mixture probabilities can be real numbers that are not representable within machine precision [Nash, 1951]. I therefore focus on approximate (ϵ-Nash) equilibria, profiles $\vec{\sigma}$ where $\epsilon(\vec{\sigma}) \leq \epsilon$ for some threshold ϵ. The problem of computing ϵ-Nash equilibria is complete for the complexity class PPAD [Daskalakis *et al.*, 2009], generally believed to be NP-intermediate. I discuss the replicator dynamics algorithm for identifying role-symmetric mixed-strategy ϵ-Nash equilibria in Section 1.3.

A pure strategy s is *dominated* by pure strategy s' if for all profiles $\vec{s}$ in which s appears, s' is a beneficial deviation: $u(s', \vec{s}_{-s}) > u(s, \vec{s})$ for all $\vec{s} \in \vec{S}$ where $s \in \vec{s}$. Dominated strategies never appear in Nash equilibria and can generally be removed from the game. *Social welfare* $\mathcal{SW}(\cdot)$ is defined as the sum of the utilities to all players under a profile or mixture:

$$\mathcal{SW} : \vec{S} \to \mathbb{R} \ldots \mathcal{SW}(\vec{s}) = \sum_{r \in R} \vec{s}_r \cdot \left\langle u(\vec{s}, r, s) \right\rangle_{s \in S_r} \tag{1.9}$$

$$\mathcal{SW} : \vec{\Sigma} \to \mathbb{R} \ldots \mathcal{SW}(\vec{\sigma}) = \sum_{r \in R} N_r u(\vec{\sigma}, r) \tag{1.10}$$

where the sum in Equation 1.9 is over dot products between utility vectors and role profiles. Social welfare is often used to compare across different equilibria or different environments how well players do in the aggregate, but may not be a useful measure if utility values for different roles incomparable or if distribution of utility among agents is important. Price of anarchy and price of stability compare optimal social welfare to equilibrium social welfare. A game's *price of anarchy* is the ratio of the social welfare of the best profile $\vec{s}$ to that of the worst Nash equilibrium, while the *price of stability* uses the best Nash equilibrium.

A potential problem with computing social welfare, price of anarchy and price of stability lies in comparing utilities across agents. At a fundamental level, utilities describe agents' ordinal preferences over outcomes, and any monotone transformation of utilities describes the same set of ordinal preferences. Mixed strategy equilibrium concepts rely on the further assumption that agents are expected utility maximizers; such an agents' incentives are unaffected by affine transformations of the utility function. If arbitrary monotone or even arbitrary affine transformations were applied to agent utilities, inter-agent utility comparisons like social welfare would hold no meaning. However, in many environments modeled by simulation-based game theory, agents share common motivations such as monetary profit, and among expected-profit maximizers, payoff comparisons are more reasonable. Further, in symmetric game models, all agents face identical incentives and in any symmetric equilibrium agents will achieve the identical expected utility. Thus the difference in social welfare between symmetric equilibria indicates exactly how well any agent fares under those equilibria. In role-symmetric games, comparisons within a role are just as valid as in symmetric games, but as mentioned in the preceding footnote, one should be wary of cross-role payoff comparisons without additional justification.

Even when payoff comparisons between agents have meaning, social welfare may be an inappropriate measure if it hides the distribution of utility among agents. In symmetric equilibria, such distributional concerns are moot, but this is not necessarily the case in asymmetric games such as Setting 1 in Chapter 2. Yet even in this case the social welfare can prove valuable. When a game has price of anarchy 1, all equilibria maximize social welfare, meaning that all equilibria have identical social welfare, and therefore *only* distributional concerns should be relevant to a planner selecting among equilibria. On the other hand, unbounded price of anarchy is achieved in Chapter 2 by making a specific agent's utility arbitrarily bad, which would also be concerning under notions of efficiency more sensitive to the distribution of payoffs.

1.3 Computing Equilibria with Replicator Dynamics

To support analysis of simulation-based games and to provide a proving ground for my proposed algorithms, I have developed an open-source Python package called *GameAnalysis*. This package provides a suite of tools for many types of game-theoretic operations, but employs data structures designed specifically to support fast computation of role-symmetric mixed strategy ϵ-Nash equilibria via replicator dynamics. All equilibria reported in this thesis outside of Section 2.2 were computed by GameAnalysis.

Replicator dynamics [Taylor and Jonker, 1978] is standard algorithm for computing symmetric mixed strategy ϵ-Nash equilibria that can also be used to compute role-symmetric equilibria. Replicator dynamics is initialized from an arbitrary role-symmetric mixed strategy $\vec{\sigma}^0$, then for each iteration $i \geq 0$, the mixture is updated as follows:

$$\vec{\sigma}^{i+1} = \left\langle \frac{(\vec{u}(\vec{\sigma}^i)_r - \mu_r) \circ \vec{\sigma}_r^i}{||(\vec{u}(\vec{\sigma}^i)_r - \mu_r) \circ \vec{\sigma}_r^i||_1} \right\rangle_{r \in R}, \tag{1.11}$$

where μ_r is the minimum payoff in the game for role r. The effect of multiplying each strategy's probability by its expected value and re-normalizing is that strategies with above-average expected value get up-weighted while those with below average expected value get down-weighted. Replicator dynamics iterates until $\vec{\sigma}^i = \vec{\sigma}^j$ for some $j < i$. If $j = i - 1$, the algorithm is said to have converged; otherwise it entered a limit cycle. In practice, convergence is generally declared if $||\vec{\sigma}^i - \vec{\sigma}^j||_\infty < \delta$ for some threshold δ, limit cycle checking is often omitted, and a maximum number of iterations is enforced.

All role-symmetric Nash equilibria $\vec{\sigma}^*$ are fixed points of the replicator dynamics update, because for every role $r \in R$, each strategy $s \in S_r$ is either unplayed: $\vec{\sigma}^*[r, s] = 0$, or has expected utility equal to that of any other strategy in the support: $u(\vec{\sigma}^*, r, s) = u(\vec{\sigma}^*)$. When replicator dynamics converges to a mixture $\vec{\sigma}$, that mixture is frequently an ϵ-Nash equilibrium. Sufficient conditions for this convergence are known, but in practice, checking the regret $\epsilon(\vec{\sigma})$ for each converged mixture $\vec{\sigma}$ is far simpler than verifying sufficient conditions.

Replicator dynamics is not guaranteed to find all equilibria in a game. Every equilibrium-finding algorithm is limited by machine precision, and Nash [1951] showed that equilibrium probabilities can be irrational numbers, but replicator dynamics faces a further restriction. While all Nash equilibria are fixed points of replicator dynamics, they may have trivial basins of attraction: replicator dynamics cannot converge to such equilibria unless initialized there. Instead, replicator dynamics may converge to other equilibria, or may enter a limit cycle. Stable fixed points of replicator dynamics correspond to evolutionarily

stable strategies, a refinement of Nash equilibrium that some analysts prefer as a predictor of agent behavior. In practice, replicator dynamics nearly always finds an equilibrium, and often finds several. As discussed throughout this thesis, beginning in Section 1.4, simulation-based game theory practitioners are for various reasons forced to content themselves with a subset of an environment's equilibria, regardless of what algorithm is used. While all experiments that follow employ replicator dynamics to identify equilibria, there is little reason to believe that my results depend significantly on this choice. First, none of my methods rely critically on replicator dynamics and could be coupled with other equilibrium search algorithms, and second, measures that don't depend on the equilibrium-finding algorithm such as regret, strategic dominance, and social welfare corroborate many of my results.

The critical step in computing equilibria via replicator dynamics is evaluating $u(\vec{\sigma}, r, s)$ for each role and strategy. This expected utility is a sum of the the payoff to s in all profiles played with positive probability under $\vec{\sigma}$ in which s appears, weighted by the profile's probability under $\vec{\sigma}$:

$$u(\vec{\sigma}, r, s) = \sum_{\vec{s} \in \vec{S}} P(\vec{s} - \hat{s} \mid \vec{\sigma}) u(\vec{s}, r, s), \tag{1.12}$$

The probability of an other-agent profile $\vec{s} - \hat{s}$ being realized under mixture $\vec{\sigma}$ can be expressed as follows:

$$P(\vec{s} - \hat{s} \mid \vec{\sigma}) = \left(\prod_{\rho \in R} \frac{(N_\rho - I(\rho = r))!}{\prod_{s \in S_\rho} (\vec{s} - \hat{s})[\rho, s]!} \right) \left(\prod_{\rho \in R} \prod_{s \in S_\rho} \vec{\sigma}_\rho[s]^{(\vec{s} - \hat{s})[\rho, s]} \right), \tag{1.13}$$

where $s \in S_r$ and $I(\rho = r)$ is an indicator with value 1 when $\rho = r$ and 0 otherwise. The first term on the right-hand side is a product over multinomial coefficients for each role and expresses the number of ways the game's players (less one role-r player) can be assigned to the strategies of $\vec{s} - \hat{s}$. The second term expresses the probability of any one of these orderings occurring under $\vec{\sigma}$. I refer to the first term as the *repetitions* of the deviation profile $\vec{s} - \hat{s}$, because a similar expression gives the number of times a role-symmetric profile would be repeated in the full asymmetric payoff matrix for the game.

Note that the expression for the deviation-profile's repetitions does not depend on $\vec{\sigma}$. This means that the repetitions can be pre-computed, and because the repetitions expression for every deviation from every profile in the support is needed when calculating the expected utility of any mixture, this pre-computation can yield enormous savings when running replicator dynamics. The GameAnalysis package applies this insight, along with clever vectorization of profiles, payoffs, and repetitions to achieve fast computation

of expected utilities. The vectorization works via the following arrays with dimensions $|\vec{S}| \times |R| \times \max_r |S_r|$:

- self.counts: represents profiles by player counts for each role and strategy
- self.values: represents payoffs $u(\vec{s}, r, s)$ for the corresponding profile in self.counts
- self.dev_reps: represents repetitions of $\vec{s} - \hat{s}$ for each strategy $s \in \vec{s}$

This representation allows $\hat{u}(\vec{\sigma})$ to be computed in the following two-line Python function (`mix` refers to $\vec{\sigma}$):

```
def expectedValues(self, mix):
    prob = (mix**self.counts).prod(1,2).reshape( \
                                self.values.shape[0],1,1)
    return (self.values * prob * self.dev_reps / mix).sum(0)
```

This game representation makes the bootstrap methods described in Chapter 3 and the test suite described in Chapter 5 feasible.

1.4 Simulation-Based Game Theory

Multi-agent interactions of interest may extend over long time periods with repeated interaction between agents and may involve uncertain outcomes, hidden information, and complex dependencies. In such environments, constructing a normal-form game by enumerating all possible strategies and determining payoffs for all possible outcomes can be an arduous task. Alternative representations such as imperfect-information or extensive form games and corresponding solution concepts can help to capture certain aspects of these interactions, but still require describing and assigning payoffs to the combinatorial space of outcomes. When such description is infeasible, the analyst may be forced to simplify the model, potentially losing key features of the environment.

Simulation-based game theory (SGT)—also known as empirical game-theoretic analysis[2] [Wellman, 2006]—replaces explicit enumeration of outcomes with a procedural description of the environment and agent strategies. Under this paradigm, the analyst writes a program, the environment simulator, that captures the salient aspects of the multi-agent interaction. Agents' strategies are implemented as functions governing their behavior in

[2]EGTA can also encompass game models built from data sources other than simulation. The methods I describe in Chapters 3 and 5 apply to EGTA, while player reduction (Chapter 4) makes sense only in simulation-based settings where the analyst controls the sampling procedure.

the simulated environment, and the agents' strategic decision is over which function to employ. The environment simulator takes as input a strategy profile, simulates agents interacting under those strategies, and returns the payoff achieved by each strategy. The simulator generally involves random elements, so the output of the simulator on input $\vec{s}$ corresponds to a noisy observation of $\vec{u}(\vec{s})$. Simulation-based game theory replaces the modeling problem of directly specifying the game with one of specifying environment and strategy simulators; in many cases, this procedural description can be simpler or easier to calibrate and verify.

The set of simulated agents, their roles, their strategies, and the simulator program implicitly define a role-symmetric game, but explicitly constructing that game is not feasible in general. Let the *true game* be the game model constructed from perfect estimation of the true mean of the sampling distribution for every payoff. In practice, it may be feasible to simulate only a subset of the profiles, because the number of profiles in a role-symmetric game $|\vec{S}| = \prod_{r \in R} \binom{N_r + |S_r| - 1}{N_r}$ grows exponentially in the number of roles, as well as the smaller of players and strategies. Further, it may be feasible to gather only a small number of noisy samples for each profile, especially in cases where simulations can take hours to run [Baarslag *et al.*, 2013; Jordan *et al.*, 2007]. If Θ is the set of observations gathered by simulation, a *modeling function* $\mathfrak{M}(\Theta)$ is used to convert simulation data into a game model; Jordan and Wellman [2009] consider the model-selection problem in detail. In the simplest case Θ contains k samples of every profile and $\mathfrak{M}(\Theta)$ outputs a payoff matrix with the sample average value for each payoff. In this case, Vorobeychik [2010] proved that equilibria in $\mathfrak{M}(\Theta)$ converge to equilibria of the true game as $k \to \infty$. More generally, $\mathfrak{M}$ could estimate payoffs using variance reduction or machine learning, and could potentially output games with fewer players or strategies than the true game, or could use only a subset of the profiles.

Analyzing large simulation-based games thus faces obstacles due to growth in the numbers of players, strategies, and observations. This thesis addresses two of these obstacles: players and observations. In Section 1.5, I discuss existing techniques for efficiently exploring the strategy space of simulation-based games. Chapter 3 addresses the question of how many observations are required by introducing methods based on bootstrapping that allow for statistical confidence in analysis of simulation-based games. Chapters 4 and 5 deal with large numbers of players through, respectively, a new form of player reduction and machine learning techniques.

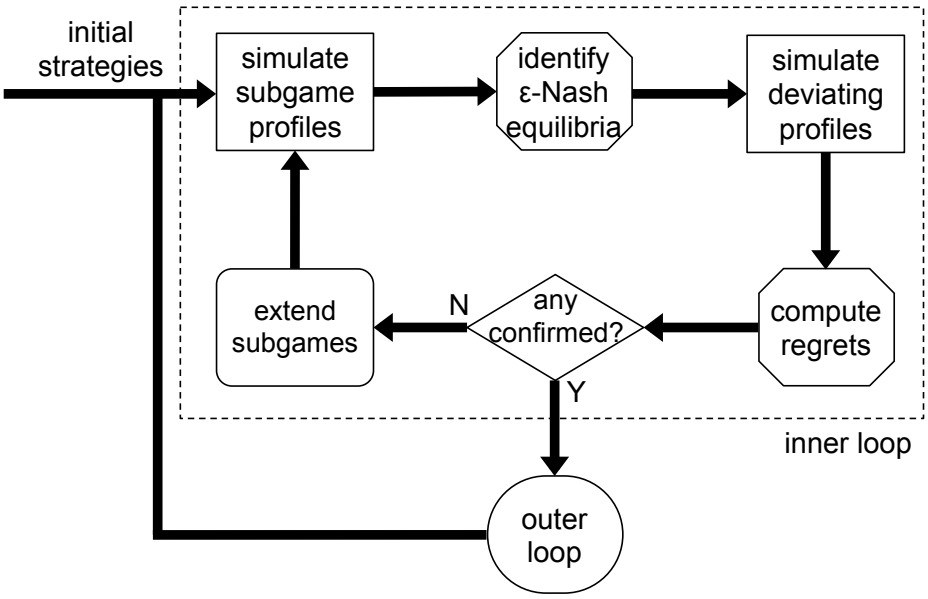


Figure 1.1: Strategy exploration inner loop, as described in steps 1–5 below.

1.5 Strategy Exploration

Because the number of profiles in a role-symmetric game grows exponentially in the number of strategies, it is important to avoid simulating all profiles in a game with a large number of strategies. This is often achieved by an iterative exploration process that searches for small-support equilibria. Lipton *et al.* [2003] showed that in two-player games, ϵ-equilibria with support size logarithmic in $|S_r|$ always exist. More generally, searching for small-support equilibria is common practice even in games where the full payoff matrix can be enumerated; Porter *et al.* [2008] demonstrated that many classes of games have equilibria involving only a small number of strategies, and proposed exploiting this fact for for finding equilibria.

Wellman *et al.* [2013] described a strategy exploration process consisting of two nested procedures: an *inner loop* to find equilibria in a game Γ with a fixed strategy set, and an *outer loop* to enlarge the game by expanding the set of strategies. Let a *subgame* Γ' of game Γ be the restriction of Γ to a subset of the strategies for each role: $\Gamma' = (R, \{N_r\}, \{S'_r\}, u)$, where $S'_r \subseteq S_r$, and the utility function is identical, but restricted to profiles in the subgame. The inner loop starts by performing an initial set of simulations, covering all profiles in a small subgame Γ^0. It then iterates the following steps, illustrated in Figure 1.1.

1. Identify the *maximal complete subgames*, $\{\Gamma^1, \dots\}$. A subgame is *complete* if simulations have been performed for every profile, and a complete subgame is *maximal* if no subgame with a superset of its strategies would be complete.

2. For each maximal complete subgame Γ^i, identify role-symmetric mixed-strategy ϵ-Nash equilibria using replicator dynamics. Let $\vec{\sigma}^i_j$ denote the j^{th} equilibrium found for subgame Γ^i. These subgame equilibria are *candidate* equilibria for the full game Γ.

3. For each $\vec{\sigma}^i_j$, each role, and each strategy $s \in S_r \setminus S^i_r$, simulate all profiles where one player uses s, and all other players use strategies in the support of $\vec{\sigma}^i_j$. This allows exact evaluation of $\epsilon(\vec{\sigma}^i_j)$ in the full game. For a given regret threshold ϵ, if $\epsilon(\vec{\sigma}^i_j) < \epsilon$, the candidate $\vec{\sigma}^i_j$ is *confirmed*, otherwise, it is *refuted*.

4. For each refuted candidate $\vec{\sigma}^i_j$ and each role ρ, *extend* the subgame with the best response to candidate $\vec{\sigma}^i_j$. That is simulate the subgame $(R, \{N_r\}, \{S'_r\}, u)$, where $S'_r = S^i_r$ if $r \neq \rho$, and $S'_\rho = S^i_\rho \cup \mathcal{BR}(\vec{\sigma}^i_j, \rho)$, unless it is subsumed by an existing complete subgame. If any new profiles are simulated, repeat from Step 1.

5. If there exists at least one confirmed candidate $\vec{\sigma}^i_j$, return. Otherwise, choose a maximal subgame Γ^i, extend it with some strategy $s \in S_r \setminus S^i_r$, and repeat from Step 1.

On termination, all equilibrium candidates are confirmed or refuted, and all maximal subgame best-responses are themselves in a completed subgame. As long as the operation of identifying subgame equilibria is complete, the procedure is guaranteed to identify at least one role-symmetric mixed-strategy ϵ-Nash equilibrium of Γ.

The outer loop takes as input a game and a confirmed equilibrium from the inner loop, and attempts to find a new strategy that refutes that equilibrium. Various approaches have been employed for generating new strategies; the credit network study in Chapter 2 takes advantage of the parametric structure of the strategy functions and performs local search. In other applications, reinforcement learning [Schvartzman and Wellman, 2009], and hand-tuning of strategies have been employed.

1.6 Thesis Overview

The remainder of this thesis covers problems to which I have applied simulation-based game theory and methods I have developed for SGT. Chapter 2 introduces a credit network formation game, discusses the limits of traditional game-theoretic approaches in this domain, and describes my simulation-based game model and its results. This work was performed in collaboration with Pranav Dandekar and Ashish Goel [Dandekar *et al.*, 2015]. Chapter 3 addresses getting the most out of sampling data in simulation-based games

through bootstrap methods for computing confidence intervals on game-theoretic analyses, and is based on joint work with Ben-Alexander Cassell [Wiedenbeck *et al.*, 2014]. Chapters 4 and 5 both address the problem of analyzing interaction among large numbers of agents without constructing exponentially-large games. Chapter 4 covers deviation-preserving reduction [Wiedenbeck and Wellman, 2012], a method for approximating large games by constructing reduced games with a much smaller number of players. Chapter 5 reformulates the problem in terms of machine learning and uses Gaussian process regression to construct more flexible and more accurate game models; many of the results in this chapter are unpublished, but an abstract of the work appeared recently [Wiedenbeck and Wellman, 2015].

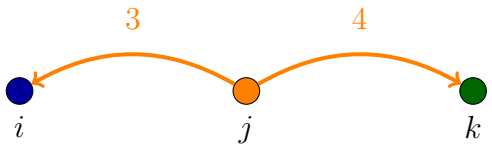
CHAPTER 2

Credit Networks

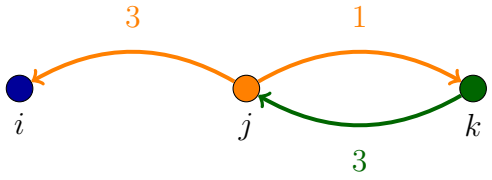
A *credit network* is a model of trust among agents that engage in transactions not denominated by a central currency. In this model, trust manifests as an agent's willingness to perform *favors* for another without immediate compensation. The interpretation of a favor is quite broad: it can be any activity, for example completing a task or delivering a product, that is costly for the provider and beneficial for the recipient. At a basic level, agents without a common currency can transact by barter: exchanging favors to immediate mutual benefit. If agents are willing to accept IOUs for future favors in lieu of direct compensation, many more transactions become possible. Accepting IOUs benefits agents by creating liquidity, but also poses risks: that they will perform far more favors than they receive, or that they will get stuck holding IOUs from another agent who refuses to honor them. To mitigate these risks, agents must select carefully whom to trust and how many IOUs to accept from them.

A credit network records these relationships in a weighted directed graph G where nodes represent agents and trust is quantified as credit balances on the edges between them. Agents specify initial credit limits, where if agent j is willing to accept 3 units of IOUs from agent i and 4 from agent k, the edge (j, i) has weight 3 in the initial credit network, while edge (j, k) has weight 4. This circumstance is represented in Figure 2.1a; subsequent transactions result in updates to the credit network. The (j, k) edge allows agent k to request a favor worth up to 4 units; Figure 2.1b shows the result of a transaction where k receives a favor worth 3 from j. A transaction at price p reduces the recipient's available credit by p units, but also results in the provider holding an IOU for p units. Because this IOU can be redeemed for favors, it can be treated exactly like a credit balance, so the transaction also increases the credit on the reverse edge by p units. Representing IOUs as credit edges means that the total credit available between two nodes is unchanged by a transaction; only the balance of credit shifts.

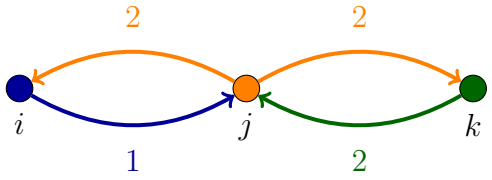
Embedding trust relationships in a credit network also reveals the potential for remote transactions. In the credit network depicted in Figure 2.1b, agent i has no direct credit from



(a) Initial credit network. Agent j trusts both i and k, but for different amounts.



(b) Agent k receives a favor worth 3 from agent j. Agent j now holds IOUs equivalent to 3 units of credit from agent k.



(c) Agent i receives a favor worth 1 from agent k, by swapping IOUs with trusted intermediary j.

Figure 2.1: Updating credit to process two simple transactions.

agent k, but they can still transact via the trusted intermediary j. Figure 2.1c shows the result of a transaction where i requests a favor worth 1 from k. The payment proceeds by i issuing an IOU to j worth 1 unit, and j returning 1 unit of k's outstanding IOU. Observe that the total credit available to the intermediary is unchanged—agent j has merely exchanged an IOU from i for one from k—and agents only hold IOUs from those to whom they issued initial credit. These properties hold for any transaction in a credit network. In a larger network, transactions could be routed along multi-hop paths, or even along multiple paths. In general, a transaction at price p is feasible if the maximum flow of credit from the provider to the recipient is at least p, and payment routing is identical to routing residual flows in a single-commodity flow network.

Some researchers have used credit networks to describe existing environments where agents engage in transactions that are not denominated in a common currency. Others have proposed implementing credit networks in various settings as a mechanism to facilitate distributed transactions on the basis of local trust relationships. The credit network model was invented independently by (at least) four distinct groups, motivated by somewhat different issues and applications, but arriving at the same essential elements.

- DeFigueiredo and Barr [2005] proposed using credit networks to ensure bounded loss from coalitions of malicious users in a reputation system for eBay-style auctions.
- Ghosh *et al.* [2007] aimed to support distributed payment and multi-user credit checking for multi-item auctions conducted without a central currency.
- Mislove *et al.* [2008] sought to deter spam by imposing a small cost on senders, and tracked willingness to accept messages using a credit network.
- Karlan *et al.* [2009] wanted to construct an economic model of trust-based social interactions such as job recommendations or informal borrowing.

The common underlying structure of these models was first noticed by Dandekar *et al.* [2011a], who introduced the unifying term "credit network" and its formal definition. Using this model, they investigated liquidity under various graph structures, finding that many decentralized credit networks support transactions nearly as well as a centralized currency. By introducing suitable definitions of transaction, credit networks apply to a wide variety of settings. For example, the inventors enumerated above interpret transactions respectively as obtaining references guaranteeing good behavior [DeFigueiredo and Barr, 2005], paying for auction winnings [Ghosh *et al.*, 2007], communicating messages [Mislove *et al.*, 2008], and providing social favors [Karlan *et al.*, 2009]. Subsequent authors proposed using this framework to support networked asynchronous bilateral trading [Liu *et al.*, 2010], and bartering of tutorial services [Limpens and Gillet, 2011]. Viswanath *et al.* [2012] argue that all reputation schemes designed for Sybil tolerance have essentially been versions of the credit network idea.

This work follows Dandekar *et al.* [2011a] in studying general properties of the credit network model. Most prior work assumes the existence of a credit network, sidestepping the question of how the network originated. If credit networks are to be implemented in reputation systems or other mechanisms, it will be crucial to understand whether and how agents issue credit. I explore several questions relating to the credit-issuing decisions of strategic agents participating in a credit network. Will they extend any credit at all? What information is important to the agents in making their credit-issuing decisions? What structure will the credit network graph have?

To answer these questions, I study a credit network formation game in which agents decide their initial credit allocations strategically taking into consideration their knowledge about likely transactions and risks. This approach follows a long line of investigation into strategic network formation that seeks to understand the emergent behavior and properties of a network when self-interested agents establish connections to one another based on

their local information [Jackson and Wolinsky, 1996; Bala and Goyal, 2000; Fabrikant *et al.*, 2003; Corbo *et al.*, 2006; Anshelevich and Hoefer, 2012]. In Section 2.1, I describe the general form of the credit network formation game. In Section 2.2 I discuss theoretical analysis of a some basic instantiations of the general game model, and in Section 2.3 I use simulation-based game theory to analyze richer variants.

2.1 Credit Network Formation Game

I study the formation of credit networks through a one-shot game where agents choose initial credit allocations. Extending credit to other agents increases liquidity in the network, enabling more profitable transactions to go through. However, it also entails risk, since a counterparty might default on their outstanding obligations. My co-authors and I explored many variations on the processes generating defaults and transactions, how transactions produce utility, how agents are allowed to extend credit, and what information agents have. There are two basic game settings that each have several variations.

The two game settings described below arise from two distinct approaches to simplifying the game model. By studying a one-shot network formation game I have already abstracted away agents entering and exiting the network dynamically or updating their credit allocations, all of which would probably occur in a real system. However, several factors still complicate game-theoretic analysis of this problem. First, credit networks can involve a large number of agents, and each agent's strategy space is combinatorial and multi-dimensional. Strategies are mappings from all the information an agent has about the environment to all possible credit assignments to the other agents. Second, the expected value to an agent of a credit assignment is defined in terms of the outcome of a stochastic transaction sequence, intermixed with adjustments of credit balances that have important but indirect effects on the probabilities of downstream transactions.

The first approach, taken in Section 2.2, is to impose simplifying assumptions that restrict the set of strategies and the types of transactions. Analysis of the restricted game must then be accompanied by evidence that results could generalize to the unrestricted setting. The second approach, detailed in Section 2.3, employs simulation-based game theory. SGT can help to characterize much richer environments, but provides only empirical evidence, not mathematical proofs, in support of its conclusions. In both of these settings, agents choose strategies for issuing initial credit to form a credit network. Agents then participate in a sequence of random transactions over the credit network. The settings differ in how they model the value of transactions and the risk posed by potential defaulters. In what follows, I first describe the components shared by both settings, followed by the details

	Analytical Model	Simulation Model
time	infinite horizon	fixed finite horizon
agents	asymmetric	ex-ante symmetric
credit	fixed budget, neighbors only	heuristic strategies
transactions	symmetric	asymmetric
risk	can give credit only to friends	random defaults affect utility
utility	transaction success probability as *both* provider and receiver	*net* value from transactions minus cost from defaults

Table 2.1: The most salient differences between analytical and simulation-based credit network formation game models. See Game Settings 1 and 2 for further details.

specific to each model.

The two settings have in common the set of agents $[N] = \{1, \ldots, N\}$, whose strategies produce an initial credit allocation for each other agent. These allocations jointly specify the set of edges E in the initial credit network $G_{init} = ([N], E)$. Agent utilities depend on a sequence of random transactions governed by a transaction distribution $\Lambda = \{\lambda_{ij} \mid i, j \in [N] \wedge i \neq j\}$, where $\sum_{i,j} \lambda_{ij} = 1$. G_0 is derived from G_{init} differently across models, but at each time step $t > 0$, the following events repeat:

- A recipient, provider pair (r^t, p^t) is chosen according to a transaction distribution Λ.
- r^t attempts to send p^t a payment through the previous time step's credit network G_{t-1}.
- If the payment is feasible, then:
  - utilities for r^t and p^t are updated, and
  - the credit network G_t is updated to reflect the payment.
- Otherwise, utilities are unchanged and $G_t = G_{t-1}$.

In both models, there is also an underlying (unweighted, undirected) social network graph H, but its role differs across models. Important aspects in which the models differ are summarized in Table 2.1.

Game Setting 1 (Analytical model used in Section 2.2)
Parameters:

- H: the social network graph (unweighted, undirected).
- X_{ij}: transaction size distributions for each pair of agents.
- Λ: transaction rates. For each pair of agents, $\lambda_{ij} = \lambda_{ji}$.
- B_i: each agent's credit budget.

A strategy for agent i assigns a credit limit c_{ij} to each neighbor $j \in \mathcal{N}(H, i)$, subject to the constraint that $\sum_j c_{ij} \leq B_i$. At each time step, the transaction size is an independent draw from the size distribution X_{ij} for the (i, j) pair drawn from Λ. Transactions begin from the initial credit network $G_0 = G_{init}$, and continue indefinitely. Utility is the steady-state success probability of all transactions in which an agent participates.

$$u^i = \lim_{\tau \to \infty} \sum_{t=0}^{\tau} \frac{I_r^t(i) + I_p^t(i)}{\tau}$$

where $I_r^t(i)$ and $I_p^t(i)$ are indicator functions that have value 1 if agent i participated in a successful transaction as respectively the recipient or the provider at time t.

The model employed in Section 2.2 is outlined in Game Setting 1. Many aspects of this model have been chosen for the sake of analytical tractability, and a few of the simplifying assumptions merit further comment. First, agents face no explicit risk of loss from defaults, but rather are constrained to issue credit only to neighbors in the social network. Second, providing a favor is not costly: agents benefit equally from all transactions in which they participate. The result is that no costs factor into the utility model, so agents reason only about benefits from transactions. However, under these restrictions strong statements can be made about all games matching this setting. Settings 1-A and 1-B are special cases of this model.

Section 2.3 approaches the credit network formation game using simulation-based game theory. Procedural description of the environment allows for a much richer model of the costs and benefits from transactions, as well as the cost from agents defaulting. This comes at the cost of a severe restriction in the strategy space: agents are treated as symmetric and allowed to choose only among a set of heuristic strategies for issuing credit. The basic model is outlined in Game Setting 2; several variants on this model are detailed in Section 2.3.

Game Setting 2 (Simulation-based model used in Section 2.3)
Parameters include a time horizon T, a risk information model, and distributions over the following:

- H: the social network graph.
- x_{ij}: value to agent i of receiving a favor from j.
- Λ: transaction rates.
- δ_i: each agent's probability of defaulting.

A strategy consists of a heuristic by which other agents are ranked, a number of agents k to be given credit, and an amount of credit c to give each. Strategies are chosen ex-ante, but the heuristics can incorporate realizations of random parameters or noisy estimates thereof. At time step $t = 0$, each agent i defaults with probability δ_i, and for each agent $j \in \mathcal{D}$ among the defaulters, other agents lose c_{ij}. Agents in $\mathcal{D}$ are removed from G_{init} to produce G_0, after which T rounds of transactions occur. Each transaction benefits the the recipient by x_{ij} and costs the provider 1. Utility is the sum of costs and benefits from transactions less the cost from defaults:

$$u^i = \sum_{t=0}^{T} \left(I_r^t(i) x_{ip^t} - I_p^t(i) \right) - \sum_{j \in \mathcal{D}} c_{ij},$$

where $I_r^t(i)$ and $I_p^t(i)$ are defined as in Game Setting 1.

2.2 Theoretical Analysis

Game Setting 1 provides the basis for the theoretical analysis. My co-authors first analyzed Setting 1-A, which imposes the further restriction that agents interact only over direct edges with their neighbors. This permits them to characterize the steady-state success probability of transactions between any pair of agents in terms of the total credit issued between them.

My co-authors show that any credit network formation game meeting the conditions of Game Setting 1-A is a potential game [Monderer and Shapley, 1996]. This implies that pure-strategy Nash equilibria always exist and that best-response dynamics always converge to one. They also show that if every transaction size distribution X_{ij} is non-increasing and has support over $[0, \infty)$, then every Nash equilibrium maximizes social welfare, so the game has price of anarchy 1. Moreover, if every X_{ij} is strictly decreasing, they can show that all Nash equilibria are mutually cycle-reachable: one credit network can be transformed into another by routing a payment from an agent to itself along a loop in the graph. This implies (by way of a proof due to Dandekar *et al.* [2011a]) that all Nash

equilibria support identical sequences of feasible transactions.

Setting 1-A (local transactions)
Game Setting 1, with the additional restriction that all transactions are local with respect to the social network.

- Transactions occur only between neighbors: $\lambda_{ij} > 0 \Rightarrow (i, j) \in H$.
- Payments can be routed only over length-1 paths.

These results, while strong, are unsatisfying in that allowing only local transactions severely restricts the benefit from participating in a credit network. The primary motivation behind the credit network model is that it facilitates long-distance transactions despite the locality of trust relationships. I therefore sought to determine whether these results would extend to the more general setting with long-distance transactions. My first theorem shows that the price-of-anarchy result depends on transaction locality: in Game Setting 1, the social welfare of an equilibrium network can be an arbitrarily small fraction of the social optimum.

Theorem 2.1
The price of anarchy in credit network formation games described by Game Setting 1 is unbounded.

Proof (by construction). Consider a game with four agents: $[N] = \{1, 2, 3, 4\}$. The social network H is a line graph over nodes in $[N]$ with edges $E = \{(1, 2), (2, 3), (3, 4)\}$. For each node $i \in [N]$, $B_i = 1$. The non-zero transaction rates are given by: $\lambda_{12} = \lambda_{21} = \lambda_{34} = \lambda_{43} = \lambda_1 > 0$ and $\lambda_{14} = \lambda_{41} = \lambda_2 \gg \lambda_1$. All other transaction rates in Λ are zero.

Agents 1 and 4 each have only one neighbor, to whom it is a dominant strategy to issue all their credit. Agents 2 and 3 each have positive transaction probabilities with only one other agent, and in both cases that agent is a neighbor, giving them a dominant strategy of issuing all credit to that neighbor. Since all agents have dominant strategies, the equilibrium network G shown in Figure 2.2a has $c_{12} = B_1$, $c_{21} = B_2$, $c_{34} = B_3$, and $c_{43} = B_4$. However, this network prevents transactions between agents 1 and 4, which occur with high probability λ_2. The network G^* shown in Figure 2.2b permits these transactions.

Let $p_{ij}(G)$ be the steady-state probability that a transaction between agents i and j succeeds in G, conditional on agents i and j being selected to transact. The social welfare

(a) The unique equilibrium network G (b) One of two optimal networks G^*

Figure 2.2: Example of a Setting 1 game with unbounded price of anarchy (Theorem 2.1). All edges depicted have credit capacity $c_{ij} = B_i$: the issuer's entire credit budget.

of each network can be expressed as a function of these probabilities:

$$\mathcal{SW}(G^*) = \lambda_1\Big(p_{12}(G^*) + p_{21}(G^*)p_{34}(G^*) + p_{43}(G^*)\Big) + \lambda_2\Big(p_{14}(G^*) + p_{41}(G^*)\Big)$$
$$\mathcal{SW}(G) = \lambda_1\Big(p_{12}(G) + p_{21}(G) + p_{34}(G) + p_{43}(G)\Big)$$

As long as X_{14} places non-zero probability on some transaction size $x \leq \min(B_1 + B_2, B_3, B_4)$, there exists a non-zero lower bound $0 < \bar{p}$ on the conditional success probabilities between agents 1 and 4 in G^*: $\bar{p} \leq p_{14}(G^*)$, and $\bar{p} \leq p_{41}(G^*)$. On the other hand, with finite budgets and positive transaction sizes, there must be an upper bound $\tilde{p} < 1$ on the conditional success probabilities for all pairs that transact in G: $\tilde{p} \geq p_{12}(G)$, $\tilde{p} \geq p_{21}(G)$, $\tilde{p} \geq p_{34}(G)$, and $\tilde{p} \geq p_{43}(G)$. These bounds yield an upper bound for $\mathcal{SW}(G)$ and a lower bound for $\mathcal{SW}(G^*)$ in terms of λ_1 and λ_2:

$$\mathcal{SW}(G) \leq 4\lambda_1\tilde{p}$$
$$\mathcal{SW}(G^*) \geq 2\lambda_2\bar{p}$$

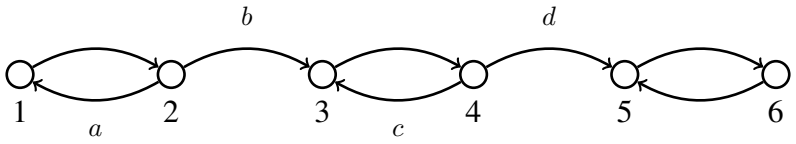
Scaling λ_1 and λ_2 can then make the social welfare from transactions among pairs 1/2 and 3/4 an arbitrarily small fraction of the social welfare from 1/4 transactions. In the limit as $\lambda_1 \to 0$, the ratio $\mathcal{SW}(G^*)/\mathcal{SW}(G) = \infty$. □

Setting 1-B (unit transactions)
Game Setting 1 with the additional restrictions that all transactions are size-1 and all agents have 1-unit budgets.

- X_{ij} is a degenerate distribution with support $\{1\}$.
- $B_i = 1$

My second theorem calls into question the generalizability of the other major result from Game Setting 1-A. With local transactions, a pure-strategy Nash equilibrium always

(a) Credit edges fixed by dominant strategies. Agents 1 and 6 have only one neighbor, while agents 3 and 5 have non-zero transaction probability with only one other agent.



(b) The transaction rates in Theorem 2.2 cause a best-response cycle where agent 2 switches between edges a and b while agent 4 switches between edges c and d.

Figure 2.3: Example of a Setting 1-B game that has no pure-strategy Nash equilibrium.

exists. In Theorem 2.2, I give an example of a credit network formation game with no pure-strategy equilibria. This theorem uses a slightly different restriction to Game Setting 1 outlined in Setting 1-B. Here, all transactions transfer one unit of credit, and all agents have credit budgets of one unit. This counterexample is somewhat weaker than that of Theorem 2.1 in the sense that it relies strongly on the discreteness of Setting 1-B.

Theorem 2.2

There exist credit network formation games described by Game Setting 1-B that have no pure-strategy Nash equilibria.

Proof (by construction). Consider a game with six agents: $[N] = \{1, 2, 3, 4, 5, 6\}$. The social network H is a line graph over nodes in $[N]$ with edges $E = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)\}$. The non-zero transaction rates are given by: $\lambda_{12} = \lambda_{21} = \lambda_{34} = \lambda_{43} = \lambda_{56} = \lambda_{65} = 0.001, \lambda_{14} = \lambda_{41} = \lambda_{26} = \lambda_{62} = 0.2435, \lambda_{46} = \lambda_{64} = 0.01$. All other entries in the transaction rate matrix Λ are zero.

The edges depicted in Figure 2.3a are the strictly dominant strategies for agents 1, 3, 5, and 6. Agents 5 and 6 have only one neighbor, with whom they transact, and to whom they issue their full budgets. Agents 3 and 5 each have exactly one other agent with whom they transact, both of whom are neighbors and therefore receive the entire credit budget. On the other hand, agents 2 and 4 do not have dominant strategies. In Figure 2.3b, agent 2 selects between edges a and b, while agent 4 selects between edges c and d. Because Λ is symmetric and transactions have unit size, steady-state transaction success probabilities are easy to calculate:

$$u(\langle a, c\rangle, 2) = 2p_{12}\lambda_{12} = 2 \cdot \frac{2}{3} \cdot .0001 = 1.\bar{3} \times 10^{-4}$$

$$u(\langle a, c\rangle, 4) = 2p_{34}\lambda_{34} = 2 \cdot \frac{2}{3} \cdot .0001 = 1.\bar{3} \times 10^{-4}$$

$$u(\langle a, d\rangle, 2) = 2p_{12}\lambda_{12} = 2 \cdot \frac{2}{3} \cdot .0001 = 1.\bar{3} \times 10^{-4}$$

$$u(\langle a, d\rangle, 4) = 2p_{34}\lambda_{34} + 2p_{46}\lambda_{46} = 2 \cdot \frac{1}{2} \cdot .0001 + 2 \cdot \left(\frac{1}{2} \cdot \frac{2}{3}\right) \cdot .001 = 7.\bar{6} \times 10^{-4}$$

$$u(\langle b, c\rangle, 2) = 2p_{12}\lambda_{12} = 2 \cdot \frac{1}{2} \cdot .0001 = 1 \times 10^{-4}$$

$$u(\langle b, c\rangle, 4) = 2p_{14}\lambda_{14} + 2p_{34}\lambda_{34} = 2 \cdot \left(\left(\frac{1}{2}\right)^2 \cdot \frac{2}{3}\right) \cdot .2345 + 2 \cdot \frac{2}{3} \cdot .0001 = 7.83 \times 10^{-2}$$

$$\begin{aligned} u(\langle b, d\rangle, 2) &= 2p_{12}\lambda_{12} + 2p_{26}\lambda_{26} \\ &= 2 \cdot \frac{1}{2} \cdot .0001 + 2 \cdot \left(\left(\frac{1}{2}\right)^2 \cdot \left(\frac{2}{3}\right)^2\right) \cdot .2345 = 5.22\bar{1} \times 10^{-2} \end{aligned}$$

$$\begin{aligned} u(\langle b, d\rangle, 4) &= 2p_{14}\lambda_{14} + 2p_{34}\lambda_{34} + 2p_{46}\lambda_{46} \\ &= 2 \cdot \left(\left(\frac{1}{2}\right)^3\right) \cdot .2345 + 2 \cdot \frac{1}{2} \cdot .0001 + 2 \cdot \left(\frac{1}{2} * \frac{2}{3}\right) \cdot .001 = 5.9391\bar{6} \times 10^{-2} \end{aligned}$$

Thus, agents 2 and 4 play the following 2×2 game, which has a best-response cycle and no pure-strategy Nash equilibrium.

	c	d
a	$1.\bar{3} \times 10^{-4}$, $1.\bar{3} \times 10^{-4}$	$1.\bar{3} \times 10^{-4}$, $7.\bar{6} \times 10^{-4}$
b	1×10^{-4}, 7.83×10^{-2}	$5.22\bar{1} \times 10^{-2}$, $5.9391\bar{6} \times 10^{-2}$

□

Together, Theorems 2.1 and 2.2 call into question the generalizability of results proved in Setting 1-A. This motivates the use of simulation-based game theory to better understand more realistic models of credit network formation. Inspired by the presence of star-like equilibrium networks in some of the SGT settings described in Section 2.3, my co-authors went on to investigate another analytical model that explicitly included default probabilities. In that setting, they faced some of the same difficulties stating general results about all equilibria, but showed that star-like networks can be socially optimal and can result from a form of sequential arrival greedy dynamics.

2.3 Simulation-Based Game Analysis

More relaxed scenarios, such as those with asymmetric transactions, unconstrained budgets, multiple credit issuance, explicit risk of defaults, or incomplete information have thus far proven elusive for analytic treatments. I therefore employ simulation-based game theory to explore environments that relax the conditions for which theorems can be proved. Simulation handily deals with the complex stochastic and dynamic factors bearing on the evaluation of credit-issuing strategies; its key advantage lies in estimating utilities for complex transaction models by tracking the state of the credit network through a sequence of random transactions. However, SGT requires complete enumeration of the strategy set and explicit instantiation of environment parameters such as the number of agents or the social network structure. Despite these inherent limitations, a systematic exploration of reasonable parameter ranges and well-motivated heuristics can provide useful and general insights.

2.3.1 Simulation Setup

Game Setting 2 above forms the basis for my simulated environment. In contrast to Game Setting 1, the simulations treat agents as ex-ante symmetric decision makers who choose a credit-issuing strategy before knowing the realization of random parameters of the simulator. However, the strategies can depend on those realizations. For example, agents can commit to giving credit to the agent with the lowest default probability before they know who that agent will be. In combination with the restriction of the strategy space to the heuristics described below, this helps to prevent agents from over-fitting their strategies to the peculiarities of the environment as was possible in Theorem 2.2. Because the payoffs to each profile are determined by repeated simulations, and environment parameters are drawn independently for each run, strategies must be robust to the distribution of environment parameters and opponent strategies. Further, the generality of the available heuristics precludes precise best responses to specific parameter values or opponent credit allocations. For example, agents giving credit to the least-likely defaulter do not have the option to withhold credit if they have the lowest default probability themselves.

I analyze twelve environments, differing on risk model, default prevalence, and transaction surplus. Each simulation run evaluates a profile of heuristic strategies for issuing credit. All agents apply their assigned strategies to issue credit based on their available information, forming an initial credit network. The simulation processes probabilistic defaults and a stochastic sequence of transactions to generate sample payoffs for the strategy profile. Strategies are selected from a suite of heuristics, which implement a variety of

criteria for issuing credit, parametrized by how many agents to issue credit and how much. The choice of profiles to simulate was driven by the inner loop (identifying equilibria) and outer loop (generating strategies) described in Chapter 1. This produced simulation-based game models for each of the twelve environments.

The Setting 2 box below provides details for some of the environment parameters. I use a population of 61 agents, and each run of the scenario comprises 10,000 transaction request events. The transaction rate λ_{ij} for each pair of agents $i \neq j$ is drawn from a Pareto distribution with shape parameter $\alpha = 2$ and then normalized. All transaction requests from i the recipient to j the provider are for a single unit. The value to i of a receiving a favor from j is drawn uniformly, $x_{ij} \sim U[1, \bar{x}]$, with $\bar{x}$ set to either 1.2 (*low value*) or 2 (*high value*). The provider incurs a constant cost of 1 from each successful transaction, but can potentially benefit from the accrued credit in subsequent transactions as a recipient. The recipient's payment covers the provider's cost by transferring one unit of credit, so the average surplus per transaction is either 0.1 or 0.5.

Default probabilities δ_i for each agent are drawn from a Beta distribution: $\beta(1, 9)$ (average default probability $\frac{1}{10}$) in the *low default* setting, $\beta(1, 2)$ (average $\frac{1}{3}$) in the *medium default* setting, and $\beta(1, 1)$ (average $\frac{1}{2}$) in the *high default* setting. I consider two models of information about default probabilities: *global risk* and *graded risk*. In the global risk environment every agent's default probability is common knowledge, whereas in the graded risk environment each agent gets sample data from the default distribution of others. The number of samples ∂_{ij} is determined by the social network distance between i and j: $\partial_{ij} = 100$ if i and j are neighbors, $\partial_{ij} = 10$ if they are linked through one other node, $\partial_{ij} = 1$ if they have a shortest-path of length three, and $\partial_{ij} = 0$ otherwise. The social network itself is an Erdös-Rényi random graph where each possible edge is present with probability $p = .05$, giving each agent an average degree of 3.

I explored environments with high, medium, or low default, and high or low value, for each of global and graded risk. The twelve environments are listed in Table 2.2, along with the number of profiles and strategies I ended up simulating, in both the full and reduced games. Three-letter environment names are coded by risk model (C[omplete information] for global risk, I[ncomplete] for graded risk), default probability (L[ow]/M[edium]/H[igh]), and recipient value (L[ow]/H[igh]). These numbers of profiles and strategies are broken down by two stages (I and II) of search, as described below.

A heuristic strategy is defined by three parameters: (i) a criterion for ranking the other agents, (ii) the number k of agents to issue credit (the best k according to the ranking criterion), and (iii) the number of units c of credit to issue to each of these chosen agents. The criteria included in heuristics along with the (k, c) values considered in this study are

Setting 2 (parameter values)
This setting elaborates Game Setting 2 in Section 2.1, by enumerating the simulation parameters used. The following parameters could in principle be varied in the credit network simulator, but were fixed across all of my experiments:

- $N = 61$
- $T = 10\,000$
- $H = G(n = 61, p = .05)$: an Erdös-Rényi random graph with average degree 3.
- λ_{ij} are drawn from $Pareto(\alpha = 2)$, then normalized so that $||\Lambda||_1 = 1$.

The following parameters were varied across experiments:

- $x_{ij} \sim U[1, \bar{x}]$, where $\bar{x} \in \{1.2, 2\}$.
- $\delta_i \sim \beta(1, b)$, where $b \in \{1, 2, 9\}$.
- The *risk information model* is either *global* or *graded*.

enumerated below, defined from the perspective of agent i's evaluation of credit prospect j:

- Default probability: lowest known default probability (δ_j) for global risk, or lowest estimated default probability based on samples ∂_{ij} for graded risk.
- Request rate: highest probability that i will request a favor from j (λ_{ij}).
- Provision rate: highest probability that j will request a favor from i (λ_{ji}).
- Request value: highest expected value of requested favors per time step ($\lambda_{ij} x_{ij}$).
- Net profit: highest difference between the expected value to i of favors i requests from j and those j requests from i ($\lambda_{ij} x_{ij} - \lambda_{ji}$).
- Index: lowest node number (arbitrary global labeling).
- Random: uniform choice.

In addition, I included the no-credit strategy, Zero, which issues no credit to anyone. Request rate, Request value, and Net profit are all variations on the idea that agents should give credit to the agents they want to transact with. Extending credit does not immediately make transactions with such agents feasible, but any payment routed through that edge will

name	Risk model	Default prob	buyer surplus	Full-game profiles (I/II)		DPR profiles (I/II)		Strategies (I/II)	
CLL	Global	low	low	4619	11695	1497	3946	17	32
CLH	Global	low	high	2179	9861	765	3359	17	32
CML	Global	med	low	3557	9425	1036	3213	8	32
CMH	Global	med	high	8619	20192	2622	6474	15	32
CHL	Global	high	low	3134	5901	1045	2090	17	32
CHH	Global	high	high	3202	6155	1101	2196	17	32
ILL	Graded	low	low	991	9322	394	3148	17	32
ILH	Graded	low	high	5377	28721	1824	8786	17	32
IML	Graded	med	low	1766	8927	565	3109	8	32
IMH	Graded	med	high	24612	39728	6818	11356	18	32
IHL	Graded	high	low	656	5080	261	1881	17	32
IHH	Graded	high	high	430	10529	201	3516	17	32

Table 2.2: Exploration performed by the iterative SGT process under various environment settings. *Strategies* gives the number of strategies added by the outer loop. *Full-game profiles* gives the number of 61-agent profiles sampled by the inner loop. *DPR profiles* gives the number of 6-player profiles in the reduced-game model.

reverse it, enabling profitable transactions. Provision rate takes the opposite approach, extending credit to the agent most likely to reverse the edge, in the hope of routing transactions through that agent later. The Random strategy is present mostly as a sanity check: its appearance in an equilibrium would cast doubt on the validity of the corresponding setting.

Observe that the Default strategies behave qualitatively differently in the global and graded risk environments. Under global risk, all agents have the same information about default probabilities. Therefore, when multiple agents issue credit to the least-likely defaulters, they are all creating edges to the same target agents. This leads to a centralized or star-like credit network, as illustrated in Figure 2.4. Such coordination on credit targets has potential advantages. If everyone including i offers credit to j, then once i provides a favor (to j directly to or another agent routing payment through j), i enjoys credit paths to essentially everyone in the network. This coordination does not result, in contrast, from mutual application of Default in the graded risk model. Under graded risk, agents have different information based on their positions in the social network. The counterparts judged to have lowest default probably are invariably those with whom the agent has had most positive experience. Since there is little experience of any kind with social strangers, these are unlikely to be judged most trustworthy (this happens only if one is unlucky enough to have

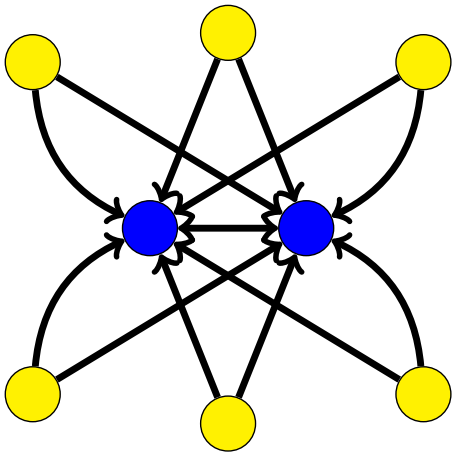


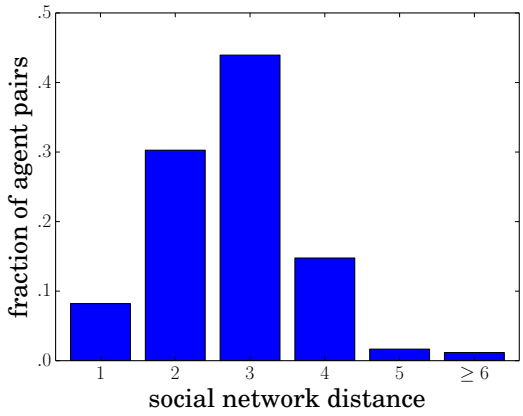
Figure 2.4: A star-like credit network similar to one that would arise from the mutual application of default-based strategies in the global risk model.

only very untrustworthy friends). As shown in Figure 2.5b, under graded risk over 65% of the top five estimated least likely defaulters are neighbors in the social network. Finally, note that the Index strategies do coordinate on a star-like network, in either the global or graded risk model. Comparing Default and Index strategies allows separation of the pure benefits of coordination from the benefits of avoiding defaulters.

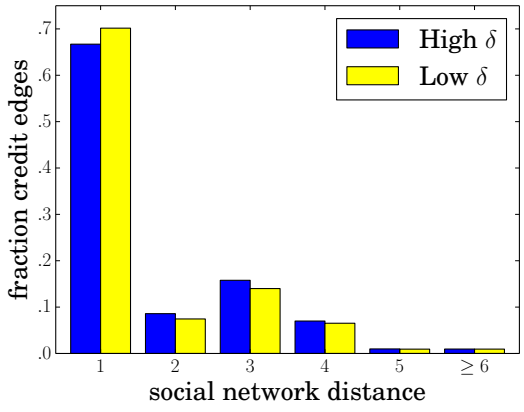
Due to the combinatorial growth of a game's profile space, operating on the 61-agent game directly is infeasible. I therefore employed a 6-player deviation-preserving reduction game (Chapter 4) in the following analyses. Figure 2.6 illustrates the application of DPR to construct a 6-player reduced game from a 61-agent full game. The analysis proceeded in two stages. Stage I considered a fixed set of 17 strategies, and ran the inner loop on eight of the twelve environments: those with high or low (not medium) default probabilities. The 17 predefined strategies were selected based on exploration in a preliminary study, and are enumerated in Table 2.3.

Criterion	Predefined (k, c)	Automatically Generated (k, c)
Default	$(1,1), (2,2), (3,2), (5,2)$	$(3,1), (4,1), (5,1), (6,1), (8,1), (9,1)$
Request rate	$(1,1), (1,2), (2,2)$	$(2,1), (4,1), (8,1), (10,1)$
Provision rate	$(2,2)$	$(6,1)$
Request value	$(2,2), (5,2)$	$(2,1), (3,1)$
Net profit	$(2,2), (6,2), (8,1)$	$(3,2), (5,1)$
Index	$(1,1), (2,2)$	
Random	$(2,2)$	
Zero	$(0,0)$	

Table 2.3: Strategies included in the SGT study: predefined for Stage I, and automatically generated in the outer loop for Stage II.



(a) Distance in H between all pairs of agents. mean ≈ 2.7.



(b) Distance in H to agents selected by the Default heuristic with $k = 5$. mean ≈ 1.6.

Figure 2.5: Under graded risk, social network neighbors are much more likely than average to be estimated as among the five least likely defaulters.

For the four medium default environments, I started with a smaller set of eight predefined strategies (the first listed for each criterion), and employed the automated strategy generation procedure (the outer loop) to extend the set. On each iteration, the outer loop searched for refutations of an equilibrium $\vec{\sigma}$ confirmed for the existing strategy set, employing local search from a particular existing strategy. The search algorithm simply hill-climbs from the existing strategy, holding its credit criterion fixed but incrementing or decrementing its k and c parameters by 1. The search halted upon reaching a local maximum in payoff, assuming all other nodes in the network play according to $\vec{\sigma}$. If that local maximum exceeds the equilibrium payoff, the new strategy is added to the set and another round of the inner loop is initiated. If instead the strategy categories (in this case, defined by credit criterion) are tried without finding a beneficial deviation, the entire process concludes.

As indicated in Table 2.2, the CML and IML environments found no new strategies, whereas the CMH environment added seven strategies to the original eight, and IMH added ten. Together, there were 15 automatically generated strategies not included among the 17 predefined Stage I strategies. These are listed in the final column of Table 2.3. For Stage II, I constructed the combined set of 32 strategies, and ran the SGT inner loop for each environment with this set.

With 17 strategies, there are 1.4×10^{16} distinct strategy profiles for the full 61-player game, and 74,613 for the six-player DPR game. These numbers grow to 3.0×10^{24} (61-player) and 2,324,784 (six-player DPR) for the Stage II set of 32 strategies. As indicated in Table 2.2, the SGT process evaluated only a very small fraction of these profiles at each stage. Nevertheless it was able to identify equilibria in each environment. Altogether

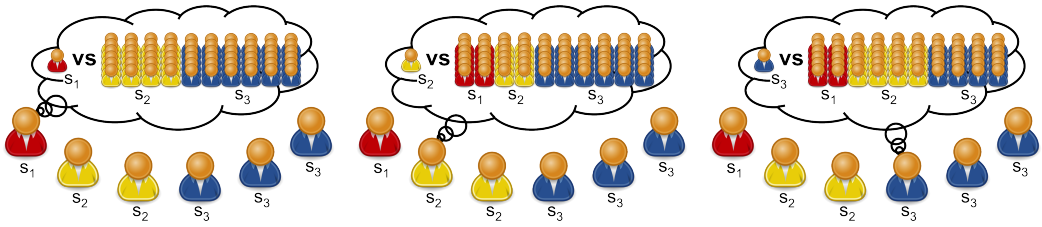


Figure 2.6: Deviation-preserving reduction as applied to the credit network formation game. Each of six reduced-game players views itself as controlling one of 61 full-game agents while each opponent controls an equal fraction (12) of the remainder. The payoff to s_1 in the reduced-game profile comes from the full-game profile depicted in (a); s_2 from (b); s_3 from (c). See Chapter 4 for an in-depth treatment of DPR.

165,536 full-game profiles were evaluated across the twelve credit network environments, from which payoffs for 53,074 DPR profiles were estimated. Each full-game profile evaluated was simulated at least 1,000 and usually upwards of 2,000 times. The simulations were performed on the University of Michigan Advanced Research Computing cluster, using an experiment management facility designed expressly for simulation-based game studies [Cassell and Wellman, 2013].

2.3.2 Results

The process described in Section 2.3.1 successfully derived equilibria for each of the twelve credit network games. Specifically, between one and six symmetric mixed-strategy Nash equilibria were identified in the six-player DPR games corresponding to each environment. All candidate subgame equilibria were either confirmed or refuted by the process, and the subgames covering best responses to all candidates were completed.

The strategies based on Provision rate, Index, and Random heuristics are not supported in any equilibria. To characterize the equilibria qualitatively, I partition the remaining strategies as follows. Class **D** represents Default, **Z** represents Zero, and **T** groups together strategies based on criteria related to transaction probability and value: Request rate, Request value, and Net profit. The symmetric mixed-strategy Nash equilibria identified are summarized in Figure 2.7. In the figure, there is one cell for each environment, displaying class labels for strategies supported in some equilibrium. A class letter circled means that a strategy in that class was confirmed as a pure strategy Nash equilibrium. Interestingly, whereas many of the equilibria found were mixed, and several environments had equilibria in multiple classes, in no case did a single equilibrium mix across the class partitions defined above.

Figure 2.7 shows a no-credit equilibrium (Z) in eight of the twelve environments: all

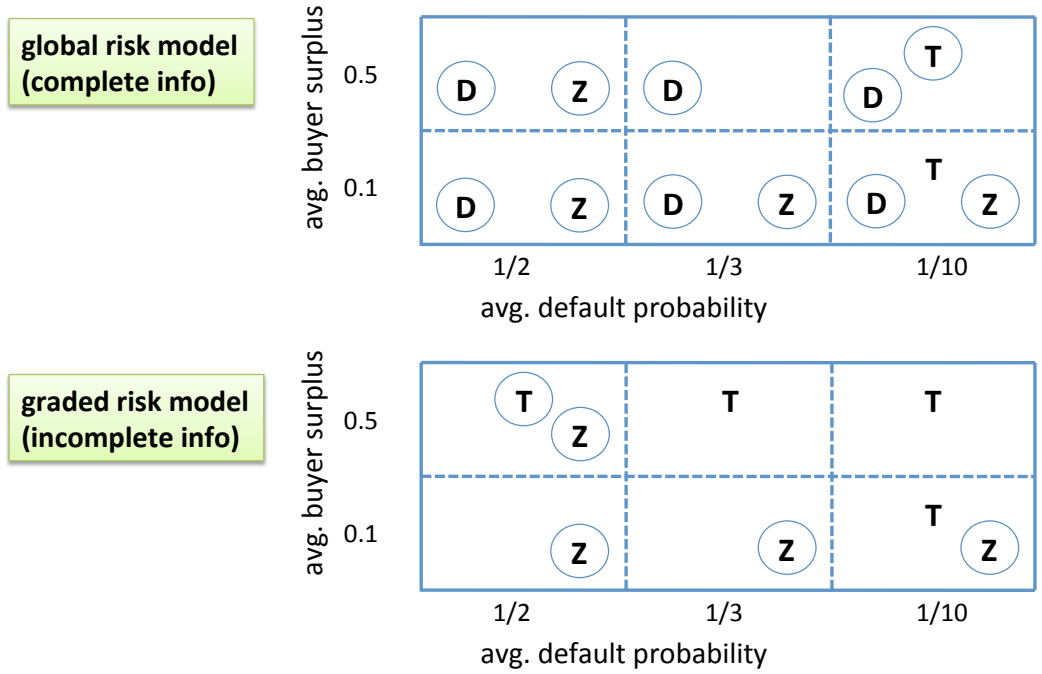


Figure 2.7: Equilibria found for the twelve credit network environments. Letters denote the strategy classes represented in equilibrium, with circled letters indicating pure-strategy equilibria.

but those with low or medium default and high recipient value. The two least favorable environments—graded risk with high or medium default and low value—have only this equilibrium, whereas all the others have some equilibrium where credit is provided. All of the global risk environments have an equilibrium where everybody plays Default, but this strategy does not appear in equilibrium for any graded risk environments. Indeed, there is a one-to-one correspondence between the equilibria for the two risk classes, except that the graded risk environments omit these Default equilibria, and when recipient value is high, these are replaced by transaction-based equilibria. The weakened information about defaults plus the lack of coordinating power render this a poor credit-issuing criterion in graded risk environments.

For completeness, I list all confirmed equilibria. Strategies are specified by their heuristic, number of opponents given credit k, and amount of credit given to each c, in the format Heuristic(k, c). Groups in brackets with probabilities represent mixed-strategy equilibria, and ungrouped strategies indicate pure-strategy equilibria.

CLL Default(1,1); Default(3,1); Default(4,1); Zero; [Request rate(2,1), 0.899; Request value(3,1), 0.101]; [Request value(2,1), 0.806; Request value(3,1), 0.194]

CLH Default(3,2); Net profit(5,1); [Default(6,1), 0.951; Default(8,1), 0.049]; [Default(5,1),

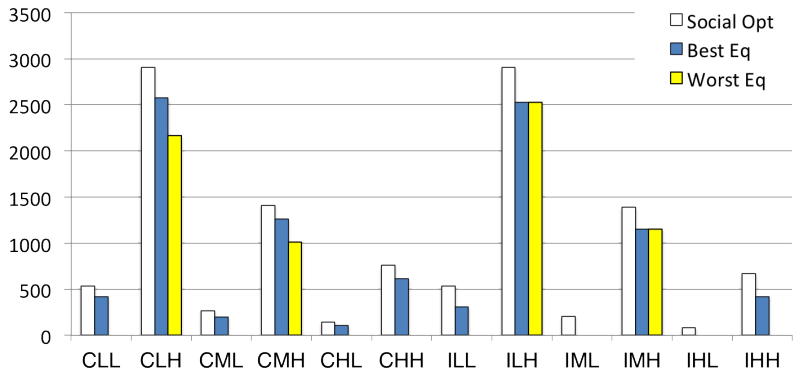


Figure 2.8: Welfare at empirical social optimum compared to welfare at equilibrium.

0.744; Default(8,1), 0.256]

CML Default(1,1); Default(3,1); Zero

CMH Default(2,2); [Default(2,2), 0.014; Default(6,1), 0.986]; [Default(3,2), 0.880; Default(4,1), 0.120]; [Default(5,1), 0.821; Default(6,1), 0.179]

CHL Default(1,1); Default(3,1); Zero

CHH Default(2,2); Zero; [Default(3,2), 0.081; Default(4,1), 0.919]

ILL Zero; [Request value(2,1), 0.637; Request value(3,1), 0.363]

ILH [Request rate(4,1), 0.229; Net profit(5,1), 0.771]

IML Zero

IMH [Request rate(4,1),0.172; Net profit(5,1),0.785; Request value(3,1),0.042]

IHL Zero

IHH Request rate(4,1); Zero

Whereas the set of equilibria evolved as the simulation-based game was refined from Stage I to Stage II, the *qualitative categories of strategy profiles represented in equilibrium* (as depicted in Figure 2.7, ignoring the circle designations) remained constant.

I next turn to the question: How well do the credit networks generated in equilibrium perform? Figure 2.8 compares the welfare (sum of agent utility) of equilibrium outcomes to that of an estimated social optimum. This estimate is actually a lower bound, equal to the greatest social welfare seen in any full-game profile simulated. Equilibrium welfare varies across equilibria, hence I present the best and worst of those identified, corresponding

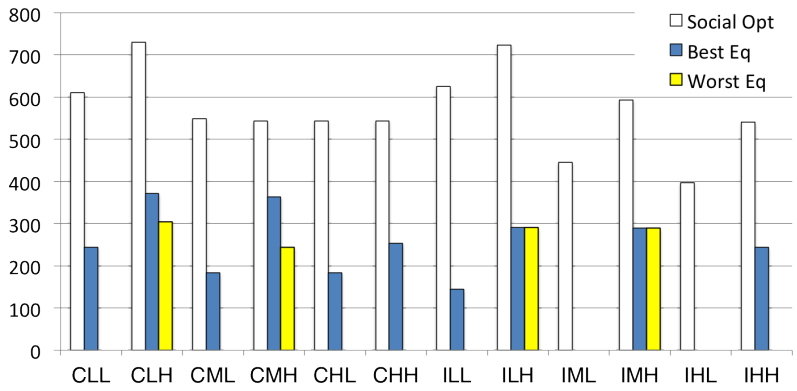


Figure 2.9: Total credit issued at empirical social optimum compared to equilibrium.

roughly to price of stability and price of anarchy respectively. These values are approximations because both player reduction and strategy exploration result in unexplored profiles that could have higher social welfare. In addition, because not all equilibria may be found, ones with either better or worse social welfare may exist with the exception that social welfare cannot go below that of a Zero equilibrium. In eight of twelve environments, the worst is the Zero equilibrium, which supports no transactions and thus yields zero welfare. Overall, when there is a substantial amount of welfare possible (*i.e.*, the most favorable environments), equilibrium network formation does a good job of obtaining most of it. For less favorable environments, a network—if it forms at all—tends to produce little utility.

It is also possible to also observe directly the amount of credit issued in equilibrium networks, as compared to the social optimum—which is not necessarily the credit-maximizing network. As seen in Figure 2.9, the comparison mirrors that for welfare, but with lower ratios of equilibrium to social optimum across the board. This is due to the diminishing returns to credit, once the network has ample credit capacity. In other words, a substantial fraction of available social welfare can be achieved without issuing this same fraction of the credit that a social planner would.

All of these results are of course relative to the particular strategy space included in the simulation-based game analysis. The choice was driven by an effort to span a diverse space, and to include strategies successful in preliminary studies or otherwise representing plausible prospects for refuting initial equilibrium candidates. The fact that adding strategies in Stage II based on automated exploration of parametric variations on the original strategies did not change the qualitative character of equilibria lends support to the robustness of these results.

2.4 Conclusions

In the simplest model of credit network formation, my co-authors proved strong results about existence equilibria and price of anarchy. Unfortunately, my counterexamples suggest that these results do not generalize to models that more closely match plausible applications of credit networks. Simulation-based game theory provided useful insight into a more interesting setting, characterizing the conditions needed for non-empty equilibrium networks, giving a sense of the heuristics that can plausibly be used for issuing credit, and demonstrating that equilibrium credit networks can often achieve most of feasible gains from trade. A particularly striking result is the importance of information about default risk: when agents shared good information about who is trustworthy, a central currency equilibrium was always feasible, but without such knowledge, a central currency never arose.

Analyzing rich strategic environments like the credit network formation game via pencil-and-paper or simulation-based game theory requires careful modeling decisions, trading off tractability with fidelity. The credit network domain demonstrates the potential for the two approaches to complement and build off of one another. Further, both approaches face concerns of generalizability that motivate my work to improve methods for simulation-based game theory, through statistical methods, player reduction, and machine learning.

CHAPTER 3

Bootstrap Methods for Statistical Confidence

By nature, the agent-based simulations on which SGT is founded produce noisy observations, requiring that many samples be gathered to accurately estimate agent payoffs. The credit network formation study presented in Chapter 2 is typical in that every profile selected for simulation was observed thousands of times. Simulation-based game analysis is therefore fundamentally statistical: as samples are added, payoff estimates may change, altering the game model and conclusions drawn from it. Ideally, practitioners would report statistical confidence intervals for their analysis results. For example, approximate Nash equilibria would be accompanied by a bound ϵ where the profile is estimated to be an ϵ-Nash equilibrium with 95% probability.

Unfortunately, statistical methods for simulation-based game theory are underdeveloped, and the few tools that exist are rarely applied. As a consequence, published results rarely reflect the underlying uncertainty of simulation-based game models. The general dearth of statistical analysis in simulation-based game theory is somewhat understandable in light of the difficulty of evaluating complex game-solution hypotheses in a traditional statistical framework. Simulation is necessary precisely because little is known about payoff distributions a priori, rendering parametric approaches that rely on such knowledge inapplicable. Moreover, the regret of a mixed-strategy equilibrium candidate is a complicated statistic that requires calculating the maximum gain over possible deviations, each of which sums over a large number of these unknown, possibly correlated distributions.

In the absence of good statistical tools, many analysts have gathered an extremely large number of samples before performing analysis (as I do in Chapter 2; see also Wah and Wellman [2015] and Wellman *et al.* [2008]). In such cases, it likely that integrating better statistical methods could have saved substantial sampling effort or allowed that effort to be spent more effectively exploring additional strategies or alternative environment settings. In other cases, where individual simulations are much more costly, gathering thousands of samples of every profile is infeasible. Researchers including Jordan *et al.* [2007] and Veness *et al.* [2011] have employed variance reduction techniques to get the most out of

limited data, but without confidence measures, such studies run the risk of gathering insufficient samples to draw empirically valid conclusions. In either case, methods to quantify statistical confidence are important for ensuring that published results reflect fundamental properties of the games studied and that simulation-based game analysis can be taken as serious scientific evidence for propositions of interest.

Because the most common analysis performed on simulation-based games is computing approximate Nash equilibria, ensuring that reported simulation-based game equilibria reflect equilibria of the true game is paramount. Vorobeychik [2010] proved that in the infinite-sample limit, simulation-based game and true-game equilibria are identical. However, after any finite number of samples, noise present in simulation data may cause spurious equilibria to appear in the simulation-based game that have high true-game regret. Thus, when reporting an ϵ-Nash equilibrium of a simulation-based game, analysts would benefit from the ability to report a 95% or other confidence bound for ϵ in the true game. Regret (Equation 1.7) is computed as a maximum over utility differences, and those expected utilities are linear combinations of a large number of payoffs, each of which is estimated by sampling from an unknown distribution. In addition, payoff observations may not be drawn independently due to simultaneous sampling of all payoffs for a profile or by use of common random numbers across profiles. The non-linearity of the regret statistic, and the unknown, possibly correlated underlying payoff distributions pose significant challenges to analytic approaches to confidence interval construction. My approach is based on *bootstrapping*—reviewed in Section 3.1—which leverages the the full observation set to characterize distributions over game-theoretic conclusions. The key idea is to simultaneously resample all payoffs in a simulation-based game to construct a bootstrap game, compute regret (or another statistic) in the bootstrap game, and repeat to create a bootstrap distribution.

3.1 Background

3.1.1 Bootstrap Statistics

The bootstrap is a computational method for estimating distributional information about a statistic computed on sample data [Davison and Hinkley, 1997]. Evaluating statistical hypotheses and computing confidence intervals often relies on knowledge of a statistic's *sampling distribution*, the distribution of values the statistic takes when computed on sets of k samples samples drawn from the population. For certain population distributions (*e.g.* Gaussian) and certain statistics (*e.g.* sample mean) a closed form for the sampling distri-

bution is available, but for unknown distributions and complicated statistics, classical tests that require such a closed form may be inapplicable. The bootstrap, in contrast, does not rest on explicit assumptions about the shape of the distribution. Instead, it uses resampling to estimate the sampling distribution empirically.

The bootstrap treats a sample set as representative of the population from which it was drawn for the purpose of computing distributional statistics. The sample set can then be treated as a population and resampled. If the original sample has size k, then each resample is a set of size k drawn with replacement from that sample. Under the assumption that the sample set is representative of the original population, each resample emulates drawing k samples from the population. The statistic is computed on each resample set, and collectively these values constitute a bootstrap distribution, which approximates an empirical sampling distribution for the statistic, but without the work of drawing many new sample sets. The bootstrap distribution can then be used in place of a sampling distribution for constructing confidence intervals or performing other tests.

3.1.2 Related Work

Two lines of related research bear mentioning: the use of bootstrapping in other simulation-based studies, and other proposed statistical methods for simulation-based game theory. In the agent-based modeling community, bootstrap statistics have seen limited use, one exampling being the suggestion by Axtell *et al.* [1996] that a bootstrap approach may be necessary for determining if two agent-based models are equivalent, due to the complicated nature of such a hypothesis. By contrast, discrete-event systems modeling has seen greater adoption of the bootstrap to analyze the output of simulation [Friedman and Friedman, 1995]. More broadly, Cohen [1995] advocates the adoption of bootstrapping in artificial intelligence experiments.

Two non-bootstrapping methods for estimating true game regret from payoff sample data have been proposed in the literature. Reeves [2005] suggested estimating the probability that a profile is an exact Nash equilibrium by sampling game matrices from the space of possible matrices induced by assuming every payoff is independent and distributed normally with mean and variance equal to its sample mean and sample variance respectively. The approach could be straightforwardly extended to give confidence intervals for mixtures. This method would be similar to the bootstrap-based methods I describe in Section 3.2 in that it constructs an empirical distribution of regret values, but differs in two important aspects. First, Reeves's empirical distribution does not emulate a sampling distribution for k-sample regret, but rather for 1-sample regret. My experiments confirm that

regret estimates for candidate equilibria shrink as the number of samples grows, indicating that single-sample regret is likely to significantly overestimate true regret. Second, Reeves assumes Gaussian noise rather than taking advantage of the sample data.

Vorobeychik [2010] presented a Bayesian framework for bounding the posterior probability that a profile or mixture is an ϵ-Nash equilibrium of the true game from payoff sample data. He provides two versions of this bound: a tight bound that assumes payoff observations are independent draws with Gaussian noise, and a weaker distribution-free bound. The first method may provide a useful guide for sampling decisions, but may not accurately quantify statistical confidence when its assumptions are violated by non-Gaussian or correlated payoff distributions. The distribution-free bound can help to establish scientific facts, but may require vastly more samples than would be called for by the first bound or other better-tailored stopping rules. Whether either of these bounds is useful in practice is an open question.

Both existing methods for simulation-based game statistics assume that payoff noise is normally distributed and that each payoff's noise is independent. Importantly, no empirical evaluations have been attempted for either of the above approaches, leaving open questions about their applicability to actual simulation-based games, and neither has been employed in recent simulation-based game studies. To help forestall similar questions and increase the likelihood of adoption for my methods, I provide extensive experimental validation.

3.2 Computing Bootstrap Confidence Intervals for Regret

My primary goal is to approximate a sampling distribution for the true-game regret of equilibria computed in simulation-based games, from which I can estimate regret confidence bounds. The regret of a role-symmetric mixture[1] $\vec{\sigma}$ depends on the payoffs to every profile that can be realized with positive probability under $\vec{\sigma}$; in the worst case—a mixture with full support—regret depends on every payoff value in the game. The key idea in adapting the bootstrap to compute confidence intervals for regret is to simultaneously resample all of these distributions. Using a resample set from each distribution, I construct a bootstrap game and compute regret in that game, contributing one observation to the bootstrap distribution for regret. This procedure is described in greater detail below, and specified formally in Algorithm 1.

Consider a simulation-based game $\Gamma = \mathfrak{M}(\Theta)$, where Θ is the data set collected via

[1] I describe my methods in terms of role-symmetric mixed strategies, but they are equally applicable to pure-strategy profiles or non-role-symmetric mixtures. My experiments evaluate the methods for pure-strategy and symmetric mixed-strategy Nash equilibria in symmetric games.

Algorithm 1 $bootstrapRegret(\Theta, \mathfrak{M}, \vec{\sigma}, b)$

Require: Θ, the simulation data
Require: $\mathfrak{M}$, the modeling function
Require: $\vec{\sigma}$, the profile to evaluate
Require: b, the number of bootstrap games to construct
  $d = \{\}$
  for $0 \leq i < b$ **do**
    $\widetilde{\Theta} = \{\}$
    for each $\theta \in \Theta$ **do**
      $\widetilde{\Theta} = \widetilde{\Theta} \cup \{resample(\theta)\}$
    end for
    $\widetilde{\Gamma} = \mathfrak{M}(\widetilde{\Theta})$
    $d = d \cup \{\epsilon(\widetilde{\Gamma}, \vec{\sigma})\}$
  end for
  return d // bootstrap distribution

simulation and $\mathfrak{M}$ is the modeling function used to instantiate a game from simulation data. Θ can be broken down into observation sets θ, each of which represents a collection of samples from a single (potentially multivariate) distribution. Most often, θ represents the simulation results for a single profile, but it may instead have samples for multiple profiles that were simulated jointly, or for individual payoff values if each strategy's payoff was determined independently. In the example shown in Figure 3.1a, $\Theta = \left\{\theta_{\langle 2,0\rangle}, \theta_{\langle 1,1\rangle}, \theta_{\langle 0,2\rangle}\right\}$, with observation sets corresponding to profiles:

$$\theta_{\langle 2,0\rangle} = \{0, 1\}$$
$$\theta_{\langle 1,1\rangle} = (\{0, 0, 0\}, \{0, 0, 0\})$$
$$\theta_{\langle 0,2\rangle} = \{0, 0, 1\}$$

The profile $\langle 0, 2\rangle$, where both agents play B as three payoff observations: two zeros and a one. The profile $\langle 1, 1\rangle$ has observations listed for A first, followed by observations for B, but with no variation in payoff observations there is only one reasonable value for $\vec{u}(\langle 1, 1\rangle)$, namely $(0, 0)$. The modeling function averages observations for each payoff, resulting in the payoff matrix in Figure 3.1b.

My method performs resampling at the level of the observation sets, drawing $|\theta|$ values with replacement from θ to form $\widetilde{\theta}$. Jointly, these resampled observation sets constitute $\widetilde{\Theta} = \{\widetilde{\theta}\}_{\theta \in \Theta}$, a resampling of Θ. The modeling function is then applied to create a bootstrap game: $\widetilde{\Gamma} = \mathfrak{M}(\widetilde{\Theta})$. Table 3.1 shows the possible values that $u\left(\langle 2, 0\rangle, A\right)$ and $u\left(\langle 0, 2\rangle, B\right)$ can take in the bootstrap game, the probability with which each bootstrap game occurs, and

$$u(\langle 2,0\rangle, A) \in \{0,1\}$$
$$u(\langle 1,1\rangle, A) \in \{0,0,0\}$$
$$u(\langle 1,1\rangle, B) \in \{0,0,0\}$$
$$u(\langle 0,2\rangle, B) \in \{0,0,1\}$$

(a) Payoff observations.

	A	B
A	$\frac{1}{2}$, $\frac{1}{2}$	0 , 0
B	0 , 0	$\frac{1}{3}$, $\frac{1}{3}$

(b) Sample-average payoff matrix.

Figure 3.1: Example observation data Θ and the resulting payoff matrix $\mathfrak{M}(\Theta)$ for a symmetric 2-player, 2-strategy game.

the corresponding regret of the mixture $\langle \frac{1}{2}, \frac{1}{2} \rangle$. There are three ways was to select $\widetilde{\theta}_{\langle 2,0 \rangle}$, and four ways to select $\widetilde{\theta}_{\langle 0,2 \rangle}$, yielding 12 unique bootstrap games. In such a simple game all bootstrap games can be enumerated and their probabilities computed, allowing exact specification of the bootstrap CDF of $\epsilon\left(\mathfrak{M}(\widetilde{\Theta}), \langle \frac{1}{2}, \frac{1}{2} \rangle\right)$ shown in Figure 3.2. In games with more profiles and non-trivial observation sets, this distribution cannot be constructed exactly, but each bootstrap game is a sample from it. I therefore construct a large number of bootstrap games, and in each, I compute the regret of $\vec{\sigma}$. These values constitute a bootstrap distribution for regret, which can be used to construct confidence intervals: the 95^{th} percentile estimates a one-sided 95% confidence interval for true-game regret.

Several different levels of granularity are possible for θ, corresponding to different levels of correlation induced by sampling and/or game model construction. In the standard case, each run of the simulator returns an observation for the payoff to every strategy in a profile, and $\mathfrak{M}$ constructs a game model by simply averaging observations of each payoff value. Each observation set therefore contains payoff data for a single profile. If deviation-preserving reduction (Chapter 4) is employed, each payoff value in the reduced game comes from a distinct full-game profile, so observation sets would normally contain data for a single payoff value. If different simulation runs use common random numbers to reduce variance, this induces a correlation across profiles, and the observation sets would preserve this correlation; in the extreme, the simulation data could consist of a single observation set: $\Theta = \{\theta\}$. By preserving correlations present in the simulation data, my method can avoid imposing the independence assumptions required by prior techniques.

In some cases, it may be worth inducing additional correlation in the bootstrap resamples that is not justified by the game's sampling process. If a large fraction of the game's payoffs has the same number of observations, bootstrap game can be constructed much more efficiently by resampling that set of payoffs with common indexing, that is by in-

prob	$u\left(\langle 2,0\rangle, A\right)$	$u\left(\langle 0,2\rangle, B\right)$	$\frac{\max}{2}$	$u\left(\langle \frac{1}{2}, \frac{1}{2}\rangle\right)$	$\epsilon\left(\langle \frac{1}{2}, \frac{1}{2}\rangle\right)$
$\frac{8}{108}$	0	0	0	0	0
$\frac{12}{108}$	0	$\frac{1}{3}$	$\frac{2}{12}$	$\frac{2}{24}$	$\frac{2}{24}$
$\frac{6}{108}$	0	$\frac{2}{3}$	$\frac{4}{12}$	$\frac{4}{24}$	$\frac{4}{24}$
$\frac{1}{108}$	0	1	$\frac{6}{12}$	$\frac{6}{24}$	$\frac{6}{24}$
$\frac{16}{108}$	$\frac{1}{2}$	0	$\frac{3}{12}$	$\frac{3}{24}$	$\frac{3}{24}$
$\frac{24}{108}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{3}{12}$	$\frac{5}{24}$	$\frac{1}{24}$
$\frac{12}{108}$	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{4}{12}$	$\frac{7}{24}$	$\frac{1}{24}$
$\frac{2}{108}$	$\frac{1}{2}$	1	$\frac{6}{12}$	$\frac{9}{24}$	$\frac{3}{24}$
$\frac{8}{108}$	1	0	$\frac{6}{12}$	$\frac{6}{24}$	$\frac{6}{24}$
$\frac{12}{108}$	1	$\frac{1}{3}$	$\frac{6}{12}$	$\frac{8}{24}$	$\frac{4}{24}$
$\frac{6}{108}$	1	$\frac{2}{3}$	$\frac{6}{12}$	$\frac{10}{24}$	$\frac{2}{24}$
$\frac{1}{108}$	1	1	$\frac{6}{12}$	$\frac{12}{24}$	0

Table 3.1: Possible regret values of a $\langle \frac{1}{2}, \frac{1}{2}\rangle$-mixture when the game in Figure 3.1 is resampled. The left-most column gives the probability with which each bootstrap game can occur, and the right-most column gives the regret value in that bootstrap game. Intermediate columns show the components from which $\epsilon\left(\langle \frac{1}{2}, \frac{1}{2}\rangle\right) = \frac{1}{2}\max\left[u\left(\langle 2,0\rangle, A\right), u\left(\langle 0,2\rangle, B\right)\right] - u\left(\langle \frac{1}{2}, \frac{1}{2}\rangle\right)$ is calculated.

cluding the same number of copies of the i^{th} element of each observation set $\theta \in \Theta$ for $i \in \{1 \ldots k\}$. In the *GameAnalysis* package, this is implemented by drawing a vector of resample counts for each observation index and multiplying all payoff observation sets by it in a single vectorized operation. Because many bootstrap games must be constructed, resampling under the approximation that Θ consists of one or a small number of observation sets can relieve a major bottleneck in estimating confidence intervals. If confidence interval calculation is used in the inner loop, running the faster version at intermediate steps may be worthwhile. In initial experiments, I found little difference when inducing extraneous correlation in this way, and the results presented in Section 3.3 therefore use common-index resampling even though the artificial noise was generated independently for each payoff.

A one-sided confidence bound for regret is appropriate for determining whether a reported equilibrium of the simulation-based game is likely to be an equilibrium of the true game, but for some purposes, two-sided confidence intervals are more appropriate. In particular, determining whether to conduct additional simulations requires distinguishing whether a profile is probably an equilibrium, probably not an equilibrium, or whether additional data is required. In this case, my co-authors used the 2.5^{th} and 97.5^{th} percentiles of

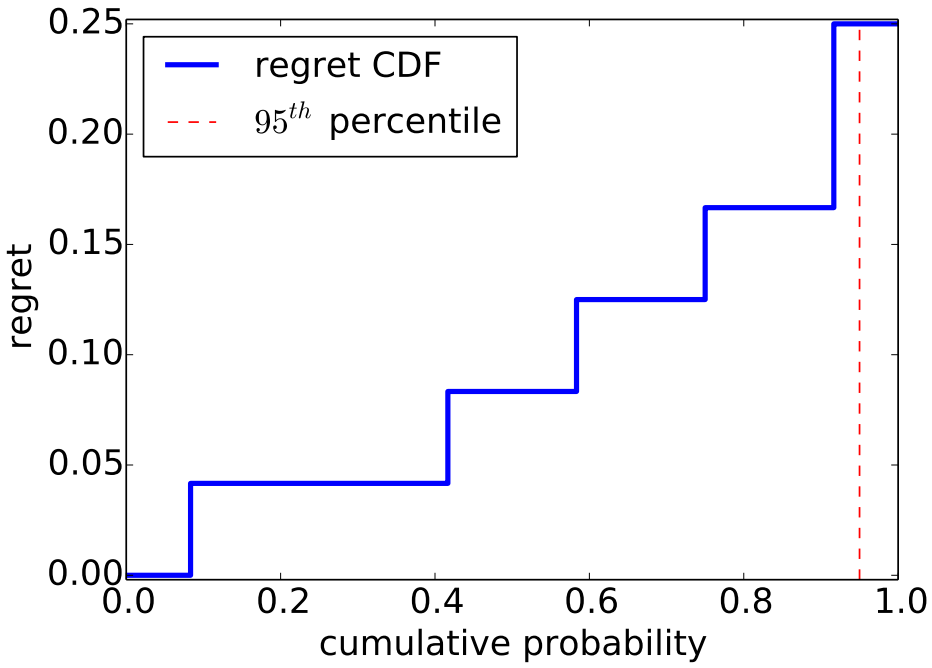


Figure 3.2: Exact CDF of the bootstrap regret of $\langle \frac{1}{2}, \frac{1}{2} \rangle$ in the game from Figure 3.1. Corresponds to the left- and right-most columns of Table 3.1. Approximated by Algorithm 1. The dashed line shows that the distribution's 95$^{\text{th}}$ percentile is the same as its maximum. With a non-trivial number of payoff observations this would not occur.

the bootstrap distribution d to give a 2-sided 95% confidence interval. Algorithm 1 could also be modified to compute confidence intervals for a statistic other than regret, such as social welfare, by replacing $\epsilon(\widetilde{\Gamma}, \vec{\sigma})$ with $f(\widetilde{\Gamma}, \cdot)$ for statistic f.

An important restriction of my method is that confidence intervals are valid only with respect to the modeling function $\mathfrak{M}$. The "true game" in this sense is $\mathfrak{M}(\Theta^{\infty})$, where $\Theta^{\infty} = \{\theta_{\infty}\}_{\theta \in \Theta}$ is the hypothetical result of taking an infinite number of samples of each observation set. If $\mathfrak{M}$ performs a player reduction (Chapter 4) or employs regression to estimate payoffs (Chapter 5), the bootstrap confidence intervals do not provide additional information about the full game. This means that bootstrap confidence intervals for regret computed in the reduced credit network formation game (Chapter 2) would be valid only for the 6-player DPR game; regret in the 61-agent full game is unknown, because the bootstrap has no information about the huge fraction of the profiles that remains unobserved.

3.3 Calibration Experiments

For the bootstrap regret confidence intervals to be useful, I need to show that they are well-calibrated. I hypothesize that bootstrap distribution of regret accurately approximates a

sampling distribution for regret. This is a sufficient condition for the 95$^{\text{th}}$ percentile of the sample-game bootstrap distribution of the regret of a candidate equilibrium to provide a valid 95% confidence bound for the true-game regret of that candidate. This hypothesis yields several testable predictions that I evaluate empirically.

The first prediction is that confidence bounds should be well-calibrated, namely the true-game regret of an equilibrium candidate should fall below the 95$^{\text{th}}$ percentile of the bootstrap distribution 95% of the time. In fact, if the bootstrap distribution closely approximates a sampling distribution for regret, then all quantiles should be well-calibrated; if other quantiles were off, this would cast suspicion on the generalizability of results for 95% confidence intervals. In addition, confidence bounds should grow tighter as data is acquired, so the 95$^{\text{th}}$ percentile of the bootstrap distribution should shrink as the number of payoff observations grows. I also expect confidence bounds to be wider when data are more noisy, so the 95$^{\text{th}}$ percentile should grow as the variance of payoff samples grows.

3.3.1 Experimental Setup

I test the predictions of my hypothesis by artificially generating symmetric true games and drawing samples from known noise distributions centered around each true game payoff. I then compute pure-strategy Nash equilibria and symmetric mixed-strategy Nash equilibria in the resulting noisy games,[2] and use the bootstrap method to estimate a regret distribution for each of these equilibrium candidates. These bootstrap estimates are compared against the true-game regrets of the equilibrium candidates. Across a large number of randomly generated true games, the hypothesis predicts that k% of true-game regret values will fall below the k^{th} percentile of the noisy game's bootstrap regret distribution.

My experiments employ two classes of synthetic games: uniform symmetric games (uSym) and congestion games (Cgst) [Rosenthal, 1973], as well as one class of simulated game: credit network games (CredNet). To generate a true game from the uSym class, I draw a value from the distribution $U[0, 100]$ for each unique payoff in a symmetric game with $n \in \{2, 4, 6\}$ players and $|S| \in \{2, 4, 6\}$ strategies. The results on uSym games shown in Section 3.3.2 are from games with 4 players and 4 strategies; results for other combinations of players and strategies are similar. To generate a true game from the Cgst class, I use 5 players and 3 strategies; each strategy s has a base value $v_b(s) \sim U[0, 3]$, a linear congestion cost $v_l(s) \sim U[0, 1]$, and a quadratic congestion cost $v_q(s) \sim [0, 1]$. Parameters of the quadratic cost functions are drawn randomly to generate congestion game instances. The Cgst payoff to a player choosing strategy s is a function of the total number $n(s)$ of

[2] In all of these experiments, mixed-strategy equilibria are computed using replicator dynamics.

players choosing that strategy: $u(s) = v_b(s) - v_l(s)n(s) - v_q(s)(n(s))^2$. CredNet games are generated based on data from the credit network formation game simulator described in Chapter 2.

In my initial experiments, I generated a CredNet game with 6 players, 6 strategies, and 2644 samples of each payoff, but found that it had unreasonably high variance. I therefore also generated a second data set with the same players and strategies called CredNet-agg, where each of the 1000 samples comes from 20 pre-aggregated runs of the simulator. The true game in the CredNet experiments is always the simulation-based game constructed using the full set of samples. To facilitate comparison of regret values across classes, I applied an affine transformation to rescale each uSym and Cgst true-game payoff matrix to match range $[0, 100]$, which closely matches the payoff range of the CredNet true game.

Given a true game from the uSym or Cgst classes, I created noisy samples of each payoff by drawing from a known distribution centered at the true-game payoff and constructed sample games from these sample sets. For Cgst, I added only normally distributed noise, but across uSym experiments I varied the noise distribution among normal, uniform, bimodal Gaussian mixture, and Gumbel (a skewed distribution with mode $<$ mean). Each of these distributions has some parameter corresponding to data spread, which I refer to as z. For all experiments, I drew $z \sim U[0, \bar{z}]$ independently for every true-game payoff. The maximum value for the spread parameter $\bar{z} \in \{0.1, 1, 10, 100, 1000\}$ was varied over four orders of magnitude. For normally distributed noise, observations of each payoff are drawn from a normal distribution with variance z; for bimodal Gaussian, the z parameter controls the variance of the two Gaussians, which are spread apart by a random draw from $|N(0, \bar{z})|$; for uniform noise, z is equal to the half-width of the distribution; for Gumbel noise, z is the scale parameter. I also tested drawing noise from different models for different payoffs in the same game, and found similar results (not shown). To create each sample game for the CredNet class, I selected a subsample without replacement out of the full set of simulator observations.

3.3.2 Experimental Results

For each combination of synthetic game class, number of players/strategies, and noise model, I generated 1000 true games. For each synthetic and simulated true game, I generated sample games with sample sizes ranging from 5 to 500, and in each sample game, I computed pure- and mixed-strategy Nash equilibria. I then computed bootstrap distributions for the regret of each equilibrium, which I compared to that equilibrium's true-game regret. Table 3.2 shows, in the "95% calibration" columns, the fraction of true-game re-

game, noise $\bar{z} = 100$	samples	95% calibration pure	95% regret pure	95% calibration mixed	95% regret mixed
uSym, normal	10	0.924	34.4	0.951	25.9
uSym, normal	100	0.947	1.5	0.955	6.3
uSym, bimodal	10	0.949	71.6	0.957	50.1
uSym, bimodal	100	0.935	13.5	0.949	12.7
Cgst, normal	10	0.928	20.6	0.966	18.1
Cgst, normal	100	0.972	0	0.941	1.8
CredNet-agg	10	0.981	1.51	0.997	1.04
CredNet-agg	100	0.971	0	0.927	0.23

Table 3.2: Calibration and mean regret of 95% confidence bounds for regret of simulation-based game equilibria across various game classes. Calibration measures the fraction of the time that true-game regrets fall below the 95$^{\text{th}}$ percentile of the bootstrap distribution.

grets that fell below the 95$^{\text{th}}$ percentile of the bootstrap distribution for a subset of game settings, noise models, and sample sizes. The data indicates that the 95$^{\text{th}}$ percentile of the regret bootstrap distribution provides a reasonably well-calibrated 95% confidence interval for true-game regret of both pure and mixed-strategy Nash equilibria computed in noisy games. The confidence intervals for the CredNet games are the least accurate; this is borne out much more starkly by Figure 3.5 below.

Table 3.2 also shows the average regret value of the 95% confidence bound. Two trends are noteworthy here: first, the bounds tighten when going from 10 to 100 samples; this is explored further in Figure 3.6 below. Second, the average regret at the bound is in many cases extremely high relative to the $[0, 100]$ payoff scale of the games. These profiles should clearly not be reported as equilibria, and their presence in the noisy games demonstrates the importance of including confidence interval estimation in the simulation-based game analysis process. The results for uSym and Cgst games in Table 3.2 are broadly representative of similar experiments with different numbers of players and strategies, different noise distributions and magnitudes, and different numbers of samples.

While Table 3.2 shows good calibration for the 95% bootstrap confidence bound, one would hope that the whole bootstrap regret distribution, and not just the 95$^{\text{th}}$ percentile is well-calibrated. Figure 3.3 shows that for 4-player, 4-strategy uSym games with normal noise this is indeed the case: each curve shows the cumulative fractions of noisy game equilibria for which the true-game regret fell below each successive percentile of the bootstrap distribution. Because the curves closely track the 45° line, it can be concluded that on average the shape of the bootstrap distribution closely matches that of the sampling distribution for regret. Such plots for other uSym games with other player and strategy counts,

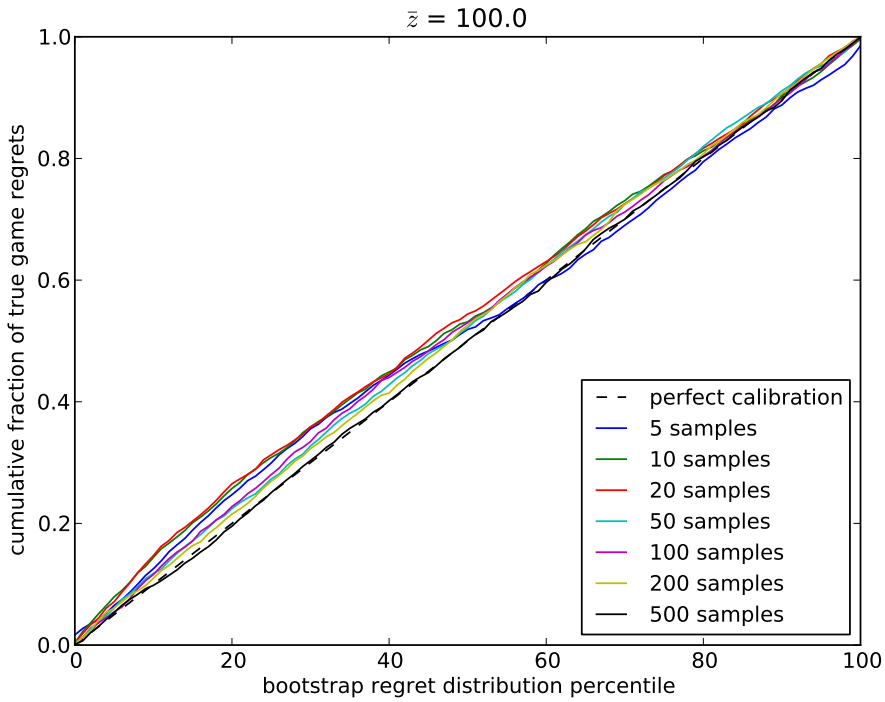


Figure 3.3: Bootstrap distributions are well-calibrated for 4-player, 4-strategy uSym games with $\bar{z} = 100$ Gaussian noise. Curves show the fraction of true-game regrets falling below each bootstrap distribution percentile.

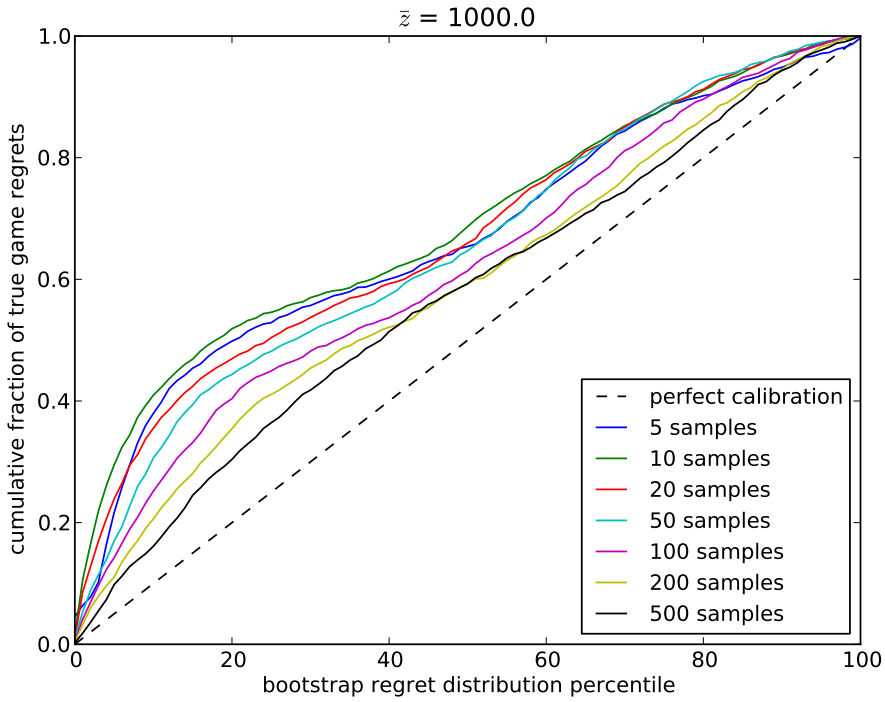


Figure 3.4: Bootstrap distributions are poorly calibrated when noise swamps payoff information. Settings identical to Figure 3.3, except $\bar{z} = 1000$.

other noise distributions, and smaller noise magnitudes are broadly similar, as are results for Cgst games.

Two cases where bootstrap calibration breaks down are illustrated in Figures 3.4 and 3.5. Figure 3.4 shows an experiment identical to Figure 3.3, except that noise magnitude has been increased to $\bar{z} = 1000$. It is relatively unsurprising that regret estimates are poor when noise variance is an order of magnitude larger than the difference between the minimum and maximum payoffs in the game. Figure 3.5 shows results for a similar experiment on the CredNet-agg data set. Note that all CredNet-agg experiments used the same true game: mean payoffs from the full data set, while each experiment's sample game was a random subsample of those observations. As a result, the experiments are not independent: similar candidate equilibria may be found across experiments, causing clusters of similar true-game regrets. Further, Table 3.2 shows that the 95% confidence bounds for CredNet-agg equilibria are quite tight, and except in the 500-sample case those bounds are rarely over-estimates. This indicates that the CredNet-agg games may have simple equilibria that can be identified using only a small number of samples, which would tend to compress the scale of the bootstrap distribution and exacerbate the aforementioned clustering of regret observations. In spite of this, my co-authors showed that the regret confidence interval from

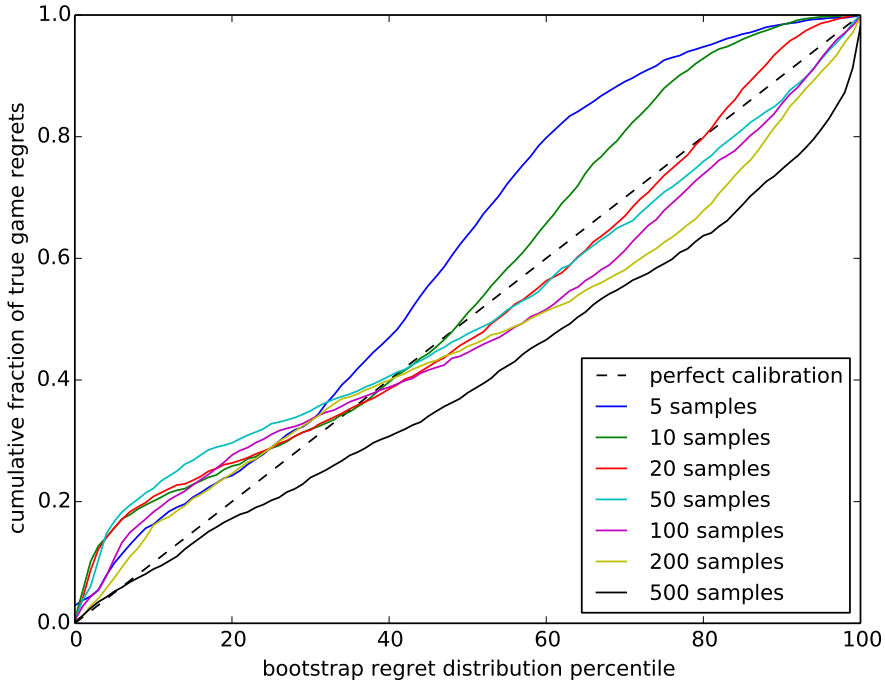


Figure 3.5: Bootstrap distributions are poorly calibrated for CredNet games. This may occur because the CredNet experiments all use the same true game.

the bootstrap can be used effectively as part of a stopping rule that makes good sampling decisions in CredNet games.

Figure 3.6 shows that the secondary implications of my hypothesis are borne out extremely well by the synthetic game experiments. It plots the regret value of the 95% confidence bounds for various levels of noise and numbers of samples in 4-player 4-strategy uSym games; results for other games are similar. Confidence intervals consistently grow tighter as the number of samples increases, with the largest gains coming from the first few samples. Moreover, whenever noise increases by an order of magnitude the confidence intervals grow wider by roughly an order of magnitude.

3.4 Conclusions

Experimental evidence demonstrates that the bootstrap method of confidence interval generation is approximately accurate for bounding the true-game regret of candidate equilibria in simulation-based games. Accuracy is lower in the experiments on credit network games than on randomly generated games, but the random game experiments may be more representative of simulation-based games in practice, due to the implausible true-game model in

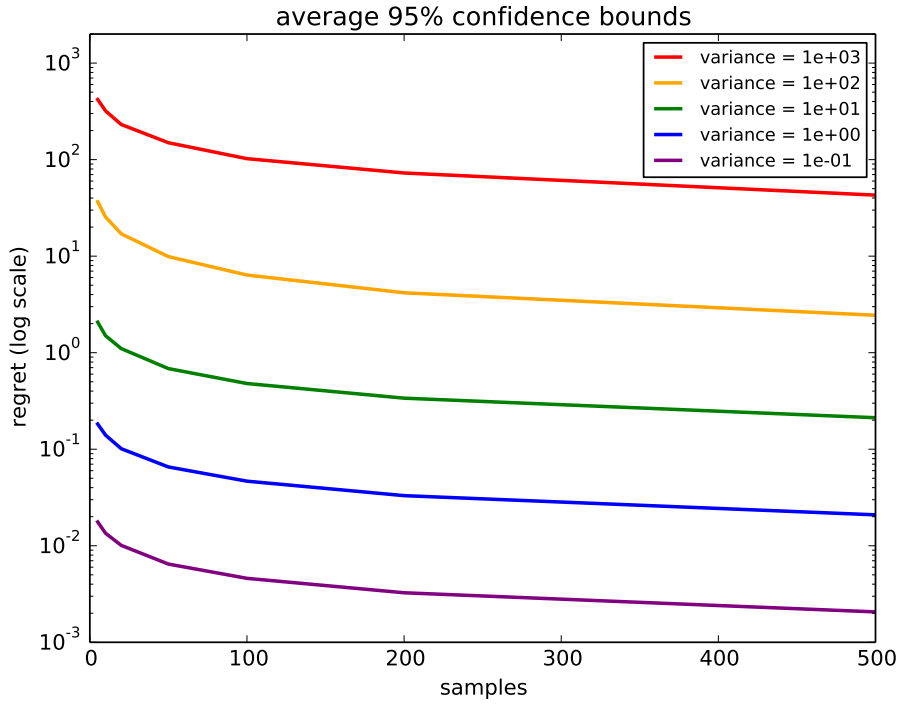


Figure 3.6: Bootstrap-estimated 95% confidence bounds for regret exhibit desirable properties of confidence intervals: they shrink with more data and grow with more noise.

the CredNet-agg domain forced by the limitations of conducting experiments with costly simulation data. Because the experiments show bootstrap confidence intervals to be accurate across multiple synthetic game classes, as well as across noise distributions and magnitudes, I recommend that practitioners of simulation-based game theory employ bootstrap methods to give regret bounds for reported equilibria.

My co-authors went on to show that the bootstrap confidence intervals could be used to control sequential sampling: running more samples until a desired confidence bound is achieved. They demonstrated that this method outperformed common rules of thumb for when to stop sampling, and that despite possible repeated testing concerns, the bootstrap method proves approximately accurate in constructing confidence intervals on regret at the conclusion of sequential sampling procedures.

This work constitutes a first systematic effort to develop and validate practical statistical methods for simulation-based game theory; future work could focus on developing theoretical foundations of applying the bootstrap to simulation-based games, and characterizing games for which the bootstrap approach is reliable. Additionally, there are other interesting properties of games, such as social welfare of equilibria, that may also benefit from using the bootstrap for statistical analysis. Other avenues of research include evalu-

ating different bootstrap designs, and using information obtained through the bootstrap to guide more sophisticated sampling, such as profile exploration [Jordan *et al.*, 2008].

CHAPTER 4

Deviation-Preserving Reduction

In principle, an environment simulator, a set of agent strategies, and a modeling function constitute a complete description of a simulation-based game. From these elements, an analyst could simulate all profiles in the game, compute equilibria, and repeat until the methods described in Chapter 3 indicate adequate statistical confidence in our conclusions. However, in practice, simulating all profiles in a game is often infeasible. Consider for example the credit network formation game described in Chapter 2. The simulated environment had 61 agents, and a total of 32 agent strategies were explored. Constructing the full game for this environment would require simulating 2.98×10^{24} profiles. In general, a role-symmetric game has $\prod_{r \in R} \binom{N_r + |S_r| - 1}{N_r}$ profiles. Some relief from this combinatorial explosion is provided by iterative exploration of the strategy space via the inner and outer loops (Section 1.5), yet the large number of agents still poses a major challenge to enumerating all relevant profiles.

My method, *deviation-preserving reduction* (DPR), enables approximate analysis of environments with a very large number of agents while sampling only a small fraction of the profiles. DPR belongs to a family of *player reduction* techniques that approximate a game with many agents by constructing a smaller game that aggregates over those agents. Player reductions specify a set of *full-game* profiles to simulate, and a modeling function that maps payoffs from those full-game profiles to payoffs in a *reduced game* with $n_r \ll N_r$ players for some or all roles. Game-theoretic analysis of the reduced game is then viewed as holding approximately in the full game.

In what follows, I compare three player reduction methods: hierarchical, twins, and deviation-preserving. I argue that deviation-preserving reduction is more effective than its predecessors at capturing key strategic properties of the full game. I also provide experimental evidence that equilibria computed in reduced games constructed by DPR have lower regret in the full game than equilibria of reduced games constructed by other methods.

4.1 Background

Player reductions work by simulating profiles in the full game, and using payoff data observed in those simulations to fill in a reduced game with a much smaller number of players. Player reduction methods differ in terms of which full-game profiles are required, and how payoffs from those profiles map to entries in the reduced-game payoff matrix. Two player-reduction methods were proposed previously: *hierarchical reduction* (HR) [Wellman *et al.*, 2005], and *twins reduction* (TR) Ficici *et al.* [2008]. I describe each reduction method by specifying its mapping function $\mathcal{M}$, which determines how full-game payoff values are used to fill the reduced-game payoff matrix. $\mathcal{M}(\vec{s}, r, s)$ takes a reduced-game profile $\vec{s}$, a role $r \in R$, and a strategy $s \in S_r$, and outputs the corresponding full-game profile $\vec{t}$, such that $u^{reduced}(\vec{s}, r, s) = u^{full}(\vec{t}, r, s)$. This mapping implicitly defines the set of profiles required to construct the reduced game: those that result from applying $\mathcal{M}$ to every profile and strategy in the reduced game. The mapping function is, in a rough sense, the inverse of the model function $\mathfrak{M}$ that maps simulation data to a game model.

4.1.1 Hierarchical Reduction

Of the two previous methods, hierarchical reduction has been more extensively used [Cassell and Wellman, 2012; Jordan *et al.*, 2007; Schvartzman and Wellman, 2009]. HR allows the analyst to select the number of players n_r in each role of the reduced game. As originally proposed, HR applied only to fully symmetric games where n evenly divides N, but the extensions to role-symmetric games and non-divisible player counts are straightforward. The key idea of hierarchical reduction is preserving the fraction of players in a role choosing each strategy. The mapping function for HR multiplies the reduced game count for each strategy by $\frac{N_r}{n_r}$, the *reduction factor* by which the number of players in role r has been scaled. Formally, $\mathcal{M}(\vec{s}, r, s) = \left\langle \left\langle \frac{N_\rho}{n_\rho} \vec{s}_\rho \right\rangle_{\rho \in R} \right\rangle$, as long as $\frac{N_r}{n_r} \in \mathbb{Z}, \forall r \in R$. When n_r does not divide N_r, the closest full-game profile by L^1 distance is used, breaking ties such that lower-index strategies have higher counts. Note that this mapping is independent of both r and s, so payoff data for all roles and strategies in a reduced-game profile comes from a single full-game profile.

As an example, consider a full game with roles $R = \{1, 2\}$, players $N_1 = 50$ and $N_2 = 12$, and strategies $S_1 = \{A, B, C\}$ and $S_2 = \{D, E\}$. The mapping function for $HR_{5,2}$ is $\mathcal{M}(\vec{s}, r, s) = \left\langle \langle 10, 10, 10 \rangle, \langle 6, 6 \rangle \right\rangle \circ \vec{s}$, where $\circ$ denotes element-wise multiplication. For the reduced-game profile $\left\langle \langle 2, 0, 3 \rangle, \langle 1, 1 \rangle \right\rangle$, payoffs are determined by simulating the full-game profile $\left\langle \langle 20, 0, 30 \rangle, \langle 6, 6 \rangle \right\rangle$. If instead, the full game had $N_1 = 51$ and $N_2 = 11$, then element-wise multiplication by $\frac{N_r}{n_r}$ would give $\left\langle \langle 20.4, 0, 30.6 \rangle, \langle 5.5, 5.5 \rangle \right\rangle$, but L^1 distance

resolves the counts for role 1, and the tie-breaking rule comes into play for role 2, selecting $\mathcal{M}(\langle\langle 2,0,3\rangle,\langle 1,1\rangle\rangle, r, s) = \langle\langle 20,0,31\rangle,\langle 6,5\rangle\rangle$.

4.1.2 Twins Reduction

Twins reduction was originally proposed for an empirical game setting where payoffs can come from observational data, and agents' roles may be unknown. Ficici *et al.* propose learning the assignment of agents to roles via clustering, and using linear regression over observed payoff data to fill in the reduced game; I refer to this variant as TR-R. In the simulation-based setting, these steps are generally unnecessary because agent roles are specified by the environment simulator and specific profiles can be simulated on demand for better estimates of reduced-game payoffs. I therefore focus primarily on the player reduction component, TR, but run a few experiments that include TR-R. I also return to the idea of using regression to fill in reduced-game payoffs in Chapter 5.

Twins reduction creates a game with $n_r = 2$ for every role. Unlike HR, the payoffs for one reduced game profile derive from several full-game profiles. Reduced-game players view themselves as controlling a single full-game agent, their *twin* (the other player with the same role) as controlling all other agents in their role, and the pair of players for any other role each controlling half of that role's agents. Formally, $\mathcal{M}(\vec{s}, r, s) = \langle\langle \vec{f}_\rho\rangle_{\rho\in R}\rangle$, where

$$\vec{f}_\rho = \begin{cases} (N_\rho - 1)(\vec{s}_\rho - \hat{s}) + \hat{s} & : \rho = r \\ \frac{N_\rho}{2}\vec{s}_\rho & : \rho \neq r \end{cases}$$

Note that because each reduced-game role has only two players, in any profile at most two strategies are played by each role. This mapping gives non-integer counts whenever N_r is odd, but as with HR, we can accommodate odd numbers of full-game players by breaking ties in favor of lower-index strategies.

Returning to the example of a full game with roles $R = \{1, 2\}$, players $N_1 = 50$ and $N_2 = 12$, and strategies $S_1 = \{A, B, C\}$ and $S_2 = \{D, E\}$, consider the reduced-game profile $\vec{s} = \langle\langle 1,0,1\rangle,\langle 2,0\rangle\rangle$. Unlike HR, the TR mapping function depends on r and s:

$$\mathcal{M}(\vec{s}, 1, A) = \langle\langle 1,0,49\rangle,\langle 12,0\rangle\rangle$$
$$\mathcal{M}(\vec{s}, 1, C) = \langle\langle 49,0,1\rangle,\langle 12,0\rangle\rangle$$
$$\mathcal{M}(\vec{s}, 2, D) = \langle\langle 25,0,25\rangle,\langle 12,0\rangle\rangle$$

Each distinct strategy gets its payoff from a unique full-game profile, meaning that up to $2|R|$ distinct full-game profiles contribute to the payoffs of a single reduced-game profile.

However, the reduced game remains role-symmetric as each player faces the same incentives as its twin.

4.1.3 Comparison of Hierarchical and Twins Reductions

The reasoning behind hierarchical reduction is that while payoffs vary with the number of agents playing each strategy, they can often be expected to do so smoothly. Kearns and Mansour [2002] formalize a related condition called *bounded influence* to define a class of compactly representable and solvable games. Even though it is easy to construct games that violate this assumption, in many natural settings with a large number of agents, it is reasonable to expect that payoffs vary smoothly with the number of agents using each strategy, and a coarse summarization of their actions could suffice for analysis.

However, hierarchical reduction fails to capture the most critical information for identifying full-game Nash equilibria. In a Nash equilibrium, no individual agent can gain by deviating to another strategy; reduced games constructed by HR contain no information about full-game unilateral deviations. In an equilibrium of $HR(\Gamma)$, no $\frac{N_r}{n_r}$-agent coalition can gain by all deviating to the same strategy. The conditions for reduced-game equilibria have very little to do with those for full-game equilibria, so only in very special cases can the two be expected to correspond.

Consider the general form of the credit network formation games described in Chapter 2. One can construct relatively simple examples, where reductions ought preserve equilibrium analysis, but where HR performs very poorly. First, in cases where defaults are likely, the full game tends to have an empty-network equilibrium. However, if transactions are relatively profitable, a group of agents deviating to issue credit simultaneously can create a network that is dense enough to be profitable in expectation. The empty network will therefore not be an equilibrium of the reduced game, where each player controls the actions of a large number of full-game agents. On the other hand, if transaction values are low and information about default probabilities is noisy, we might find spurious trade-based equilibria in the HR game. In such settings, most agents need to issue credit for the network to be dense enough to be profitable, so if a large number of agents deviated to issue no credit, the network would fail to route most transactions and the reduced-game player controlling them would be worse off. However, in the full game, a single agent might be able to get away with free-riding, meaning that the reduced-game equilibrium has a beneficial deviation in the full game.

The natural solution to this problem is to incorporate information about the value of unilateral agent deviations into the payoffs of the reduced game. Twins reduction takes a

first step in this direction, correctly capturing individual agents' incentives to deviate whenever their role is playing a single pure strategy. In fact, TR perfectly captures any full-game role-symmetric pure-strategy equilibria, as demonstrated by the following proposition.

Proposition 4.1
A role-symmetric profile[1] $\vec{t} = \left\langle \langle \vec{t_r} \rangle_{r \in R} \right\rangle$ *is a pure-strategy Nash equilibrium of* $TR(\Gamma)$ *if and only if the role-symmetric profile* $\vec{f} = \left\langle \langle \lceil \frac{N_r}{2} \rceil \vec{t_r} \rangle_{r \in R} \right\rangle$ *is a pure-strategy Nash equilibrium of* Γ.

Proof. $\vec{t}$ is a Nash equilibrium of $TR(\Gamma)$ iff for all roles $r \in R$ we have for all strategies s played in $\vec{t_r}$ and all deviations $s' \in S_r$, that $u^{TR}(\vec{t}, r, s) \geq u^{TR}(\vec{t} - \hat{s} + \hat{s'}, r, s')$. For a twin-symmetric profile, the mapping function gives the same full-game profile for every strategy, namely for each such r and s, we have $\mathcal{M}(\vec{t}, r, s) = \vec{f}$. For a deviating profile, the twins reduction mapping gives $\mathcal{M}(\vec{t} - \hat{s} + \hat{s'}, r, s') = \vec{d}$, where $\vec{d} = \left\langle \langle \vec{d_\rho} \rangle_{\rho \in R} \right\rangle$ is defined as follows:

$$\vec{d_\rho} = \begin{cases} (N_\rho - 1)\hat{s} + \hat{s'} & : \rho = r \\ \vec{f_\rho} & : \rho \neq r \end{cases}$$

So for each such $\vec{d}$, we have $u(\vec{f}, r, s) \geq u(\vec{d}, r, s')$. The set of $\vec{d}$ profiles is precisely the set of single-agent deviations from $\vec{f}$ in the full game, so whenever $\vec{t}$ is a Nash equilibrium of $TR(\Gamma)$, we know that $\vec{f}$ is a Nash equilibrium of Γ. □

Despite its ability to capture role-symmetric pure-strategy equilibria, twins reduction is generally ineffective at identifying mixed-strategy Nash equilibria of the full game. Because each reduced-game role has only two players, profiles with more than two strategies for the same role are entirely unrepresented. This makes reduced-game expected value estimates for mixed strategies with more than two strategies in the support almost entirely worthless. Even mixtures with support-size of two will have reduced-game expected values that are linear combinations of profiles with N_r or $N_r - 1$ agents playing the same strategy. This will give reasonable estimates of full-game expected values only if payoffs are linear in the number of players choosing each strategy.

Because a twins reduction game always has two players per role, the size of the full-game profile set that must be simulated depends only on the number of roles and strategies: $|\vec{S}^{TR}| = \sum_{r \in R} |S_r|^2 \prod_{\rho \in R \setminus \{r\}} \frac{|S_\rho|(|S_\rho|+1)}{2}$, which simplifies to $|\vec{S}^{TR}| = |S|^2$ in a symmetric game. Under hierarchical reduction, the one-to-one mapping between full-game and reduced-game profiles means that the set of full-game profiles required has size

[1] Ficici *et al.* [2008] refer to *twin-symmetric equilibria*, because they only consider reduced game roles with exactly two players. They do not state this proposition. In their setting, clustering and regression are applied before reduction, introducing extra errors that prevent capturing full-game equilibria exactly.

$|\vec{S}^{HR}| = \prod_{r \in R} \binom{n_r + |S_r| - 1}{n_r}$, or in a symmetric game $|\vec{S}^{HR}| = \binom{n+|S|-1}{n}$. Both reductions require a number of profiles that is exponential in the number of roles, but TR grows quadratically in $|S|$ and fixes n, while HR games grow exponentially in the smaller of n and $|S|$. HR lets the analyst select the reduced-game player counts n_r to trade off simulation time and approximation accuracy.

4.2 The Deviation-Preserving Reduction Method

Deviation-preserving reduction is designed to capture the best aspects of both hierarchical reduction and twins reduction. DPR achieves sensitivity to unilateral deviation by borrowing from TR the idea that a deviation by a reduced-game player should correspond closely to a deviation by a full-game agent. It achieves better approximation of mixed-strategy equilibria by borrowing HR's aggregation of agents and adjustable granularity. In a DPR game, each player views itself as controlling a single full-game agent, but views the profile of *opponent* strategies in the reduced game as an aggregation of all other agents in the full game. Formally, we can represent the DPR mapping as follows: $\mathcal{M}(\vec{s}, r, s) = \left\langle \langle \vec{f}_\rho \rangle_{\rho \in R} \right\rangle$, where

$$\vec{f}_\rho = \begin{cases} \frac{N_\rho - 1}{n_\rho - 1}(\vec{s}_\rho - \hat{s}) + \hat{s} & : \rho = r \\ \frac{N_\rho}{n_\rho}\vec{s}_\rho & : \rho \neq r \end{cases}$$

Note that the other-role component is identical to HR, and that the same-role component holds out one deviator like TR, but then aggregates the remaining agents in a manner similar to HR. As before, a slight extension is required to handle non-divisible reduced-game player counts; as with HR, the closest profile by L^1 distance is used, with ties broken in favor of the profile with more players choosing the lower-index strategy. DPR's divisibility condition on role-symmetric games is somewhat stricter that that of HR: for each role, n_r must divide N_r and $(n_r - 1)$ must divide $(N_r - 1)$ to avoid tie-breaking. For fully symmetric games, only the second criterion is required.

For an illustrative example, consider again the game with roles $R = \{1, 2\}$, players $N_1 = 25$ and $N_2 = 12$, and strategies $S_1 = \{A, B, C\}$ and $S_2 = \{D, E\}$. The mapping for $DPR_{5,2}$ operates on the profile $\vec{s} = \left\langle \langle 2, 0, 3 \rangle, \langle 1, 1 \rangle \right\rangle$ as follows:

$$\begin{aligned} \mathcal{M}(\vec{s}, 1, A) &= \left\langle \langle 7, 0, 18 \rangle, \langle 6, 6 \rangle \right\rangle \\ \mathcal{M}(\vec{s}, 1, C) &= \left\langle \langle 12, 0, 13 \rangle, \langle 6, 6 \rangle \right\rangle \\ \mathcal{M}(\vec{s}, 2, D) &= \left\langle \langle 10, 0, 15 \rangle, \langle 1, 11 \rangle \right\rangle \\ \mathcal{M}(\vec{s}, 2, E) &= \left\langle \langle 10, 0, 15 \rangle, \langle 11, 1 \rangle \right\rangle \end{aligned}$$

Note that the player counts for role 2, which has two reduced-game players, could belong to a TR game. If all roles have reduced-game players $n_r = 2$, then DPR is identical to TR. In addition, DPR inherits from twins reduction the ability to exactly identify role-symmetric pure strategy Nash equilibria.

Proposition 4.2
A role-symmetric profile $\vec{t} = \left\langle \langle \vec{t}_r \rangle_{r \in R} \right\rangle$ *is a pure-strategy Nash equilibrium of* any *reduced game* $DPR(\Gamma)$ *if and only if the role-symmetric profile* $\vec{f} = \left\langle \langle \lceil \frac{N_r}{2} \rceil \vec{t}_r \rangle_{r \in R} \right\rangle$ *is a pure-strategy Nash equilibrium of* Γ.

Proof. Identical to Proposition 4.1. □

Like hierarchical reduction, deviation-preserving reduction allows the analyst to specify the size of the reduced game, trading off approximation accuracy with model tractability. However, for any given reduced game size, DPR requires a larger set of full-game profiles: one for every reduced game payoff value, as opposed to HR's one-per-profile. The reduced game has $|\vec{S}^{DPR}| = \sum_{r \in R} |S_r| \binom{n_r + |S_r| - 2}{n_r - 1} \prod_{\rho \in R \setminus \{r\}} \binom{n_\rho + |S_\rho| - 1}{n_\rho - 1}$ payoff values, or in a symmetric game $|\vec{S}^{DPR}| = |S_r| \binom{n_r + |S_r| - 2}{n_r - 1}$. In symmetric games, $|DPR(\Gamma)|$ is larger than $|HR(\Gamma)|$ by a factor of $\frac{n|S|}{n+|S|-1}$; with multiple roles, the difference grows. Figure 4.1 shows the number of full-game profiles required to construct HR and DPR games with one role, five strategies, and various numbers of players. In general, reduction methods should be evaluated according to the tradeoff they provide between computational resources required and accuracy of analysis. DPR is therefore preferable only if it can give better analysis results as a function of the number of full-game profiles required; performing better as a function of reduced-game size is insufficient.

Many of the profiles simulated to construct a deviation-preserving reduction game are quite similar. Consider $\mathcal{M}(\left\langle \langle 2, 0, 3 \rangle, \langle 1, 1 \rangle \right\rangle, 1, C) = \left\langle \langle 12, 0, 13 \rangle, \langle 6, 6 \rangle \right\rangle$ from above and $\mathcal{M}(\left\langle \langle 3, 0, 2 \rangle, \langle 1, 1 \rangle \right\rangle, 1, A) = \left\langle \langle 13, 0, 12 \rangle, \langle 6, 6 \rangle \right\rangle$ from the same reduced game. These profiles differ by the deviation of a single role-1 agent between strategies A and C, both of which are played by many other agents. The basic assumption made when aggregating agents—that payoffs vary smoothly in the number of agents playing each strategy—implies that these profiles should have very similar payoffs, suggesting that we could get away with simulating only one of the two.

I therefore also tested a variant on DPR, called DPR' that arbitrarily selects one profile out of each such set of similar profiles, reducing the total number required. Whenever DPR would prescribe simulating several full-game profiles that differ by one strategy and no strategy is played by exactly one agent, DPR' requires only one. As with the tie-breaking rules for non-divisible profiles, DPR' uses the full-game profile with more agents

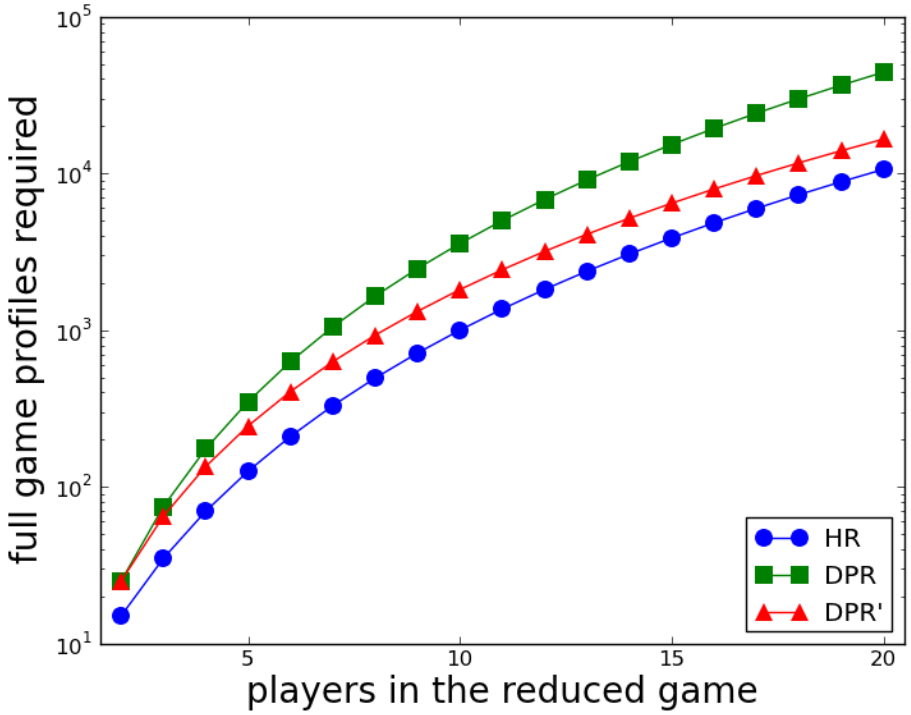


Figure 4.1: Number of full-game profiles required to construct reduced games (log scale), for $R = \{0\}$, and $|S_0| = 5$.

playing the lower-index strategy. In the example above, $\mathcal{M}(\langle\langle 3,0,2\rangle,\langle 1,1\rangle\rangle, 1, A) = \mathcal{M}(\langle\langle 2,0,3\rangle,\langle 1,1\rangle\rangle, 1, C) = \langle\langle 13,0,12\rangle,\langle 6,6\rangle\rangle$. Constructing a symmetric DPR' game takes a total of $\frac{|S|^2+n-2}{|S|+n-2}\binom{n+|S|-2}{n-1}$ profiles. DPR' always requires strictly more full-game profiles than HR, but as long as $n_r > 2$ for some role, DPR' requires strictly fewer profiles than DPR. When $n_r = 2$ for all roles, DPR' is also equivalent to TR. The red curve in Figure 4.1 shows the growth of DPR'.

4.3 Validation Experiments

To demonstrate the effectiveness of deviation-preserving reduction as a method for approximating large simulation-based games, I employ a set of validation experiments. While player reduction can be applied to any role-symmetric game, I focus my experiments, as in Chapter 3, on fully symmetric games. Because symmetric games have only one role, I refer to N, n, and $|S|$ in place of N_r, n_r, and $|S_r|$, and also subscript reductions with the number of players in the reduced game, for example HR_4 for a 4-player hierarchical-reduction game. The goal of a player reduction is to replace a full game that is too large

to effectively analyze with a more manageable reduced game. Comparing across reduction methods thus requires evaluating how well analysis that was performed on a reduced game translates back to the full game. This presents a problem, in that full games of interest for reduction methods are by definition far too big to effectively analyze. I therefore perform experiments on medium-size full games with $N = 12$, $|S| = 6$, giving $|\vec{S}| = 6188$ as well as $N = 100$, $|S| = 2$, giving $|\vec{S}| = 101$. These games are small enough to be represented fully and admit Nash equilibrium computation, but large enough that reduced games of varying size can be constructed.

Player reductions can potentially support a variety of game-theoretic analyses, but of paramount importance is identifying (role-) symmetric mixed-strategy ϵ-Nash equilibria. Successful player reduction methods will construct reduced games where the approximate equilibria of the reduced game are approximate equilibria of the full game, though necessarily with larger ϵ. My primary metric for comparing among reduction methods is the regret in the full game of approximate equilibria computed in the reduced game. Analysts often draw conclusions from the set of strategies supported in equilibrium, and to a lesser extent, the probabilities with which those strategies are played. It is therefore better for reduced games to give equilibria with similar distributions to full-game equilibria in terms of strategies supported and their probabilities. Finally, it can be helpful to identify dominated strategies in simulation-based games, so a reduction that more-accurately represents full-game dominance relationships is preferable.

As discussed in Sections 4.1.3 and 4.2, HR and DPR require different numbers of full-game profiles. My experiments therefore tested the hypothesis that DPR outperforms HR relative to the number of profiles simulated, not simply relative to the size of the reduced game. Because TR requires a fixed set of profiles, the relevant question is whether adjusting the size of the reduced game can help: I hypothesized that DPR analyses would improve when more data was added by increasing the number of players in the reduced game. Twins reduction with regression can use any set of profiles, so experiments involving TR-R varied the size of the input set over a range similar to that required to construct other reduced games.

4.3.1 Regret of Reduced Game Equilibria

The first set of experiments explores full-game regret of reduced-game equilibria. These experiments use three classes of 12-player 6-strategy games: congestion games, local-effect games, and credit network games. In a congestion game [Rosenthal, 1973], agents select a fixed-size subset of available facilities and payoffs are decreasing in the number of agents

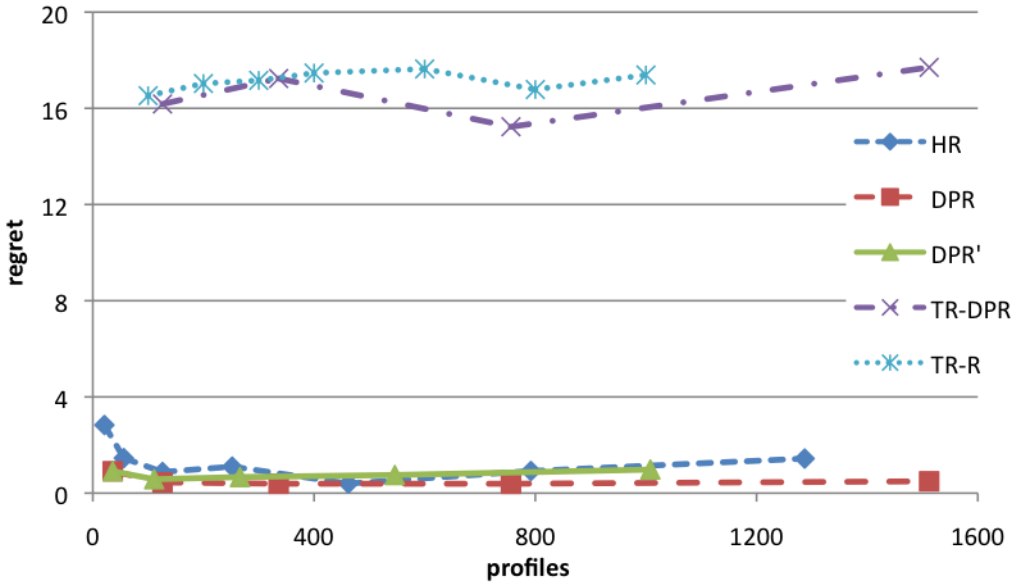
choosing a facility. Unlike Chapter 3 (5 agents, 3 facilities, 1 selected), here I generated congestion games where agents select two out of four possible facilities, and each facility has a quadratic cost in the number of agents selecting it; agents' utilities sum over the congestion costs incurred. Parameters of the quadratic cost functions are drawn randomly to generate congestion game instances. Local-effect games [Leyton-Brown and Tennenholtz, 2003] have a directed graph over actions and each action's payoff is a function of the number of agents choosing it and adjacent actions. I constructed local-effect games by generating random graphs and random quadratic functions for the cost of each strategy. In addition, for each pair of adjacent strategies, I generated a random quadratic payoff function that was not necessarily negative. For each of these classes, I generated 250 random game instances. The credit network games differ from those analyzed in Chapter 2 in that only 12 agents participated (for tractability), and only trade-based strategies were used (to avoid trivial no-trade or central currency equilibria). 250 random credit network game instances were generated by collecting 100 samples of each profile, and then for each game, selecting the outcome of one simulation of every profile uniformly at random.

For each random full game, I constructed the following set of reduced games:

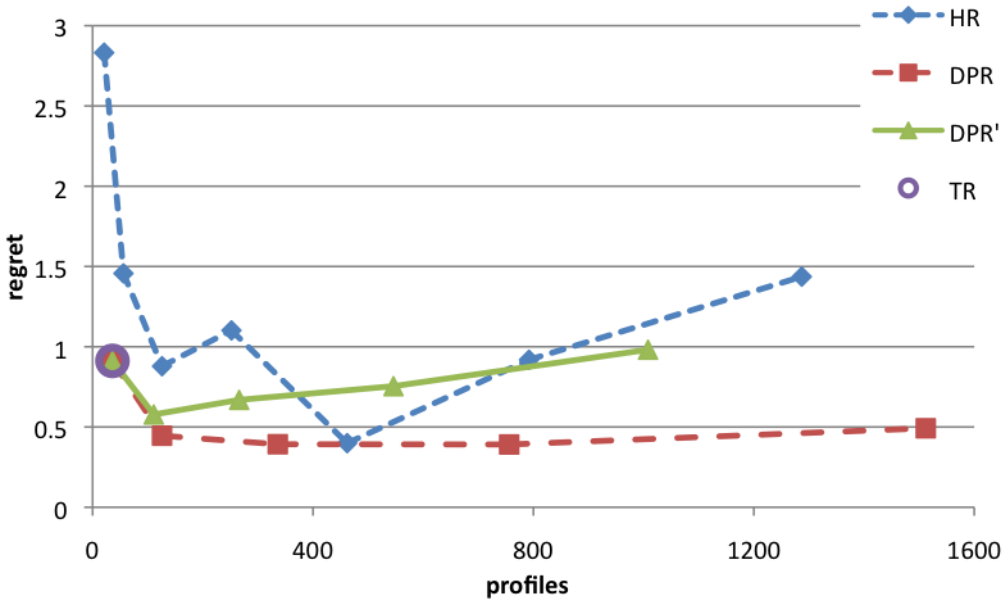
- HR: 2–8 players
- TR (fixed size), and TR-R: various input set sizes
- DPR: 3–6 players ($DPR_2 \equiv TR$)

For TR-R, I followed Ficici *et al.* [2008] in sampling profiles from the full game uniformly at random, and I chose various sample set sizes ranging over the number of profiles required to construct the various other reduced games. In each reduced game, I computed a single symmetric mixed-strategy ϵ-Nash equilibrium using replicator dynamics, initialized from a uniform mixture. I then computed the full-game regret of each reduced game equilibrium. Regret values averaged over each set of 250 reduced games are shown in Figures 4.2 and 4.3.

The first finding of note is that twins reduction performs very poorly with linear regression. The top line in Figure 4.2a shows the regret of equilibria found in TR-R games with random sampling of profiles, which is an order of magnitude worse than the HR, TR, DPR, and DPR'. Two observations led me to try the method labeled TR-DPR: first, that sampling profiles according to uniform agent play leads to a very low likelihood of observing payoffs for profiles where almost all agents play the same strategy, and these are exactly the profiles whose payoffs the regression estimates. Second, simulating all the profiles for DPR or DPR' makes available a substantial amount of payoff data that goes unused in



(a) all reduction types



(b) rescaled to exclude regression-based twins reductions

Figure 4.2: Average full-game regret of reduced-game equilibria in local-effect games. $N = 12$, $|S| = 6$, $2 \leq n \leq 8$.

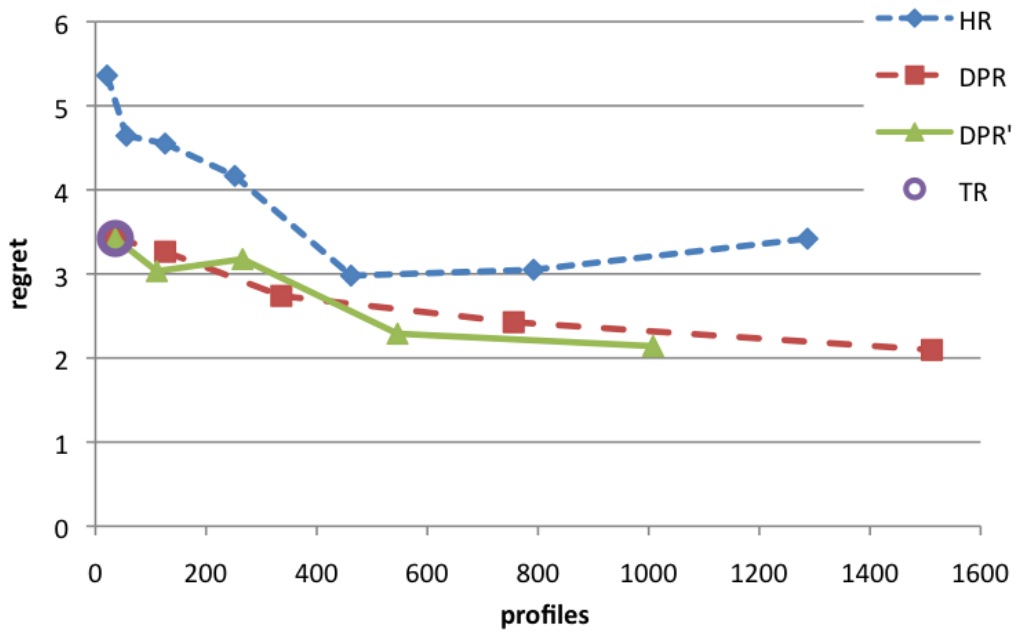


Figure 4.3: Average full-game regret of reduced-game equilibria in credit network games. $N = 12$, $|S| = 6$, $2 \leq n \leq 8$.

constructing the reduced game. I therefore thought to try using all of the profiles simulated for the deviation-preserving reduction as input to the linear regression of TR-R. As is clear from Figure 4.2a, this improves very little on random sampling.

In retrospect, it is not particularly surprising that approximating payoffs by linear regression performs so poorly: all of three classes of test games and most games requiring simulation have nonlinear payoffs. A better regression model could potentially alleviate this problem, and I return to this and related questions in Chapter 5. I also ran TR-R and TR-DPR on each of the other game classes, but the results are similarly poor, and are excluded from subsequent figures.

Figures 4.2b, 4.3, and 4.4 show that deviation-preserving reduction outperforms hierarchical reduction and twins reduction in a wide variety of settings. In 12-agent, 6-strategy local-effect games, DPR is clearly better than HR, but the comparison to DPR' is less conclusive. It seems initially surprising that hierarchical reduction would perform worse with increased reduced-game size; generally adding more data should improve the model. Note, however, that in the reduced games where HR performance degrades ($n \in \{5, 7, 8\}$ in Figure 4.2b, $n \in \{7, 8\}$ in Figure 4.3), n does not divide $N = 12$. This also leads to the observation that because 11 is prime, the deviation-preserving reduction never has the advantage of $n - 1$ dividing $N - 1$, and yet consistently performs well. Results for the 12-agent, 6-strategy credit network game appear in Figure 4.3. Here again, DPR and DPR'

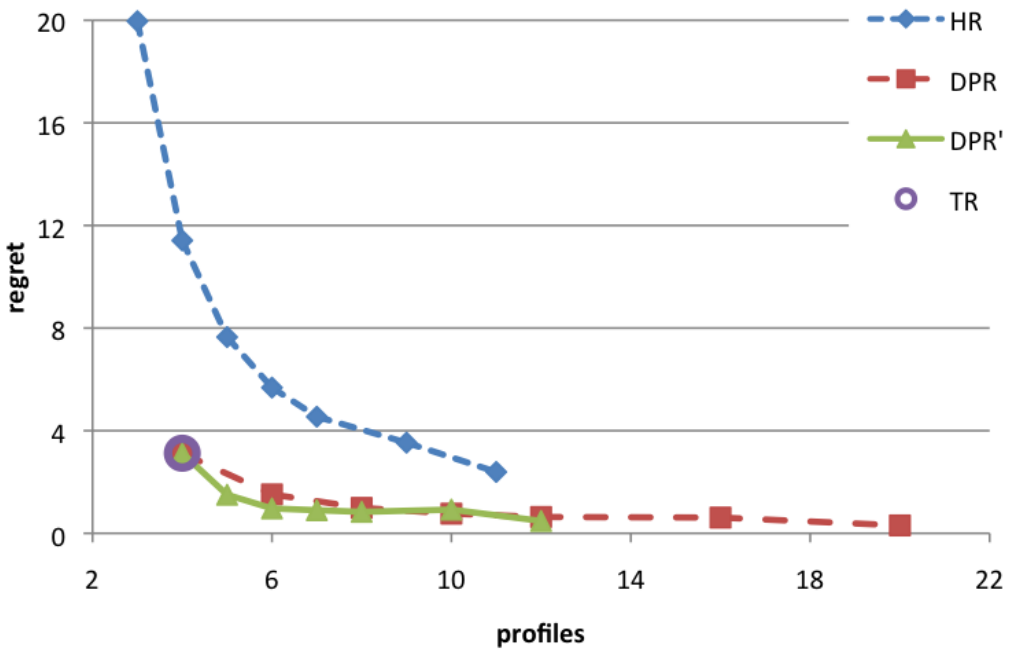


Figure 4.4: Full-game regret of reduced-game equilibria in congestion games. $N = 100$, $|S| = 2, 2 \leq n \leq 10$.

perform similarly, and better than HR. The results from 12-agent, 6-strategy congestion games (not shown) are broadly similar.

In a first attempt to understand effect of very substantial player reductions, I created an additional set of congestion games with $N = 100$ and $|S| = 2$, and performed the same experiments. The results in Figure 4.4 show clear separation between hierarchical reduction and both variants of deviation-preserving reduction, suggesting that the difference DPR and DPR' may shrink with the size of the full game. Further, DPR with $n > 2$ outperforming TR with $|S| = 2$ indicates that its advantage stems in part from better representation of non-linear payoff functions, not merely from representing interactions among more than two strategies.

Across all game classes and sizes examined (including those not shown), for any given HR game size, there is a DPR game that provides more accurate analysis on average, while simulating fewer full-game profiles. Virtually all of these differences are significant at $p < 0.05$; the only exceptions are 4-player DPR versus 6-player HR in the 12-player congestion game (Figure 4.4) and 12-player credit network game (Figure 4.3). In addition, DPR_3 outperforms TR across all game classes; the difference is significant at $p < 0.05$ in all cases except the credit network game. The difference between DPR_4 and TR is significant in all cases.

	credit network		congestion		local-effect	
n	HR	DPR	HR	DPR	HR	DPR
2	3.72	1.49†	1.68	0.39†	2.72	0.45†
3	3.72	1.64†	0.99*	0.17†*	1.04*	0.20†*
4	3.72	1.60†	1.05	0.10†*	0.98	0.12†*
5	1.98*	1.54†	1.10	0.08†	1.01	0.09†
6	1.19†*	1.39	0.98	0.05†	0.85*	0.08†

Table 4.1: Average number of strategies by which reduced- and full-game NE support sets differ. * indicates significant difference between n and $n-1$; † indicates significant difference between DPR_n and HR_n.

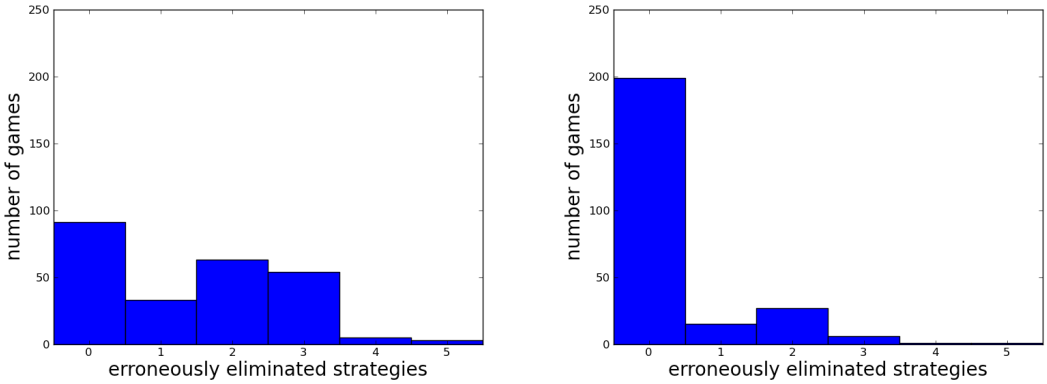
	credit network		congestion		local-effect	
n	HR	DPR	HR	DPR	HR	DPR
2	0.713	0.703	0.435	0.069†	0.503	0.128†
3	0.764*	0.690†	0.141*	0.039†*	0.154*	0.064†*
4	0.860	0.640†	0.117*	0.022†*	0.117*	0.037†*
5	0.643*	0.641	0.141	0.019†*	0.144	0.027†*
6	0.467†*	0.564*	0.099*	0.018†	0.088*	0.026†

Table 4.2: Average L^2 distance between full- and reduced-game NE distributions. * indicates significant difference between n and $n-1$; † indicates significant difference between DPR_n and HR_n.

4.3.2 Comparison to Full-Game Equilibria

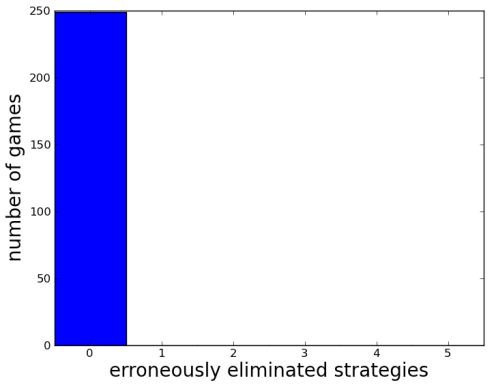
I also compared reduced-game equilibria under HR and DPR to equilibria computed directly in the corresponding 12-agent, 6-strategy full games. The comparisons use two metrics: similarity of support sets, and L^2 distance between distributions. I computed one symmetric mixed-strategy ϵ-Nash equilibrium in each full game and another in each reduced game by running replicator dynamics initialized with a uniform mixture. Table 4.1 shows mean differences in support sets of corresponding full- and reduced-game equilibria. I consider a strategy to be in the support of a symmetric ϵ-Nash equilibrium if it is played with probability 0.01 or greater and compute the size of the symmetric difference between the support sets. In nearly all cases, support sets of DPR_n match those of full-game equilibria significantly ($p < 0.05$) better than both HR_n and HR_{n+1}. In addition, for congestion games and local-effect games, $DPR_{>2}$ significantly outperforms TR, whereas in credit network games, there is no significant difference between TR and DPR.

Table 4.2 shows the mean L^2 distance between the vector representations of the same full- and reduced-game equilibria. For congestion and local-effect games equilibria of DPR games match full-game equilibria significantly better than those of HR and TR



(a) HR_2 (21 profiles)

(b) HR_4 (126 profiles)



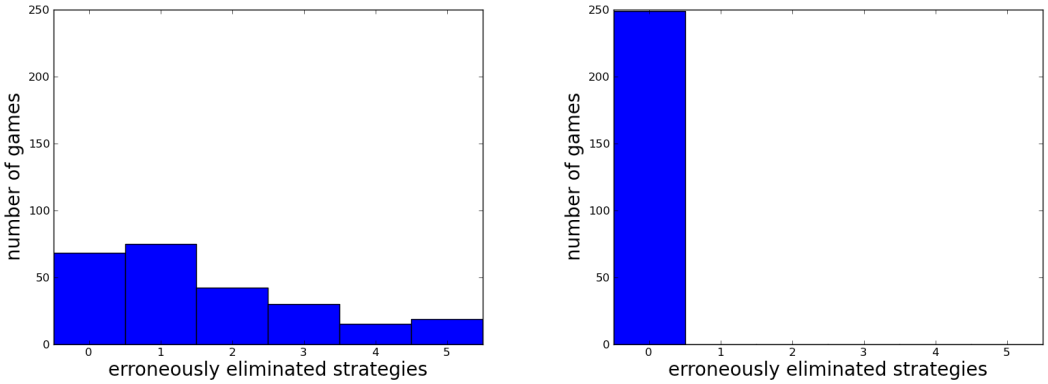
(c) $TR \equiv DPR_2 \equiv DPR'_2$ (36 profiles)

Figure 4.5: Histograms showing the number of strategies surviving iterated elimination of dominated strategies in full but not reduced congestion games. $N = 12$, $|S| = 6$, 250 random games. $TR \equiv DPR_2$ outperforms HR, sampling far fewer profiles.

games. In the credit network games, HR and DPR perform similarly, and DPR's advantage over TR is smaller.

4.3.3 Dominated Strategies

Another useful operation in the analysis of simulation-based games is to check for dominated strategies. A dominated strategy is one that no agent should ever play because there is an alternative strategy that is at least as good in response to any profile of opponent strategies. I ran experiments on the aforementioned 12-agent, 6-strategy congestion and credit network games, comparing the set of strategies that remain after iterated elimination of strictly dominated strategies in the full game against those that remain in 2, 4, and 6-player reduced games. DPR and DPR' produce very similar results, and both improve over hierarchical and twins reduction. Figures 4.5 and 4.6 show histograms of the number



(a) $TR \equiv DPR_2 \equiv DPR'_2$ (36 profiles) (b) DPR'_4 (336 profiles)

Figure 4.6: Histograms showing the number of strategies surviving iterated elimination of dominated strategies in full but not reduced credit network games. $N = 12$, $|S| = 6$, 250 sample games. DPR'_4 avoids the aggressive elimination occurring in TR.

of strategies eliminated in reduced games but not eliminated in full games.

In congestion games (Figure 4.5), twins reduction and both forms of deviation-preserving outperform hierarchical reduction, eliminating fewer strategies in the reduced game that survive in the full game, even when hierarchical reduction samples vastly more profiles. These congestion games often exhibit dominated strategies in the full game, but strategies almost never survived in a reduced game that were dominated in the full game.

In credit network games (Figure 4.6), no strategies are dominated in the full game, but in the twins reduction game, many strategies are eliminated. Moving to DPR_4 or DPR'_4 solves this problem almost entirely. These experiments also confirm that for all reduction types, increasing the number of players in the reduced game reduces the number of strategies erroneously found to be dominated.

4.4 Conclusions

Deviation-preserving reduction combines the most appealing aspects of hierarchical reduction and twins reduction. It also performs better than both prior methods experimentally: equilibria from DPR games have lower full-game regret and more closely resemble full-game equilibria, even when sampling fewer full-game profiles. In addition, performing iterated elimination of dominated strategies on deviation-preserving reduction games stays more faithful to the full game compared to other player reductions. The alternative DPR' formulation performs reasonably well in the same tests. The simulation savings from DPR' are greatest when the reduced game has many players but few strategies, so DPR' may

prove useful in such cases. Though it may not be obvious how to choose between DPR and DPR', the evidence is quite compelling that deviation-preserving reduction is the best available player reduction method for analyzing large simulation-based games.

Since the paper proposing deviation-preserving reduction [Wiedenbeck and Wellman, 2012] first appeared, DPR has largely supplanted HR as the reduction method of choice for simulation-based game modeling. Projects applying deviation-preserving reduction have included my own work on strategic formation of credit networks described in Chapter 2, and several studies related to market-making and high-frequency trading, most recently that of Wah and Wellman [2015].

A useful direction for future work in this domain would be investigating the sorts of games that are most amenable to reduction. Player reduction relies fundamentally on assumptions about payoff smoothness, but these assumptions have not been characterized formally. It seems likely that certain properties of role-symmetric games could be established that suggest whether or how player reduction should be applied. In general, simulation-based game theory is applied when payoffs are not well understood, so such properties would be most valuable if they could be inferred from simulation data.

The work I describe in Chapter 5 can also be viewed as an extension of player reduction. Those methods fundamentally solve the same problem: facilitating approximate analysis of role symmetric games with many players without enumerating the full normal form representation of the game. It may turn out to be the case that constructing a reduced game as an intermediate representation is not the only or even the best way of achieving this goal.

CHAPTER 5

Learning Game Models

As described in Chapter 4, deviation-preserving reduction improves on prior player reduction methods, enabling higher-fidelity analysis of large simulation-based games from limited samples. My evaluation of DPR focused on the accuracy of game-theoretic analysis performed in the reduced game, rather than on directly comparing reduced- and full-game payoff matrices, because the purpose of any simulation-based game model is to describe equilibrium behavior in the simulated environment, and reduced games are of interest primarily as a tool for achieving this goal. Acknowledging this objective changes the problem of approximating large games from one of constructing the best reduced game to one of performing the best analysis with limited sampling, and opens the door to techniques that identify equilibria without explicitly constructing a payoff matrix.

5.1 Motivation: Limitations of Player Reduction

Viewed as tools for approximate equilibrium analysis, DPR and other reductions serve two key functions: identifying the set of profiles simulated to construct the observation set Θ, and specifying through $\mathfrak{M}(\Theta)$ how data from those profiles contributes to computation of approximate equilibria. Recall from Chapter 1 that replicator dynamics computes equilibria iteratively by evaluating the expected utility of a mixture[1] and updating the mixture relative to those expectations. Equation 1.12, reproduced below, gives the formula for expected utility of strategy s played against mixture $\vec{\sigma}$ as a sum over the game's profiles of the payoff to s weighted by the profile's probability under $\vec{\sigma}$:

$$u(\vec{\sigma}, r, s) = \sum_{\vec{s} \in \vec{S}} P(\vec{s} - \hat{s} \mid \vec{\sigma}) u(\vec{s}, r, s) \tag{1.12}$$

Computation of expected utilities in a reduced game can be viewed as an approximate

[1] Estimating expected utilities is also a crucial step in other equilibrium computation algorithms.

computation of expected utilities in the full game. The approximate sum is restricted to a the subset of profiles $\vec{S}^{\,DPR}$ used to construct the reduced game, and profiles $\mathcal{M}(\vec{s}, r, s)$ are re-weighted according to their probability of occurrence $P(\vec{s} - \hat{s} \mid \vec{\sigma})$ under $\vec{\sigma}$ in the reduced game:

$$u^{DPR}(\vec{\sigma}, r, s) = \sum_{\mathcal{M}(\vec{s},r,s) \in \vec{S}^{\,DPR}} P(\vec{s} - \hat{s} \mid \vec{\sigma}) u(\mathcal{M}(\vec{s}), r, s) \tag{5.1}$$

As a method for choosing profiles to simulate and providing weights for the resulting observations, deviation-preserving reduction has a couple of clear shortcomings. First, DPR makes inefficient use of simulation data. When a profile $\vec{s}$ is simulated, the simulator outputs a payoff value for every strategy $s \in \vec{s}$. However, because each reduced-game payoff comes from a distinct full-game profile, DPR uses the payoff value from only one strategy of each simulated profile. In the example from Chapter 4 with $R = \{1, 2\}$, $N_1 = 25$, $N_2 = 12$, $S_1 = \{A, B, C\}$, and $S_2 = \{D, E\}$, the mapping for $DPR_{5,2}$ gave $\mathcal{M}\left(\left\langle\langle 2, 0, 3\rangle, \langle 1, 1\rangle\right\rangle, 1, A\right) = \left\langle\langle 7, 0, 18\rangle, \langle 6, 6\rangle\right\rangle$. Simulating this profile gives payoff data for strategies A, C, D, and E, but only the data for A is used in constructing the reduced game; this profile is not in the domain of the mapping function $\mathcal{M}(\cdot, r, s)$ for any other strategy, and therefore does not contribute to expected utility estimates for other strategies. Figure 5.1 shows the fraction of simulation data wasted in various symmetric DPR games. In games with multiple roles, significantly more data is wasted. In the credit network study discussed in Chapter 2, approximately two thirds of all payoff data in θ was ignored by $\mathfrak{M}(\theta)$.

Player reductions are also also overly rigid in how they constrain the sampling process. Like other player reductions, DPR provides a mapping from reduced-game profiles to full-game profiles. To accurately estimate payoffs, each full game profile in the range of $\mathcal{M}$ is simulated repeatedly, but other full-game profiles cannot contribute to the reduced game model. As more samples of a single profile are collected, the returns to additional samples of that profile diminish, and using DPR, the only alternative to sampling the same profiles again is to increase the number of players in the reduced game. The experiments in Chapter 4 show that this generally improves accuracy, but few full-game profiles are shared across reduced-game sizes, requiring that sampling start almost from scratch in the larger reduced game. If an alternative $\mathfrak{M}$ could make use of a broader set of observations, simulation resources could probably be allocated more effectively by taking fewer samples of a larger set of profiles.

In this chapter, I dispense with the idea of fully enumerating the payoff matrix for a reduced game and develop a machine-learning-based approach to estimating expected

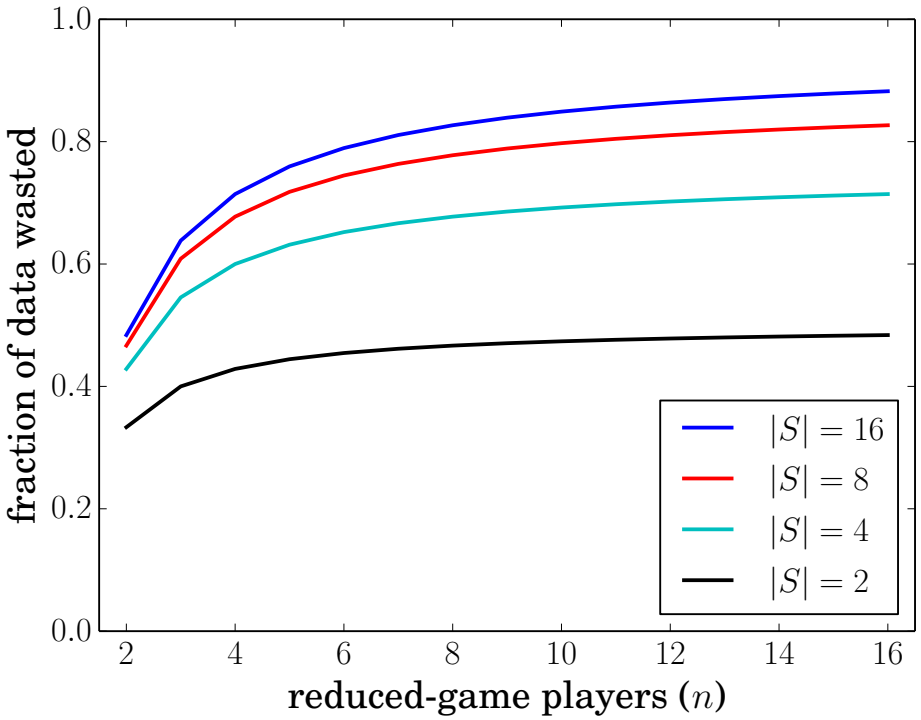


Figure 5.1: DPR uses only one strategy's payoff from each simulated profile; payoff data for all other strategies is ignored. In a symmetric game, a $\frac{(n-1)(|S|-1)}{n|S|-1}$ fraction of payoff data is ignored. This fraction is plotted for various plausible reduced-game sizes.

values of role-symmetric mixtures from simulation data. This approach uses Gaussian process regression to take arbitrary simulation data as input and learn a utility function for each strategy. It exploits the vector representation of a profile, concatenating role profiles into a vector of counts for all strategies as the independent variable input to each regression. I test several methods for estimating expected values from these regression functions. I validate the learning method experimentally, using an experimental suite that allows testing on much larger games than those in Chapter 4 and on noisy sample data. The experiments demonstrate that learning methods can outperform DPR in estimating expected values and enabling Nash equilibrium computation, but leave several open questions about the best ways to select profiles to simulate and to incorporate information about payoff variance.

5.2 Related Work

This work is the first to use machine learning to approximate arbitrary symmetric games. Vorobeychik *et al.* [2007] demonstrated the use of regression methods to learn payoff functions over continuous strategy spaces. My work can be viewed as extending their work to the domain of categorical strategies. Their learning methods relied on strategy sets that

were fully described by varying continuous parameters, such as bids in a single-unit auction and learned utility functions with respect to those parameters. My method relies on the large number of symmetric players to provide learnable structure, allowing it to work on arbitrary strategy sets. However, because utility functions are learned separately for each strategy, my method provides no generalizability to unexplored strategies.

Some researchers have investigated explicitly learning compact representations of games. Duong *et al.* [2009] developed a method for detecting graphical structures representing independences among players' payoffs. When present, such independences may allow significantly compacted game representations, but may not exist, especially in symmetric games, where independence of players' payoffs would imply an environment devoid of game-theoretic interactions. Honorio and Ortiz [2015] likewise learn graphical game models, in their case from observations of play rather than payoffs, using assumptions about the structure of utility functions and the way that play is generated conditional on the actual payoff function. Gao and Pfeffer [2010] also exploit rationality assumptions in their approach, which uses weighted constraint satisfaction to combine both payoff and play observations in learning game models.

The most relevant prior work is the player reduction methods discussed in Chapter 4, as they solve the same fundamental problem of exploiting symmetry and smoothness in the utility functions to approximate large games from few samples. As the best of the player reduction methods, DPR is the standard against which I evaluate the learning methods. In the paper introducing twins reduction, Ficici *et al.* [2008] propose using linear regression to learn the payoffs of a TR game from observation data. The experiments in Chapter 4 showed that this approach worked poorly on simulation-based games, but this could be attributable to the inflexibility of either linear regression or twins reduction. This method inspires one of the techniques I test for estimating expected values: constructing DPR games from the learned payoff functions.

5.3 Learning and Applying Payoff Models

To learn a payoff model for a large simulation-based game, I perform a separate regression to estimate utilities $\hat{u}_s$ for each strategy. The independent variable input to each regression is a vector representation of a profile $\vec{s}$ with dimension $\sum_{r \in R} |S_r|$; for strategy s, the dependent variable input is the payoff to strategy s in the corresponding profile $u(\vec{s}, r, s)$. I use Gaussian process regression [Rasmussen and Williams, 2006] to perform this learning. Gaussian process regression is a locally-weighted method in which the estimated value at any point is most heavily influenced by nearby training points. As such, it does not produce

a closed-form estimate of $\hat{u}_s$, but rather permits queries at unknown points $\hat{u}_s(\vec{s})$. Gaussian process regression relies on a Gaussian process prior over possible regressor functions; this prior can be viewed as a particular formalization of the implicit assumption behind player reduction methods that utility functions are "smooth."

For every strategy $s \in \bigcup_{r \in R} S_r$ in any role, I run a regression to learn a mapping $\hat{u}_s$ from profiles to payoffs that approximates the utility function $u(\cdot, r, s)$, where $s \in S_r$. For any given profile $\vec{s}$, the independent variable input to the regression for strategy s is the concatenation of the role-profile vectors in $\vec{s} - \hat{s}$. Returning to the example of profile $\big\langle\langle 7, 0, 18\rangle, \langle 6, 6\rangle\big\rangle$, the input to the regression for $\hat{u}_A$ is the profile $\big\langle 6, 0, 18, 6, 6\big\rangle$, while the regression for $\hat{u}_E$ receives the profile $\big\langle 7, 0, 18, 6, 5\big\rangle$. The example profile has no data for strategy B and is therefore excluded from the regression for $\hat{u}_B$.

Figure 5.2 gives an overview of the learning method. Given a set of profile observations Θ, a data set $\Theta_s = \{\theta \in \Theta \mid s \in \theta\}$ is constructed for each strategy s, containing all observations of payoffs to strategy s in profiles where s is played by at least one agent. For each strategy, I then run Gaussian process regression on Θ_s to learn $\hat{u}_s$. The regression uses the payoff data Θ_s to perform a Bayesian update to its Gaussian process prior over possible functions. The resulting Gaussian process process posterior does not have a known closed form, but rather can be queried at any profile for mean and variance estimates. Gaussian process regression was selected for its flexibility; polynomial regression or other methods that output an explicit functional form could simplify subsequent analysis, but may not be adequately expressive. In environments where simulation-based game modeling is necessary, the analyst is unlikely to have advance knowledge of appropriate functional forms for u_s.

Running Gaussian process regression on each of the observation sets Θ_s gives a model that can be queried for estimates of $u(\vec{s}, r, s)$ for any profile, role, and strategy. However, finding Nash equilibria (and other analyses, like computing social welfare) requires estimating the expected value to a single agent playing strategy s against opponents playing a role-symmetric mixture $\vec{\sigma}$. Recall from Equation 1.12 (reproduced in Section 5.1) that this expectation is a weighted sum over all profiles with positive probability under $\vec{\sigma}$. In principle, $\hat{u}_s$ can be queried at each of these profiles, but in any game where player reduction or payoff learning is necessary, it is clearly infeasible to query the learned u_s function for every profile $\vec{s}$ in the support of a mixture $\vec{\sigma}$. If polynomial regression, or some other method that outputs a suitable functional closed-form were used in place of Gaussian process regression, this sum could in principle be approximated by an integral. However, with locally-weighted regression methods that do not yield a closed form, other methods for approximating $\hat{u}(\vec{\sigma}, r, s)$ are necessary.

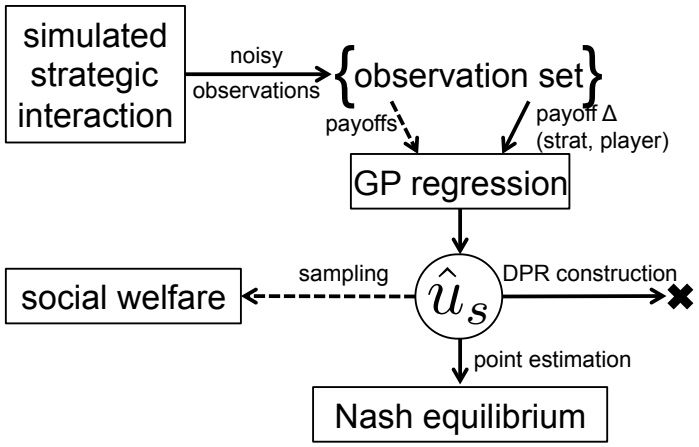


Figure 5.2: Model construction. The dependent variable input for Gaussian process regression can be either payoffs or differences from average payoffs. Expected values—used in social welfare and Nash equilibrium computation—can be estimated by sampling, point estimation, or constructing a DPR game.

5.3.1 Estimating Expected Values

I propose and test three methods for estimating $\hat{u}(\vec{\sigma}, r, s)$ from the learned utility models: *sampling*, *point estimation*, and constructing DPR games.

The sampling method draws a large number (k) of profile realizations $\vec{s}_j \sim \vec{\sigma}$ randomly according to the role-symmetric mixture. If evaluating $\vec{\sigma} = \langle\langle .2, .4, .4\rangle, \langle .7, .3\rangle\rangle$ in the example game with $N_1 = 25$ and $N_2 = 12$, each profile realization is created by drawing 25 samples from $\langle .2, .4, .4\rangle$ and 12 samples from $\langle .7, .3\rangle$ to construct each profile $\vec{s}_j$. The utility function $u_s(\vec{s}_j)$ is then queried at each profile realization and the resulting payoffs are averaged:

$$\hat{u}(\vec{\sigma}, r, s) \approx \frac{1}{k} \sum_{j=1}^{k} \hat{u}_s \left(\vec{s}_j \sim \vec{\sigma}\right). \tag{5.2}$$

This method provides an unbiased estimate of $\hat{u}(\vec{\sigma}, r, s)$ that is correct in the limit as $k \to \infty$. However, the sampling estimate requires time linear in k to compute, and gives unstable payoff estimates. That is, if expected values for the same mixture were estimated multiple times, each evaluation would give a slightly different result, potentially preventing replicator dynamics from converging. These drawbacks are acceptable when computing regret and social welfare, but finding Nash equilibria involves a large number of expected value computations, and requires stable expected value estimates to ensure convergence.

The second method, point estimation, relies on the large number of agents in the game. As the number of agents approaches infinity, the distribution over profiles under a mixture $\vec{\sigma}$ approaches a point mass where the fraction of each role's agents playing each strategy

is equal to that strategy's probability in the mixture. The point estimation method makes a single query to $\hat{u}_s$ at the point where population proportions match mixture probabilities: $\left\langle N_r - I_{s \in S_r} \right\rangle_{r \in R} \circ \vec{\sigma}$, where $I_{s \in S_r}$ is an indicator with value 1 if s belongs to S_r and 0 otherwise. In the example above, $\hat{u}(\langle\langle .2, .4, .4\rangle, \langle .7, .3\rangle\rangle, 1, A)$ is estimated by querying $\hat{u}_A$ at the point $\langle\langle 4.8, 9.6, 9.6\rangle, \langle 8.4, 3.6\rangle\rangle$. The general formula for point estimation is given by:

$$\hat{u}(\vec{\sigma}, r, s) \approx \hat{u}_s \left(\left\langle N_r - I_{s \in S_r} \right\rangle_{r \in R} \circ \vec{\sigma} \right). \tag{5.3}$$

This method is correct in the limit as $N_r \to \infty$ for all $r \in R$ (and may therefore be adequate for large finite N_r), can be evaluated quickly, and provides numerically stable expected value estimates amenable to use with replicator dynamics.

The final method for estimating $\hat{u}(\vec{\sigma}, r, s)$ constructs a DPR game from the learned models and computes expected values in that game. For each payoff value $u^{DPR}(\vec{s}, r, s)$ in a DPR game, the regression function $\hat{u}_s$ is queried at the full-game profile $\mathcal{M}(\vec{s}, r, s)$ and the result entered in the reduced-game payoff matrix. The primary advantage of this technique is that it allows a DPR game to be constructed from an arbitrary set of profile observations. The first expected value computation with this method incurs overhead of constructing the DPR game, but subsequent expected value computations are fast, and when computing equilibria, the construction step can amortized over many queries.

Gaussian process regression has a number of hyperparameters, most importantly the kernel and the characteristic length scale. I used the scikit-learn machine learning package [Pedregosa *et al.*, 2011] to perform Gaussian process regression, and initial tests with scikit-learn's default parameters were disappointing; the learned Gaussian processes did a poor job estimating expected values of symmetric mixed strategies. Choosing an appropriate kernel by 3-fold cross-validation on the payoff data and determining the length scale by maximum likelihood estimation drastically improved accuracy. This is, in retrospect, not particularly surprising, but noteworthy because cross-validation on pure-strategy observation data led to better performance evaluating mixed strategies with the aforementioned methods.

5.3.2 Computing Equilibria

As in previous chapters, I compute Nash equilibria in the learned game models using replicator dynamics. Recall from Chapter 1 that on each iteration, replicator dynamics updates the mixture in proportion to the expected value of each strategy. Any of the three methods for estimating $\hat{u}$ can be used in place of u in the replicator dynamics update in Equation 1.11, but point estimation or DPR construction are preferred over sampling because

they give stable estimates when extremely similar mixtures are evaluated, enabling replicator dynamics to converge.

Early experiments indicated success in estimating social welfare, but poor performance computing equilibria. It seemed that the regressions were learning broad trends in payoffs effectively, but failing to capture payoff differences crucial to computing regret and therefore finding Nash equilibria. To help the Gaussian processes capture payoff differences, I modified the input to the regression so that instead of learning u_s for each profile $\vec{s}$, it learns the *difference* between the payoff to strategy s and the average payoff $\bar{u}(\vec{s})$ in the profile for the role to which s belongs. I tried two variations on the payoff-difference calculation, using the strategy-weighted and player-weighted averages. For strategy-weighted average, the payoff difference u^{Δ}_{strat} is defined by

$$u^{\Delta}_{strat}(\vec{s}, r, s) \equiv u(\vec{s}, r, s) - \frac{1}{|\{s' \mid s' \in \vec{s}_r\}|} \sum_{s' \in \vec{s}_r} u(\vec{s}, r, s').$$

For player-weighted average, the payoff difference u^{Δ}_{player} is defined by

$$u^{\Delta}_{player}(\vec{s}, r, s) \equiv u(\vec{s}, r, s) - \frac{1}{N_r} \sum_{s' \in S_r} \vec{s}[r, s'] u(\vec{s}, r, s').$$

The learned payoff differences $\hat{u}^{\Delta}_s$ can be used in place of learned payoffs $\hat{u}_s$ as input to replicator dynamics. Restricting Equation 1.11 to the components for a single role r and strategy s for simplicity of exposition gives:

$$\vec{\sigma}^{\,i+1}[r, s] \propto \left(u(\vec{\sigma}^{\,i}, r, s) - \mu_r\right) \vec{\sigma}^{\,i}[r, s],$$

which, when $\hat{u}^{\Delta}$ is substituted for u, becomes

$$\vec{\sigma}^{\,i+1}[r, s] \propto \left(\hat{u}(\vec{\sigma}^{\,i}, r, s) - \frac{1}{N_r} \sum_{s' \in S_r} \vec{s}[r, s'] \hat{u}(\vec{s}, r, s') - \mu_r\right) \vec{\sigma}^{\,i}[r, s].$$

When using point estimation, the profile-average payoff can, in a rough sense, be thought of as absorbed into the minimum payoff term μ. The profile-average payoff depends on the profile, but under point estimation only one profile is queried for each expected value calculation, so subtracting the mean payoff simply lowers μ, causing replicator dynamics to update more slowly. Importantly, if this update converges, all strategies played with positive probability have equal estimated expected value, making $\vec{\sigma}$ a Nash equilibrium, subject to the inaccuracies of the Gaussian process and point estimation methods. Alongside payoff

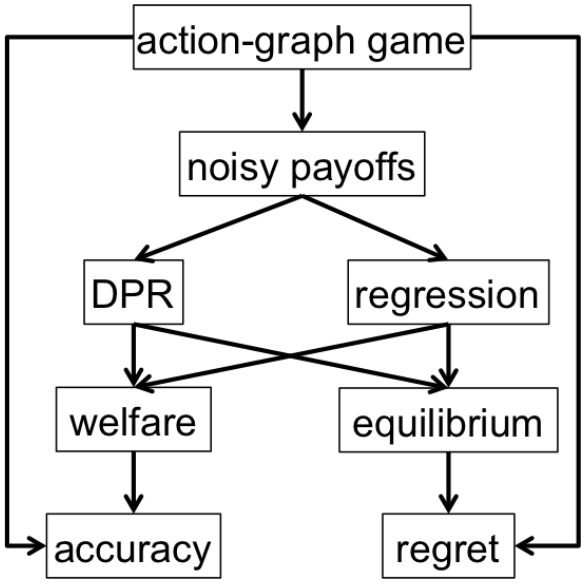


Figure 5.3: Experiment design. Action-graph representations of random local-effect games are generated and payoffs are sampled with Gaussian noise noise. Nash equilibria and social welfare are estimated using both learning and DPR, and compared against the full game.

differences, the mean payoff can be learned as a function of the profile, allowing expected values to be estimated directly for computing regret or social welfare.

5.4 Validation Experiments

To compare learned game models against deviation-preserving reduction, I developed new machinery that permits evaluation on substantially larger and more complex full games. The key to this experimental framework is the *action-graph game* representation [Jiang *et al.*, 2011]. In an action-graph game, strategies are embedded in a directed graph, and an agent's payoff depends on the strategy it chooses and the number of agents choosing the same strategy or neighboring strategies. As in Chapter 4, the full games I generate have local-effect game representations that are exponentially more compact than their symmetric payoff matrix (or even action-graph game) representations, but these local effect functions are useful only for profile-payoff queries, not for mixed-strategy analysis. However, the action-graph representation allows me to compute exact values for regret and social welfare in much larger true games than before. As opposed to the 12-player, 6-strategy games with roughly 6 thousand profiles used to validate DPR in Chapter 4, the following experiments include games up to 81 players and 8 strategies for a total of over 6 billion profiles in the full symmetric representation.

Testing on a class of games with a compact representation, like local-effect games ensures that the ground-truth payoff functions contain internal structure that could be exploited in principle by a machine learning approach. The premise that learning can induce reasonable game models from sparse data implies that games of interest will generally have some amount of structure (else inducing from sparse data is hopeless), though the form of this structure is not known a priori. The Gaussian process regression method is not in any way tuned to learning local effect games with quadratic payoff functions, but these games do satisfy the assumption that payoffs vary smoothly in profile space.

5.4.1 Experimental Setup

The general experimental setup is illustrated in Figure 5.3. For the experiments that follow, I generated 100 random local-effect game instances of each of the following sizes: 8 strategies and 17, 33, 49, 65, and 81 players, as well as 6 strategies and 61 players. In all of these games, strategies have two or three action-graph neighbors with equal probability, and the neighbors are selected uniformly at random. This means that each strategy's payoff can depend on the number of players choosing three or four different strategies (including the strategy itself). Each strategy's payoff is the sum of self-interaction and neighbor interaction terms. Interaction terms are quadratic functions of the number of players choosing a strategy with random coefficients. For self-interaction functions, I drew constant, linear, and quadratic coefficients from $\mathcal{N}(0, 4)$, $\mathcal{N}(-4, 2)$, and $\mathcal{N}(-2, 1)$ respectively. These distributions result in roughly a 2.3% chance that any given linear or quadratic self-interaction term will be positive, making symmetric pure-strategy equilibria possible but rare. For neighbor-interaction functions, constant terms were set to 0, while linear and quadratic coefficients were drawn from $\mathcal{N}(0, 2)$ and $\mathcal{N}(0, 1)$ respectively. When drawing samples from these full games, I add noise drawn from $\mathcal{N}(0, 100)$.

Using these games, I performed two experimental settings. Setting 1 is intended to demonstrate the validity of my learning approach as a replacement for the current state of practice employing deviation-preserving reduction. Setting 2 explores various parameterizations of the learning methods and various sampling regimes as input to DPR.

In Setting 1, I construct DPR games by drawing 10 samples of every required profile. Across the 8-strategy games, I constructed only 5-player DPR games, while for the 6-strategy games, I constructed 2, 3, 4, 5, 6, and 7-player DPR games. For each of these DPR games, I drew an equal-size observation set, but spread those observations over a larger set of profiles. These observation sets for learning were chosen by selecting random profiles near those used by DPR, treating the fraction of agents playing each strategy in a DPR

profile as a probability distribution and drawing 10 samples. I also tested further spreading of samples by selecting profiles near those required by a DPR game with extra players, but drawing fewer samples per profile to hold the total number of samples constant. For instance, adding one extra player when comparing against the 2-player, 6-strategy DPR game, which has 36 profiles and therefore 360 total observations, would mean selecting 2–3 profiles to observe near each of the 126 profiles required to construct the 3-player, 6-strategy DPR game. I do not claim that these sampling methods are optimal, but rather that they establish a baseline for the performance of my regression models.

On each observation set in Setting 1, I built learning models using payoff data directly ($\hat{u}$), and using differences from strategy-average payoffs ($\hat{u}^{\Delta}_{strat}$). The former were used for computing social welfare of role-symmetric mixed strategies with both sampling and point estimation, while the latter were used for computing role-symmetric mixed strategy ϵ-Nash equilibria with point estimation. In all experiments from the first set, learning was performed on mean payoffs whenever multiple samples were taken of the same profile, rather than incorporating information about sample variance into the learned models.

In Setting 2, I used 6-strategy full games with 17, 33, 49, and 65 players, and constructed 3-player DPR games with 10, 20, 40, or 80 samples per profile, as well as 5-player DPR games with 10, 20, or 40 samples per profile. For each of these DPR games, I constructed regression game models from the same data set, learning payoffs directly and player-average differences. All learning for the Setting 2 experiments made use of sample variance information, adding normalized variance to the diagonal of the Gaussian process's covariance matrix.

In Setting 2, whenever payoff differences were learned, the mean payoff was also learned, so that accurate expected value estimates could be recovered. In this set, I computed social welfare using all three methods for expected value estimation, and regret of estimated Nash equilibria using point estimation and by constructing DPR games. When constructing DPR games from learned utility functions, I varied the size of the DPR game to be both smaller and larger than the game from which samples were drawn. However, constructing a DPR game performed very poorly in initial tests, and was therefore excluded from several of the experiments that follow. The preliminary experiments indicated that increasing the size of the constructed DPR game improved expected value estimation *independent of* the data set on which learning was performed. This may merit further investigation in the future, but is unlikely to yield practical gains.

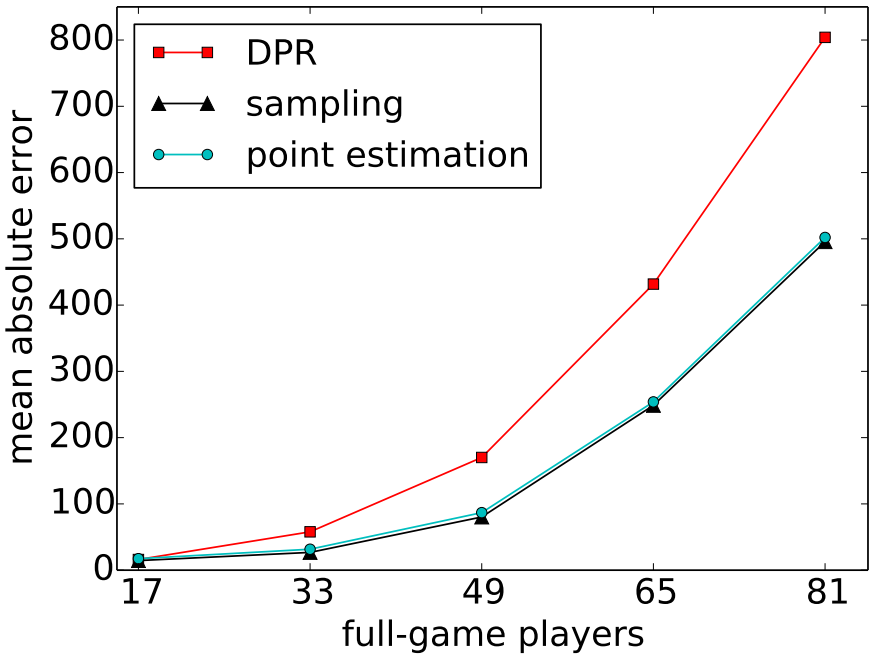


Figure 5.4: **Error of expected-value estimates, holding reduced-game size fixed.** DPR and learning perform similarly on 17-player full games, but the learning methods do much better on larger games. Estimating expected values by sampling is only slightly better than using point estimation.

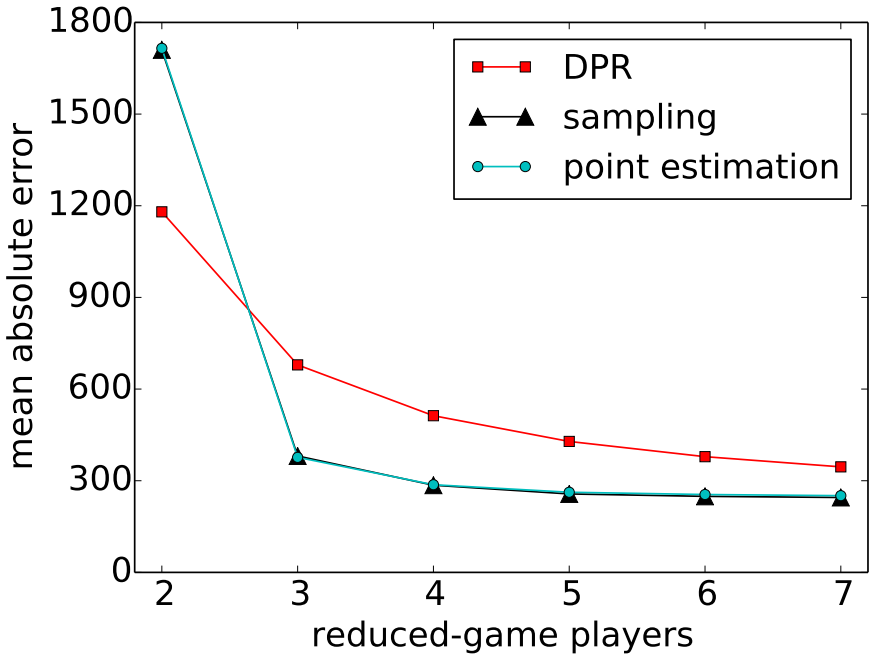


Figure 5.5: **Error of expected-value estimates, holding full-game size fixed.** For most reduced-game sizes, learning outperforms DPR, but the profiles used as input in the 2-player case are concentrated around symmetric pure strategies, leaving most of the profile space unlearned.

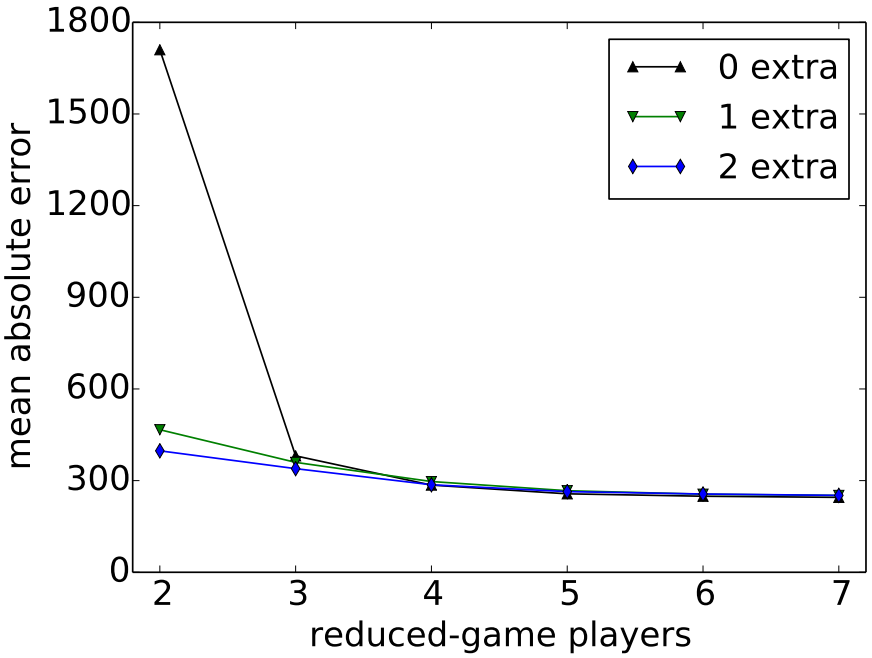


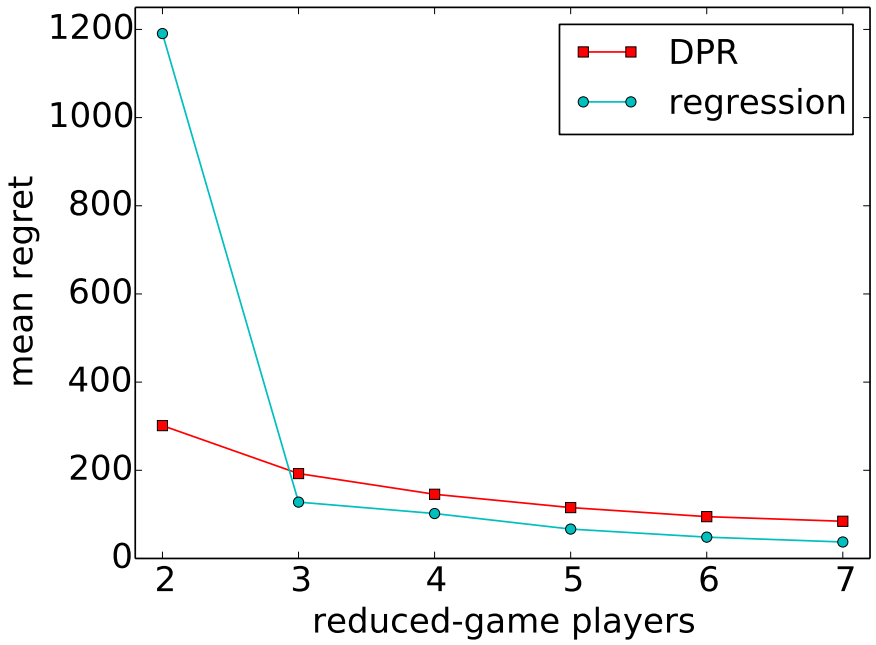
Figure 5.6: **Error of expected-value estimates, holding full-game size fixed.** Allocating the same number of samples to the learning method, but spreading them over more profiles by sampling as though the DPR game had extra players, overcomes the problems with the smallest observation set, but is irrelevant for larger reduction sizes.

5.4.2 Experimental Results

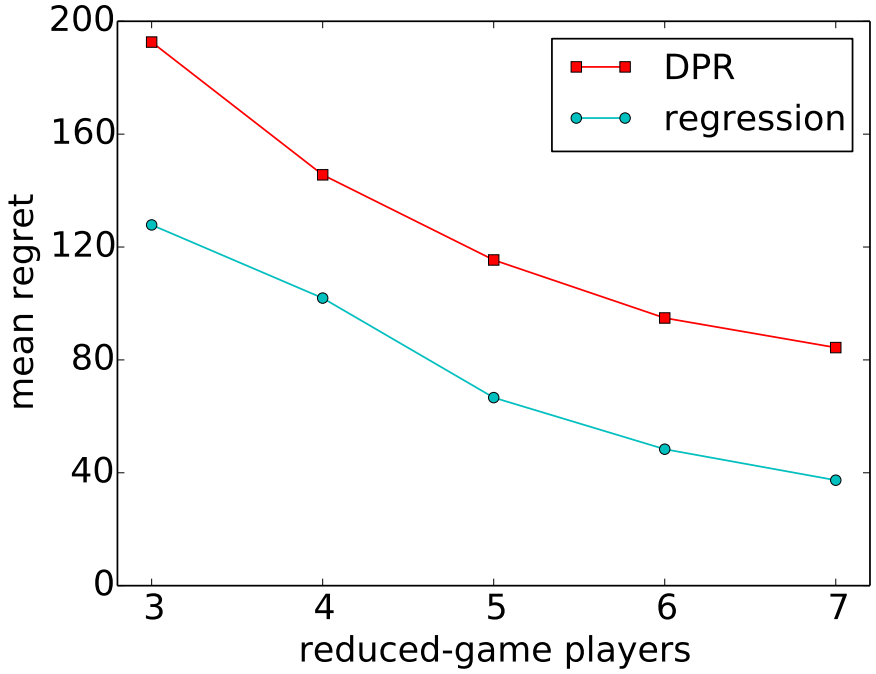
The first experiments on Setting 1 compared performance of game learning using both sampling and point estimation against DPR, at estimating $u(\vec{s}, r, s)$. Accuracy in estimating these expected values translates directly to accuracy in estimating mixed-strategy social welfare, which is a probability-weighted sum of these values (Equations 1.9 and 1.10).

I evaluated each method on 321 mixtures spaced on a grid; each point plotted in Figures 5.4 and 5.5 shows a method's average absolute error $u(\vec{s}, r, s) - \hat{u}(\vec{s}, r, s)$ across the 321 mixtures, 8 strategies, and 100 games. I varied both the size of the full game being approximated and the amount of data used to approximate it. Figure 5.4 shows estimation error plotted against the number of players in a full game with 8 strategies, holding the size of the observation set fixed at a 5-player DPR game. Figure 5.5 plots estimation error against the size of the DPR game, holding the full game fixed at 61 players and 6 strategies.

Figure 5.4 shows that for 17-player full games, all three methods have similar performance (DPR in fact falls between sampling and point estimation), but for larger games, the regression-based methods significantly outperform DPR. While it appears from the figure that approximation error is exploding with the number of players for all three methods, two mitigating factors should be noted. First, the number of profiles in a symmetric



(a)



(b)

Figure 5.7: **Regret of identified equilibria.** The payoff difference regression method performs extremely poorly at identifying equilibria with a very small amount of data, but starting from 3-player reduced game data sets, its equilibria have significantly lower regret than those from DPR.

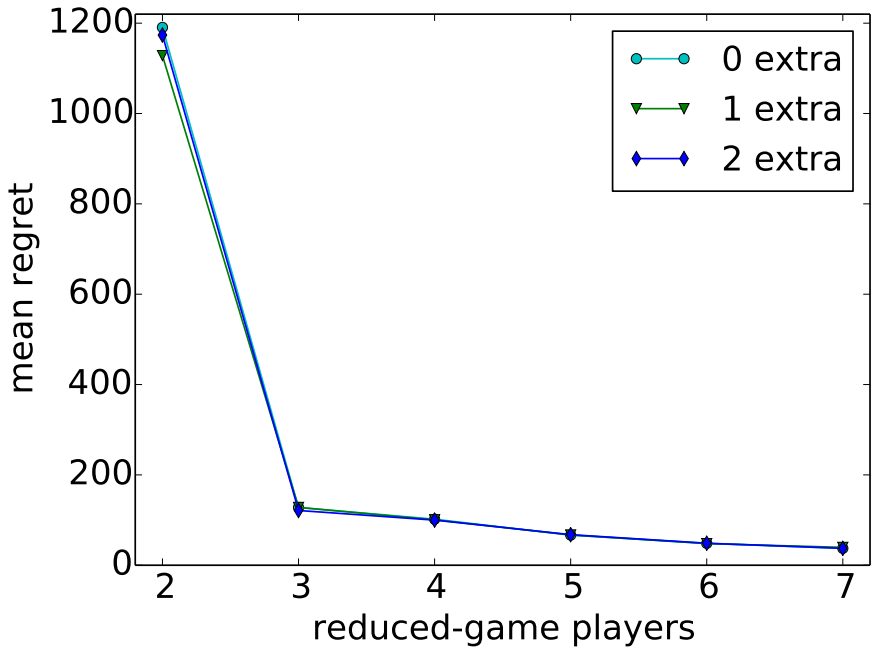


Figure 5.8: **Regret of identified equilibria.** Spreading the observation set across more profiles does not improve the quality of equilibria.

game with 8 strategies is $O(N^7)$, so the linear spacing of the horizontal axis undersells the growth in game size. Second, the average payoff magnitude in our action-graph games grows quadratically in the number of players, and I have not normalized estimation error to the payoff scale.

Figure 5.5 shows that both reduction and regression perform extremely poorly on the smallest data set—that corresponding to the 2-player DPR game, which has a total of 360 profile observations—but regression does much worse. Across all other data set sizes, regression easily outperforms DPR. Once again, sampling holds a consistently small but significant advantage over point estimation. The regression methods' poor performance on the 2-player reduction data set is attributable to the zero-mean Gaussian process prior. Due to the manner in which DPR games are constructed, all observations in this setting are allocated to profiles where nearly all players choose the same strategy. This means that the regression has almost no data for profiles with a mix of several different strategies, and is dominated by its prior over most of the profile space. DPR, by contrast, estimates expectations in a 2-player game and therefore interpolates linearly between the extreme profiles for which it has data.

Figure 5.6 plots estimation error against reduced-game size for the default data spread, as well as one and two extra players, as explained above. In this graph, the curve labeled "0 extra" corresponds to the curve labeled "sampling" in Figure 5.5; the other two curves also

use the sampling method, but results for point estimation would look identical. Spreading the 360 samples in the smallest input set over a wider collection of profiles vastly improves results for the 2-player DPR input size. This confirms the explanation of the anomaly at 2 players in Figure 5.5. For reduced games with more than two players, the extra spread in the profiles has a negligible effect on Gaussian process model accuracy.

The second set of experiments on Setting 1 compared the quality of equilibria computed using the payoff-difference learning method and DPR. For each game in the 61-player 6-strategy data set, I computed equilibria using each method at 2–7 reduced-game players, and for each equilibrium, I computed regret in the full game. Figure 5.7 summarizes the results; 5.7a shows that the Gaussian process method again performs extremely poorly on the smallest data set, but 5.7b zooms in on the 3–7-player reduced games where learning demonstrates a consistent advantage. Learning's absolute regret advantage over DPR stays roughly constant over the range shown in Figure 5.7b at approximately 50 in the arbitrary payoff units of our action-graph games, but the relative advantage grows from a factor of 1.5 to 2.25.

As with the expected value experiments above, I tested the equilibrium computation method using input data with observations spread over more profiles by adding extra players to the DPR profile set while holding the number of observations constant. Figure 5.8 shows the results of these experiments. Unlike expected values, even with the smallest data set (corresponding to DPR_2), equilibrium-finding did not benefit from more-widely spread observations. While I cannot rule out the possibility that some better allocation of a small sample set exists, I are forced to conclude that computing equilibria in 82 million-profile games with payoff-difference learning will generally require more than 360 observations.

Results from Setting 2 are shown in Figures 5.9 and 5.10. In both figures, the points at 792, 1584, and 3168 profiles sampled correspond to 5-player DPR games, while the remaining four sample counts are from 3-player DPR games. For both regression and reduction, increasing the number of samples generally helped, but this effect was dwarfed by the effect of spreading out the samples by increasing the number of players in the reduced game. Regressions on player-weighted differences performed worse than learning payoffs directly in both experiments across virtually all full-game and data set sizes. Results for player-weighted differences are therefore not shown, and the points labeled *GP* correspond to directly-learned payoffs.

The results for expected value estimation in Figure 5.9 indicate that with the help of variance information, learned models can approximate true-game expected values better than DPR even if both receive the same DPR data set. The results for regret of equilibria computed in learned and DPR games shown in Figure 5.10 are less encouraging for the

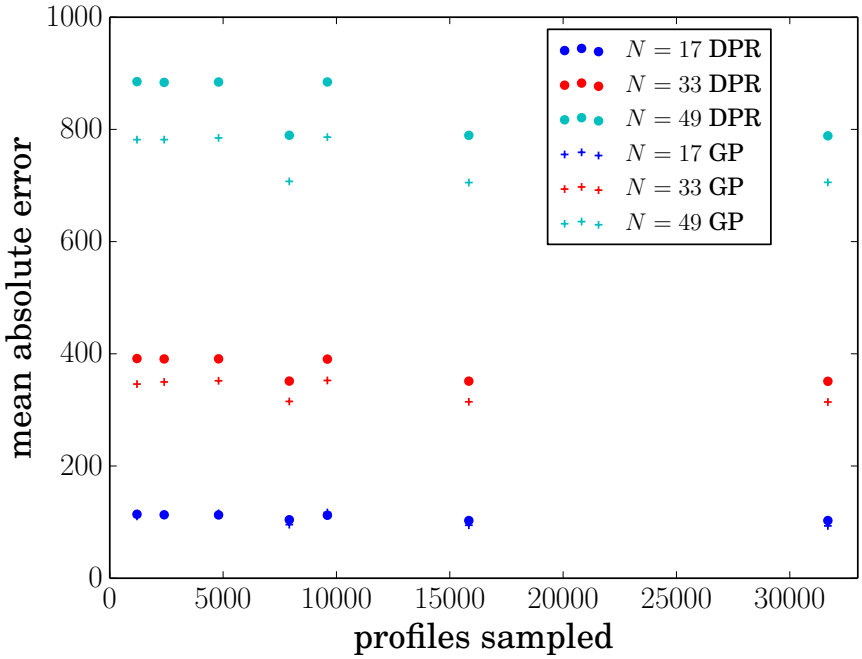


Figure 5.9: **Setting 2: Error of expected-value estimates.** Regression performs well here despite using the same data set as DPR. Neither method benefits substantially for more samples of the same profiles.

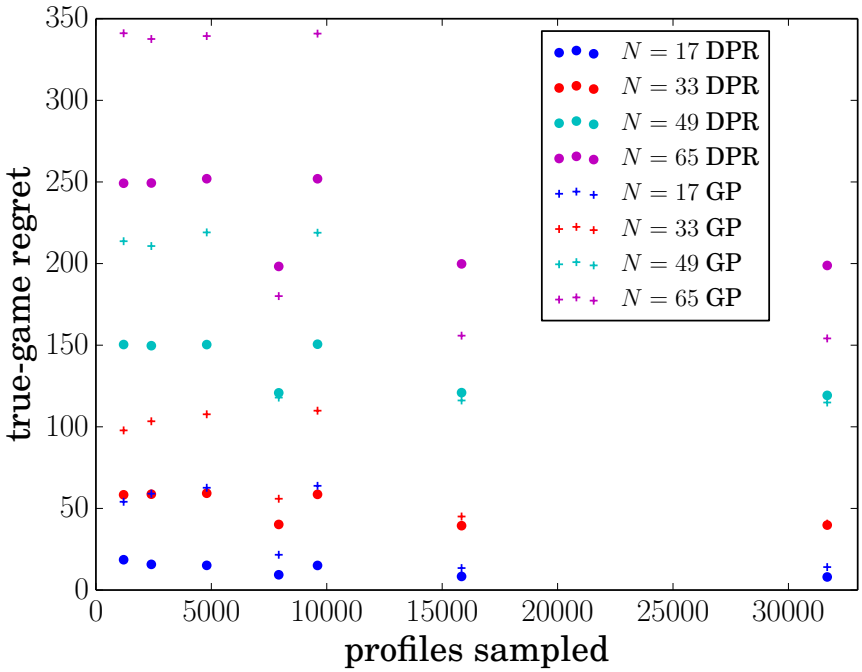


Figure 5.10: **Setting 2: Regret of identified equilibria.** Regression never outperforms DPR with $n = 3$, but in $n = 5$ reduced games, regression does as well as or better than DPR.

learning method. In small games and with small amounts of data DPR gives consistently lower-regret equilibria, though in the largest data sets, regression gives reasonable results.

Several additional sets of experiments are clearly necessary in Setting 2. First, spreading samples over more profiles needs to be tested in combination with inputting variances to the Gaussian process regression. This alone may make a pivotal difference in favor of the learning model. Second, huge disparity between Setting 1 and Setting 2 results on learning differences from profile-average payoffs needs to be explained.

5.5 Conclusions

I have demonstrated a method for approximating simulation-based games with a large number of players from a small amount of data using machine learning. The method uses Gaussian process regression on a count-vector representation of profiles can learn utility functions for each strategy. From the learned Gaussian processes, we can estimate expected values of symmetric mixed strategies using either sampling or point estimation. Sampling gives the most accurate estimates and is therefore preferable for computing social welfare, while point estimation is nearly as accurate and numerically stable, making it best for input to replicator dynamics when computing Nash equilibria. Learning payoff differences may improve Nash equilibrium computation.

The action-graph game experimental framework allows testing on games with underlying payoff structure that should be learnable, but is unknown to the learning methods. The test suite includes games many orders of magnitude larger than any used in previous game-approximation experiments. Experimental results show that learning can outperform DPR as it is commonly applied in approximating large simulation-based games, producing lower error in social welfare estimates and lower regret of identified equilibria when given sufficient data.

An advantage of learning over reduction is that it can take an arbitrary set of profiles as input and could therefore be applied to empirical games based on data that derives from an observational process rather than a directed simulation. In simulation-based games, this opens an avenue for future research exploring better allocations of sampling effort for input to payoff regression models. In addition, the flexibility of learned models to accept additional data could facilitate interleaving of sampling and analysis, with preliminary estimates of equilibria guiding additional sampling.

The Gaussian process models that my method learns hold more information than I am so far taking advantage of: I query the mean of the distribution at points of interest, but never query variance. Variance information could help to guide sampling decisions, prioritizing

exploration in regions of the profile space where payoff estimates are uncertain. It would also be useful to support additional types of game-theoretic reasoning—like identifying dominated strategies—with the learned payoff models, and variance information might contribute to confidence bounds on these analysis results.

CHAPTER 6

Conclusion

Game-theoretic reasoning is a powerful tool for understanding multi-agent systems that relies on precise description of agent interactions and incentives. Simulation-based game theory enables analysis based on procedural descriptions of agents and their environment, but faces computational bounds when applied to large-scale interactions. This thesis provides methods that help address the scaling of simulation-based game analysis with respect to the number of agents and the amount of data.

6.1 Summary of Contributions

The study of strategic formation of credit networks described in Chapter 2 demonstrates the value of simulation-based game theory as a modeling tool where classical game-theoretic analysis comes up short. Because interactions among agents in a credit network depend on the state of the network, which depends on the history of transactions, a clean analytical characterization of agent incentives is out of the question. Further, because interactions extend over a long period of time with branching possibilities for the network's state, enumerating all possible outcomes is a fool's errand.

One approach to understanding strategic behavior in such complex scenarios is to assume away some of the complexity to produce an analytically tractable model, then solve that model and follow up by exploring how insights from the simple model may generalize. My co-authors pursue this simplification approach, but I show that their results generalize poorly to even slightly more complex settings. Using simulation-based game analysis, I analyze a much richer credit network formation scenario. This analysis necessarily provides weaker results than the analytical approach, but gives insights about a much more realistic setting. I explore heuristics for issuing credit, and the sorts of credit networks that can result, showing among other results that the information structure plays a critical role in agents' ability to form centralized networks.

The credit network domain demonstrates the need for new techniques to scale simulation-based game theory. With 61 agents and 32 strategies, the full game would have 3.0×10^{24} profiles, so simulating all of them is clearly infeasible. Iterative exploration through the inner and outer loop procedures helps to reduce the number of strategies that must be considered simultaneously, but is not sufficient to ensure tractable analysis. Player reduction was needed to allow exploration of a non-trivial number of strategies; the very first version of the credit network study [Dandekar *et al.*, 2011b] employed hierarchical reduction. The recognition of potentially spurious HR equilibria in this setting motivated the development of deviation-preserving reduction, which has been used in subsequent credit network SGT studies.

The credit network simulator produces payoff observations with relatively high variance. To combat this, at least 1000 and sometimes many more samples were taken of every profile simulated in the iterative exploration of the DPR game. The number of simulations was generally chosen using a rule-of-thumb that could easily have led to far too much or too little sampling effort. Too little sampling would jeopardize the validity of game analysis results, while too much sampling could have impeded exploration of a larger set of strategies or additional environment parameters. Bootstrap-based statistical methods for simulation-based game theory can help ensure that samples are allocated efficiently and that results from the SGT study can be taken as serious scientific evidence about the simulated environment.

In Chapter 3, I show how ideas from bootstrap statistics can be adapted to analysis of simulation-based games. By simultaneously resampling all payoffs in a simulation-based game, confidence intervals can be computed for SGT statistics. Most importantly, this technique allows analysts to report statistically valid regret estimates for equilibrium candidates in simulation-based games. Confidence intervals for regret also allow for better sampling control in SGT studies. I demonstrated empirically that bootstrap regret distributions and the resulting confidence intervals are well-calibrated across a number of game classes and noise distributions.

In Chapter 4, I describe deviation-preserving reduction, a player reduction method for approximating games with a large number of agents by constructing smaller games from simulation data. Deviation-preserving reduction improves over other player reduction methods by ensuring that the information most crucial to computation of Nash equilibrium—the payoff differences resulting from unilateral deviation—is preserved in the reduced game. DPR also allows the analyst to choose the reduced game size, trading off time to construct the model with accuracy of approximation. In my experiments, I applied DPR and other player reductions to various classes and sizes of games, and demonstrated that

DPR provides better analysis with respect to the full game across a number of measures, most importantly the full-game regret of reduced-game equilibria.

In Chapter 5, I re-frame the player reduction problem as a machine learning problem, and propose methods based on Gaussian process regression to construct game models from sparse simulation data. This approach makes use of data ignored by DPR and can allow for better allocation of sampling effort. I test several approaches for learning payoff data and for extracting expected values of mixed strategies from the learned models. I compare DPR and learning approaches on much larger full games through an action-graph game test suite. My experiments show that learned payoff models can yield better payoff estimates and equilibria than DPR.

6.2 Future Directions

Deviation-preserving reduction has been employed in a number of recent simulation-based game studies. This success is attributable in part to the strong similarity between DPR and HR—which it has largely supplanted—from a methodological perspective. A researcher interested in simulation-based analysis of a large game likely would already have needed to employ player reduction, and substituting DPR for another reduction method is conceptually straightforward. Another factor that certainly helped promote adoption of DPR is ease of use: through the GameAnalysis package I have provided support for analyzing deviation-preserving reduction games, and with my guidance, DPR schedulers were incorporated into the EGTA-Online system for running game-theoretic simulations [Cassell and Wellman, 2013]. Finally, wider adoption was likely aided by my own use of DPR in studying credit network formation, which both pushed me and my colleagues to improve support for DPR and added further demonstration of its viability to the validation experiments described in Chapter 4.

Evidence in this thesis indicates that my bootstrapping and model learning methods represent substantial improvements over the current state of practice in simulation-based game theory. I therefore want to learn from the example of deviation-preserving reduction and take active steps to promote them, rather than allow them to fall out of use like a number of the related works cited throughout this thesis. To that end, I am actively engaged in simplifying the interface to both methods and better integrating them with the rest of the GameAnalysis suite. An obvious point for integrating bootstrapping with existing tools is in the automated scheduling of simulations for the SGT inner loop. At present sampling decisions are often driven by equilibria and regrets computed from a small number of samples. Using bootstrapping to control these sampling decisions would immediately

improve the reliability of conclusions from the inner loop search process, and would also promote the use of bootstrap statistics for the eventual reporting of results from studies using automated strategy exploration.

The formulation of replicator dynamics that I give in Equations 1.11–1.13 bears a strong resemblance to Markov chain Monte Carlo problems. This resemblance is strengthened when considering sampling-based expected value estimates as input to replicator dynamics in Section 5.3. Reformulating this problem to apply MCMC algorithms like Metropolis-Hastings might yield significant benefit when computing equilibria from incomplete sampling of the payoffs. Importance sampling approaches could similarly help to focus on relevant regions of the strategy space and reduce the sampling burden in large games.

A number of practical questions about game model learning will probably be driven forward best by practical application. For example, at present there is no clear prescription for how to select profiles to simulate as inputs to the regression; in Chapter 5, I proposed further methodological studies to resolve this question, but I expect that practical application of game model learning is at least as likely to suggest good ideas. I therefore intend to make sure that my next simulation-based game study can make use of learning to push the method forward and provide practical demonstration of its applicability.

In Chapters 3, 4, and 5, I describe potential extensions to the proposed methods and avenues for additional validation, but there is also room for further work exploring combined application of these techniques. In particular, bootstrapping and game model learning could potentially be mutually reinforcing. At present, my methods do not take full advantage of the learned Gaussian process models, ignoring the produced variance estimates. These estimates provide additional information about model uncertainty that could contribute to bootstrap-based confidence measures. The learning methods could also potentially benefit from bootstrap statistics for model selection. Further, both tools open new questions about how samples should be gathered for simulation-based game studies that should ideally be explored in tandem. It is also worth considering how my discrete profile-based learning methods could be combined with the prior work of Vorobeychik *et al.* [2007] learning payoff functions over continuous strategy spaces, and with outer-loop strategy exploration methods [Schvartzman and Wellman, 2009; Wellman *et al.*, 2013] to improve scaling in the size of the strategy set $|S_r|$.

My thesis provides several techniques that help improve analysts to derive generalizable results from simulation-based game analysis. One sort of generalization that has to date received little attention is how equilibrium analysis varies with the number of agents. Many simulation-based game studies compare across multiple environments with different simulation parameters, sometimes including different numbers of agents. However, the

number of simulated agents is often chosen fairly arbitrarily (this was certainly the case in the credit network formation game), and can in principle have a large influence on the results of SGT models. More principled techniques for generalizing game analysis across simulations with a varying number of agents would greatly improve the explanatory power of simulation-based game theory.

Another avenue for generalization through simulation-based game theory may lie in incorporation of non-standard preference models. Much work in behavioral game theory has explored ways that human decision-making differs from game-theoretic predictions, but resulting models are often analytically intractable. SGT may present an easier route to making use of behavioral models, and SGT could reap further benefits in generalizability from methods to incorporate behavioral models of agent decisions. Simulation-based game theory provides a computational approach that expands the reach of game-theoretic analysis. This thesis pushes the boundaries of what problems can be studied with simulation-based game theory through better approximations and use of data, but much room for new SGT techniques remains.

BIBLIOGRAPHY

Elliot Anshelevich and Martin Hoefer. Contribution games in networks. *Algorithmica*, 63:51–90, 2012.

Robert Axtell, Robert Axelrod, Joshua M. Epstein, and Michael D. Cohen. Aligning simulation models: A case study and results. *Computational and Mathematical Organization Theory*, 1(2):123–141, 1996.

Tim Baarslag, Katsuhide Fujita, Enrico H. Gerding, Koen Hindriks, Takayuki Ito, Nicholas R. Jennings, Catholijn Jonker, Sarit Kraus, Raz Lin, Valentin Robu, and Colin R. Williams. Evaluating practical negotiating agents: Results and analysis of the 2011 international competition. *Artificial Intelligence*, 198:73–103, 2013.

Venkatesh Bala and Sanjeev Goyal. A noncooperative model of network formation. *Econometrica*, 68:1181–1229, 2000.

Ben-Alexander Cassell and Michael P. Wellman. Asset pricing under ambiguous information: An empirical game-theoretic analysis. *Computational and Mathematical Organization Theory*, 18(4):445–462, 2012.

Ben-Alexander Cassell and Michael P. Wellman. EGTAOnline: An experiment manager for simulation-based game studies. In *Multi-Agent Based Simulation XIII*, volume 7838 of *Lecture Notes in Artificial Intelligence*. Springer, 2013.

Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. MIT Press, 1995.

Jacomo Corbo, Antoni Calvó-Armengol, and David Parkes. A study of Nash equilibrium in contribution games for peer-to-peer networks. *SIGOPS Operating Systems Review*, 40:61–66, 2006.

Pranav Dandekar, Ashish Goel, Ramesh Govindan, and Ian Post. Liquidity in credit networks: A little trust goes a long way. In *12th ACM Conference on Electronic Commerce*, pages 147–156, San Jose, 2011.

Pranav Dandekar, Bryce Wiedenbeck Ashish Goel, and Michael P. Wellman. Strategic formation of credit networks: Preliminary report. In *Trading Agent Design and Analysis Workshop*, 2011.

Pranav Dandekar, Ashish Goel, Michael P. Wellman, and Bryce Wiedenbeck. Strategic formation of credit networks. *ACM Transactions on Internet Technology*, 15(1):3:1–3:41, 2015.

Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *Journal on Computing*, 39(1):195–259, 2009.

A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*. Cambridge University Press, 1997.

Dimitri B. DeFigueiredo and Earl T. Barr. TrustDavis: A non-exploitable online reputation system. In *7th IEEE International Conference on E-Commerce Technology*, pages 274–283, Washington, DC, 2005.

Quang Duong, Yevgeniy Vorobeychik, Satinder Singh, and Michael P. Wellman. Learning graphical game models. In *21st International Joint Conference on Artificial Intelligence*, pages 116–121, 2009.

Alex Fabrikant, Ankur Luthra, Elitza N. Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *22nd ACM Symposium on Principles of Distributed Computing*, pages 347–351, 2003.

Sevan G. Ficici, David C. Parkes, and Avi Pfeffer. Learning and solving many-player games through a cluster-based representation. In *24th Conference on Uncertainty in Artificial Intelligence*, pages 187–195, Helsinki, 2008.

Linda W. Friedman and Hershey H. Friedman. Analyzing simulation output using the bootstrap method. *Simulation*, 64(2):95–100, 1995.

Xi Alice Gao and Avi Pfeffer. Learning game representations from data using rationality constraints. In *26th Conference on Uncertainty in Artificial Intelligence*, pages 185–192, 2010.

Arpita Ghosh, Mohammad Mahdian, Daniel M. Reeves, David M. Pennock, and Ryan Fugger. Mechanism design on trust networks. In *3rd International Workshop on Internet and Network Economics*, pages 257–268, 2007.

Jean Honorio and Luis Ortiz. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research*, to appear, 2015.

Matthew O. Jackson and Asher Wolinsky. A strategic model of social and economic networks. *Journal of Economic Theory*, 71(1):44–74, 1996.

Albert Xin Jiang, Kevin Leyton-Brown, and Navin A. R. Bhat. Action-graph games. *Games and Economic Behavior*, 71(1):141–173, 2011.

Patrick R. Jordan and Michael P. Wellman. Generalization risk minimization in empirical game models. In *8th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 553–560, Budapest, Hungary, 2009.

Patrick R. Jordan, Christopher Kiekintveld, and Michael P. Wellman. Empirical game-theoretic analysis of the TAC supply chain game. In *6th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1188–1195, Honolulu, HI, 2007.

Patrick R. Jordan, Yevgeniy Vorobeychik, and Michael P. Wellman. Searching for approximate equilibria in empirical games. In *7th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1063–1070, Estoril, Portugal, 2008.

Dean Karlan, Markus Mobius, Tanya Rosenblat, and Adam Szeidl. Trust and social collateral. *Quarterly Journal of Economics*, 124(3):1307–1361, 2009.

Michael Kearns and Yishay Mansour. Efficient Nash computation in large population games with bounded influence. In *18th Conference on Uncertainty in Artificial Intelligence*, pages 259–266, Edmonton, 2002.

Kevin Leyton-Brown and Moshe Tennenholtz. Local-effect games. In *18th International Joint Conference on Artificial Intelligence*, pages 772–780, Acapulco, 2003.

Freddy Limpens and Denis Gillet. A competence bartering platform for learners. In *10th International Conference on Advances in Web-Based Learning*, pages 148–153, Hong Kong, China, 2011.

Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Playing large games using simple strategies. In *4th ACM Conference on Electronic Commerce*, pages 36–41, San Diego, CA, 2003.

Zhengye Liu, Hao Hu, Yong Liu, Keith W. Ross, Yao Wang, and Markus Mobius. P2P trading in social networks: The value of staying connected. In *29th IEEE International Conference on Computer Communications*, pages 1–9, San Diego, CA, 2010.

Alan Mislove, Ansley Post, Peter Druschel, and Krishna P. Gummadi. Ostra: Leveraging trust to thwart unwanted communication. In *5th Usenix Symposium on Networked Systems Design and Implementation*, pages 15–30, San Francisco, 2008.

Dov Monderer and Lloyd S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ryan Porter, Eugene Nudelman, and Yoav Shoham. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior*, 63(2):642–662, 2008.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

Daniel M. Reeves. *Generating trading agent strategies: Analytic and empirical methods for infinite and large games*. PhD thesis, University of Michigan, 2005.

Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

L. Julian Schvartzman and Michael P. Wellman. Stronger CDA strategies through empirical game-theoretic analysis and reinforcement learning. In *8th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 249–256, Budapest, Hungary, 2009.

Peter D. Taylor and Leo B. Jonker. Evolutionary stable strategies and game dynamics. *Mathematical biosciences*, 40(1):145–156, 1978.

Joel Veness, Marc Lanctot, and Michael Bowling. Variance reduction in Monte-Carlo tree search. In *Advances in Neural Information Processing Systems*, pages 1836–1844, 2011.

Bimal Viswanath, Mainack Mondal, Krishna P. Gummadi, Alan Mislove, and Ansley Post. Canal: Scaling social network-based Sybil tolerance schemes. In *7th European Conference on Computer Systems*, Bern, Switzerland, 2012.

Yevgeniy Vorobeychik, Michael P. Wellman, and Satinder Singh. Learning payoff functions in infinite games. *Machine Learning*, 67(1-2):145–168, 2007.

Yevgeniy Vorobeychik. Probabilistic analysis of simulation-based games. *ACM Transactions on Modeling and Computer Simulation*, 20(3), 2010.

Elaine Wah and Michael P. Wellman. Welfare effects of market making in continuous double auctions. In *14th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 57–66, Istanbul, 2015.

Michael P. Wellman, Daniel M. Reeves, Kevin M. Lochner, Shih-Fen Cheng, and Rahul Suri. Approximate strategic reasoning through hierarchical reduction of large symmetric games. In *20th National Conference on Artificial Intelligence*, pages 502–508, Pittsburgh, 2005.

Michael P. Wellman, Anna Osepayshvili Jeffrey K. MacKie-Mason, and Daniel M. Reeves. Bidding strategies for simultaneous ascending auctions. *B. E. Journal of Theoretical Economics (Topics)*, 8(1), 2008.

Michael P. Wellman, Tae Hyung Kim, and Quang Duong. Analyzing incentives for protocol compliance in complex domains: A case study of introduction-based routing. In *Twelfth Workshop on the Economics of Information Security*, Washington, 2013.

Michael P. Wellman. Methods for empirical game-theoretic analysis (extended abstract). In *21st National Conference on Artificial Intelligence*, pages 1552–1555, Boston, 2006.

Bryce Wiedenbeck and Michael P. Wellman. Scaling simulation-based game analysis through deviation-preserving reduction. In *11th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 931–938, Valencia, 2012.

Bryce Wiedenbeck and Michael Wellman. Learning payoffs in large symmetric games (extended abstract). In *14th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1745–1746, 2015.

Bryce Wiedenbeck, Ben-Alexander Cassell, and Michael P. Wellman. Bootstrap statistics for empirical games. In *13th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 597–604, Paris, 2014.