A Regression Approach for Modeling Games with Many Symmetric Players

Bryce Wiedenbeck
Swarthmore College
bwieden1@swarthmore.edu

Fengjun Yang
Swarthmore College
fyang1@swarthmore.edu

Michael P. Wellman
University of Michigan
wellman@umich.edu

Abstract

We exploit player symmetry to formulate the representation of large normal-form games as a regression task. This formulation allows arbitrary regression methods to be employed in in estimating utility functions from a small subset of the game's outcomes. We demonstrate the applicability both neural networks and Gaussian process regression, but focus on the latter. Once utility functions are learned, computing Nash equilibria requires estimating expected payoffs of pure-strategy deviations from mixed-strategy profiles. Computing these expected values exactly requires an infeasible sum over the full payoff matrix, so we propose and test several approximation methods. Three of these are simple and generic, applicable to any regression method and games with any number of player roles. However, the best performance is achieved by a continuous integral that approximates the summation, which we formulate for the specific case of fully-symmetric games learned by Gaussian process regression with a radial basis function kernel. We demonstrate experimentally that the combination of learned utility functions and expected payoff estimation allows us to efficiently identify approximate equilibria of large games using sparse payoff data.

The study of intelligent agents naturally gives rise to questions of how multiple agents will interact. Game-theoretic models offer powerful methods for reasoning about such interactions and have therefore become a key component of the AI toolkit. Like many core AI methods, game-theoretic analysis has benefited from recent advances in machine learning algorithms and from the availability of ever-larger data sets. Most notably, automated playing and solving of extremely large extensive-form games has advanced considerably with the aid of learning algorithms that generalize across game states. In this paper, we consider extremely large *normal-form* games, and show that learning algorithms generalizing across symmetric players can help to compactly represent and efficiently solve such games.

In both normal and extensive form games, the chief obstacle to computing equilibria is the enormous size of the standard input representation (Papadimitriou and Roughgarden 2005; Bowling et al. 2015). In games like poker and go, the extensive form representation is far too big to be constructed explicitly, but observations of particular states of the game can be straightforwardly generated by simulation. Using simulated data from a small subset of the game's states, learning techniques have succeeded in extracting general principles that apply to never-before-seen parts of the game. Agents employing such learning have achieved high-level play (Silver et al. 2016; Moravčík et al. 2017) and identified approximate equilibria (Heinrich, Lanctot, and Silver 2015).

In normal-form games, the representational complexity derives principally from the number of players. A standard payoff matrix representation of a game with $|P|$ players and $|S|$ strategies per player records a vector of payoff values in each of $|S|^{|P|}$ cells. In many AI applications, symmetries among agents permit some economy of representation, but even a fully symmetric game must record payoffs for $\binom{|P|+|S|-1}{|P|}$ distinct outcomes (Cheng et al. 2004). As a result, analysts are often restricted to studying games with small numbers of players and strategies, or with special-purpose compact representations.

In the present work, we use data about a small subset of a game's outcomes as input to learning methods that seek to extract general principles of the game's utility function. Data of this form is common in the field of *empirical game-theoretic analysis* (Wellman 2006). EGTA is used to study multi-agent interactions where a closed-form game model is unavailable or intractable. In such settings, an agent-based model can often capture key features of the interaction and provide data about agent incentives. Generally, a single run of an agent-based simulation generates a noisy sample of each payoff value in one cell of the payoff matrix. Settings where EGTA has been employed include identifying bidding strategies in continuous double auctions (Phelps, Marcinkiewicz, and Parsons 2006), designing network routing protocols (Wellman, Kim, and Duong 2013), understanding credit provision in the absence of a central currency (Dandekar et al. 2015), and finding ways to mitigate the effects of high-frequency trading (Wah and Wellman 2016).

Because of the combinatorial growth of the payoff matrix, filling every cell by simulation quickly becomes intractable as the number of players and strategies grows. Early EGTA studies were often restricted to small numbers of players and strategies (Phelps, Marcinkiewicz, and Parsons 2006; Walsh et al. 2002). More recently, growth in the strategy space

has been kept in check by iterative exploration approaches, and games with many players have been approximated using player reduction methods (Wellman, Kim, and Duong 2013; Wah, Hurd, and Wellman 2015). Our work presents an alternative to player reduction, where the use of machine learning allows for better generalization from more flexible data sets, resulting in better approximations of large symmetric games.

The key to learning that generalizes across outcomes is that games with large numbers of players generally exhibit substantial structure beyond player symmetries. With many players, we often expect that no single opponent can unilaterally exert an outsized influence on a player's payoff. For example, in a 100-player game, the difference in payoff between 36 or 37 opponents choosing a particular strategy is likely to be small. A related notion of bounded influence is formalized and studied by Kearns and Mansour (2002). Further, when a game represents interactions among computational agents, and the payoff matrix is too big to be represented, we should expect that the agents themselves are not reasoning about the full game, but rather some more-compact summarization. As a result, in many large games of interest, payoff functions will exhibit smoothness and simplicity that make them amenable to machine learning.

Related Work

Much previous work in machine learning for game analysis has focused on extensive-form games like poker (Heinrich, Lanctot, and Silver 2015; Moravčík et al. 2017). In these settings, the game being studied is precisely defined, but too large to be analyzed directly, so learning is used to express strategies compactly and estimate expected payoffs in various game states without performing an exhaustive search. Our work is motivated by empirical settings where the game model, in addition to being extremely large, is initially unknown and must be induced from data. As is common in such settings, we treat the set of strategies as fixed and exogenously specified. Further, our methods treat strategies as categorical, requiring no relationship among the strategies to make generalizations.

We are interested analyzing in games with a sufficiently large number of players to pose representational challenges even if the set of strategies can be fully enumerated. A common approach to analyzing normal-form games with many players is to employ game models with compact representations that can be analyzed directly. Examples include potential games (Monderer and Shapley 1996), graphical games (Kearns 2007), and action-graph games (Jiang, Leyton-Brown, and Bhat 2011). In an empirical setting, where the game model is not known in advance, such representations are difficult to apply, but some researchers have investigated learning compact representations from data. Duong et al. (2009) developed a method for detecting graphical structures representing independences among players' payoffs. 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. Neither of these approaches can be applied to symmetric games, because graphical games derive their compactness from player independence, which cannot arise in non-trivial symmetric games.

An alternative approach that has been used when EGTA environments simulate a large number of symmetric players is called *player reduction*. Player reductions define a reduced game with a small number of players, and fill in the reduced game's payoff matrix using data from the full game by aggregating the decisions of several symmetric players. Equilibria are then computed in the reduced game and treated as approximate equilibria of the full game. Several player reduction methods have been proposed, varying on the choice of full-game profiles to simulate and how they map them to payoffs in the reduced game.

The first player-reduction method to see widespread use was hierarchical reduction (Wellman et al. 2005), which treats each reduced-game player as controlling an equal fraction of the full-game agents. Hierarchical reduction has been largely supplanted by a more recent technique called *deviation-preserving reduction* (DPR) (Wiedenbeck and Wellman 2012). DPR treats each player in the reduced game as controlling the strategy choice of a single agent in the full game, but each player views its opponents as an aggregation of the remaining full-game agents. This means that payoff differences resulting from a single reduced-game player switching strategies reflect payoff changes from a single full-game agent deviating, making DPR equilibria more reflective of full-game equilibria. We employ DPR as a benchmark against which to compare our learning methods.

Relative to DPR or other player reductions, our methods take advantage of greater flexibility in allowable input data. Player reduction methods prescribe a fixed set of profiles to simulate, and to ensure accurate estimates of reduced game payoffs, users often end up simulating the same profile many times. Our learned models do not require a fixed set of full-game profiles and can therefore spread sampling effort over a wider variety of profiles. DPR also ignores some freely available data: simulating profiles with many strategies but using the payoff data for only one strategy. Our regressions can always make use of any data that is available.

Most closely related to the present work, Vorobeychik, Wellman, and Singh (2007) demonstrated the use of regression methods to learn payoff functions over continuous strategy spaces. The present paper 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. By contrast, our methods can handle arbitrary sets of strategies, relying instead on the game having a large number of symmetric players, which is typical of environments defined by agent-based simulations.

Background and Notation

Game Theory

We focus on games represented in normal form that have significant symmetry among players. In the following presentation, we focus on fully-symmetric games, where all players have the same set of strategies and face the same

set of incentives. However, most of our models generalize straightforwardly to role-symmetric games, where players are partitioned into some number of roles (such as buyers and sellers), and players are symmetric within, but not across roles. Formally, a *symmetric game* consists of:

- a set of players P
- a set of strategies S
- a utility function $u : S \times \vec{S} \to \mathbb{R}$

A *profile* $\vec{s}$ is an assignment of one strategy to every player. Because players are symmetrical, we can represent a profile by a vector of the number of players choosing each strategy. We denote the set of all profiles $\vec{S}$. The utility function $u(s, \vec{s})$ maps a profile $\vec{s}$ and a strategy s to the utility of a player choosing strategy s when players jointly choose profile $\vec{s}$. A mixed strategy σ specifies a probability distribution over a player's strategies. A symmetric mixture $\vec{\sigma}$ is a common mixed strategy played by all agents.

A player selecting pure strategy s when other players jointly play according to symmetric mixture $\vec{\sigma}$ receives an expected payoff:

$$u(s, \vec{\sigma}) = \sum_{\vec{s} \in \vec{S}} Pr[\vec{s}|\vec{\sigma}]u(s, \vec{s}) \tag{1}$$

A player from playing according to a symmetric mixture $\vec{\sigma}$ receives expected payoff:

$$u(\vec{\sigma}) = \sum_{s \in S} \vec{\sigma}(s)u(s, \vec{\sigma})$$

The regret of a symmetric mixture is the maximum amount that any player could gain by deviating to a pure strategy:

$$regret(\vec{\sigma}) = \max_{s \in S} u(s, \vec{\sigma}) - u(\vec{\sigma})$$

A symmetric Nash equilibrium is a symmetric mixture with $regret(\vec{\sigma}) = 0$. It follows from a proof by Nash (1951) that a (role-) symmetric game must have a (role-) symmetric Nash equilibrium, but finding a Nash equilibrium is computationally hard. Typical analysis of normal form games seeks an ϵ-Nash equilibrium, a profile $\vec{\sigma}$ with $regret(\vec{\sigma}) \leq \epsilon$. In this paper, we measure our success in approximating large games in two ways. First, we compare expected payoffs $u(s, \vec{\sigma})$ estimated by our model to ground truth expected payoffs in the game being learned, averaged over a wide range of symmetric mixtures. Second, we identify ϵ-Nash equilibria in our learned models and compute their regret in the ground-truth game. In the experiments presented here, we compute symmetric mixed-strategy equilibria using replicator dynamics (Gintis 2009), but we see similar results when computing equilibria with fictitious play.

Gaussian Process Regression

Gaussian process regression (GPR) is a flexible method for supervised learning (Rasmussen and Williams 2006) that learns a mapping from input $\vec{x}_i$ to output y_i. The set of n input points of dimension d can be collected into an $n \times d$ matrix $\mathbf{X}$, and the corresponding targets into the n-dimensional column vector $\vec{y}$. GPR estimates the value at a new point $\vec{x}$ as $k_*(\vec{x})K^{-1}\vec{y}$, where k_* and K are defined as:

$$k_*(\vec{x}) \equiv [k(\vec{x}_1, \vec{x}), k(\vec{x}_2, \vec{x}), \ldots, k(\vec{x}_n, \vec{x})]$$

$$K \equiv \begin{bmatrix} k(\vec{x}_1, \vec{x}_1) & k(\vec{x}_1, \vec{x}_2) & \cdots & k(\vec{x}_1, \vec{x}_n) \\ k(\vec{x}_2, \vec{x}_1) & k(\vec{x}_2, \vec{x}_2) & \cdots & k(\vec{x}_2, \vec{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\vec{x}_n, \vec{x}_1) & k(\vec{x}_n, \vec{x}_2) & \cdots & k(\vec{x}_n, \vec{x}_n) \end{bmatrix}$$

The kernel function $k(\cdot, \cdot)$ must be specified by the user; we use the radial basis function (RBF) kernel:

$$k\left(\vec{a}, \vec{b}\right) = c \cdot \exp\left(-\frac{1}{2l^2}\left\|\vec{a} - \vec{b}\right\|_2^2\right) \tag{2}$$

The RBF kernel has an important hyperparameter l, the length-scale over which the function is expected to vary. This can be estimated by MLE, but in our experiments, we found it important to constrain l to the range $[\,1\,, |P|\,]$, and that a length scale close to these bounds was sometimes evidence of a poor fit.

Distributions

We denote a multivariate Gaussian distribution with mean vector $\vec{\mu}$ and covariance matrix $\mathbf{\Sigma}$ as $\mathcal{N}(\cdot \mid \vec{\mu}, \mathbf{\Sigma})$. The product of two Gaussians can be rewritten in the following way (Petersen and Pedersen 2008):

$$\begin{aligned} &\mathcal{N}(\vec{x} \mid \vec{\mu}_1, \mathbf{\Sigma_1}) \cdot \mathcal{N}(\vec{x} \mid \vec{\mu}_2, \mathbf{\Sigma_2}) = \\ &\qquad \mathcal{N}(\vec{\mu}_1 \mid \vec{\mu}_2, \mathbf{\Sigma_1} + \mathbf{\Sigma_2}) \cdot \mathcal{N}(\vec{x} \mid \vec{\mu}_3, \mathbf{\Sigma_3}) \end{aligned} \tag{3}$$

where $\vec{\mu}_3$ and $\mathbf{\Sigma_3}$ are defined as

$$\vec{\mu}_3 \equiv \mathbf{\Sigma_3}(\mathbf{\Sigma_1}^{-1}\vec{\mu}_1 + \mathbf{\Sigma_2}^{-1}\vec{\mu}_2)$$
$$\mathbf{\Sigma_3} \equiv (\mathbf{\Sigma_1}^{-1} + \mathbf{\Sigma_2}^{-1})^{-1}$$

We denote the multinomial distribution of n draws from discrete distribution $\vec{p}$ as $\mathcal{M}(\cdot \mid n, \vec{p})$. When n is large and $\vec{p}$ is not near the edge of the domain, a multinomial distribution is well-approximated by the following Gaussian distribution (Severini 2005):

$$\mathcal{M}(n, p) \approx \mathcal{N}(np, M) \tag{4}$$

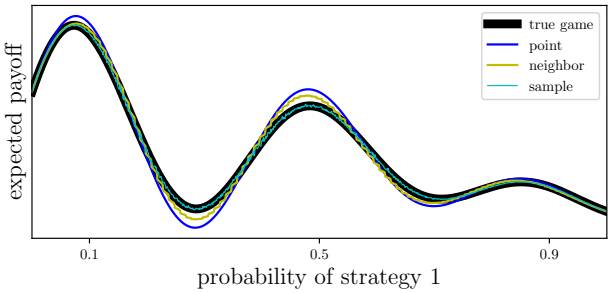
where M is defined as

$$M \equiv \begin{bmatrix} p_1 - p_1^2 & -p_1p_2 & \cdots & -p_1p_n \\ -p_2p_1 & p_2 - p_2^2 & \cdots & -p_2p_n \\ \vdots & \vdots & \ddots & \vdots \\ -p_np_1 & -p_np_2 & \cdots & p_n - p_n^2 \end{bmatrix}$$

Methods

Our method for approximating normal-form games relies on two key steps. First, we use regression to learn a mapping from pure-strategy profiles to payoffs. This mapping allows us to generalize from a small data set to functions that can be efficiently queried for arbitrary profiles. Second, we use queries to these utility functions to estimate expected payoffs of playing each pure strategy against a symmetric mixture. These expected payoff estimates enable us to compute symmetric mixed-strategy ϵ-Nash equilibria.

(a) Generic methods.



(b) GP-specific integration method.

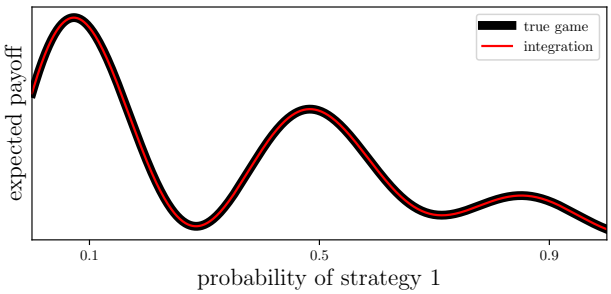


Figure 1: Comparing methods for estimating the expected payoff to strategy 1 in a 100-player, 2-strategy game learned from complete data with GPR. **Zoom recommended.**

Payoff Learning

The key insight that enables us to learn payoffs from player symmetry is that strategy profiles in a (role-) symmetric game can be encoded as a vector of strategy counts. For each strategy s, an entry in this vector encodes the number of players selecting strategy s. By representing each strategy as a separate dimension of the regression input, this method does not emphasize generalizing across strategies. Instead, it allows us to learn general effects caused by many players selecting the same strategy. Such effects are common in the literature, appearing in congestion games, local effect games, and more-specialized compact representations of large games

Further emphasizing the categorical nature of normal-form game strategies, we run a separate regression for each strategy's utility function. This serves two purposes: first, it lowers the overall runtime of the regression, and second, it allows us to specialize the data set. Given a collection of profiles and corresponding payoff values, we construct the data set for strategy s by selecting all profiles (and corresponding payoffs) where at least one player chooses s. Any regression method can be run on this data set; we focus our testing on Gaussian process regression, but have also run proof-of-concept tests using neural networks.

Data Selection In initial testing, we found that our methods produced low average error in estimating expected utilities, but surprisingly poor results in identifying Nash equilibria. The root cause of this problem was inaccurate regression estimates near the edges of the profile space (zero or one players selecting a strategy). Such profiles were rare in randomly-generated data sets, but are extremely important in computing equilibria, because in most equilibria, only a small number of strategies are played with non-zero probability (Porter, Nudelman, and Shoham 2008). As a result, the data set for all experiments that follow include a large over-representation of profiles in which zero or one players play various strategies.

Expected Payoff Estimation

Given a learned payoff model for a symmetric game, we want to identify symmetric mixed-strategy ϵ-Nash equilibria. The critical input to computing equilibria is the expected payoff of playing pure strategy s against opponents jointly following a symmetric mixture $\vec{\sigma}$, given by equation 1. Computing this expectation exactly requires summing over all profiles in the game, and is therefore infeasible in large games. We propose and evaluate several methods for estimating expected payoffs without computing the full sum.

Generic Methods We consider three methods for estimating expected payoffs that are applicable regardless what regression method was used to learn utility functions. The first method, *sampling*, selects k random profiles according the distribution $\vec{s}_i \sim Pr[\vec{s}|\vec{\sigma}]$, and computes the average payoff:

$$u(s, \vec{\sigma}) \approx \frac{1}{k} \sum_{i=1}^{k} \hat{u}(s, \vec{s}_i)$$

Where $\hat{u}$ is the regression estimate. The second method, *point*, queries the learned function only at the modal profile:

$$u(s, \vec{\sigma}) \approx \hat{u}\left(s, |P|\,\vec{\sigma}\right)$$

The third method *neighbor* computes a weighted sum over just the profiles within d deviations of the maximum-likelihood profile. Letting $\bar{s}$ be the maximum-likelihood profile, we define the set $N = \{\vec{s} \mid \|\vec{s} - \bar{s}\|_1 \leq d\}$. The neighbor estimate is then:

$$u(s, \vec{\sigma}) \approx \frac{1}{\sum_{\vec{s} \in N} Pr[\vec{s}|\vec{\sigma}]} \sum_{\vec{s} \in N} Pr[\vec{s}|\vec{\sigma}] \hat{u}(s, \vec{s})$$

All three generic methods have strengths and weaknesses. Sampling provides an unbiased estimator, and is correct in the limit as $k \to \infty$. However, it provides unstable estimates that are unsuitable for most algorithms that compute Nash equilibria. Point estimation is fast, requiring far fewer queries to the regression model than any other method. It also provides smoothly-varying estimates (subject to the smoothness of the regression model) that are correct in the limit as $|P| \to \infty$. However, its payoff estimates exhibit bias that can interfere with equilibrium computation or other analysis. Neighbor estimation provides a sort of middle ground, in that avoids the randomness of sampling and has lower bias than point estimation. When $d = 0$ neighbor approximates point, and when $d = |P|$, neighbor computes the exact expected payoff, so d can be chosen to trade off accuracy with computation time. Unfortunately, neighbor suffers from discrete-steps in its estimates as the maximum-

likelihood profile changes. This problem, illustrated in figure 1a, can occasionally prevent iterative methods for equilibrium computation from converging.

Continuous Approximation A natural approach to the problem of estimating a sum over a ver large number of low-probability terms is to approximate the summation with an integral. In this section we show how this can be done for a fully-symmetric game learned using Gaussian process regression with a radial basis function kernel. In the case of a one-role game, the probability of a profile can be expressed as a multinomial, $Pr[\vec{s}|\vec{\sigma}] = \mathcal{M}(\vec{s} \mid n-1, \sigma)$, so we can re-write equation 1 as:

$$\begin{aligned} u(s,\vec{\sigma}) &= \sum_{\vec{s}\in\vec{S}} \mathcal{M}(\vec{s} \mid n-1,\sigma)\, u(\vec{s},s) \\ &\approx \sum_{\vec{s}\in\vec{S}} \mathcal{M}(\vec{s} \mid n-1,\sigma)\, k_*(\vec{s}) K^{-1}\vec{y} \qquad (5) \\ &= \tilde{k}K^{-1}\vec{y} \qquad (6) \end{aligned}$$

Equation 5 employs the GPR payoff estimate for strategy s. We reach equation 6 by noting that $K^{-1}\vec{y}$ does not depend on $\vec{s}$, so it can be pulled out of the sum, and by defining $\tilde{k} \equiv \sum_{\vec{s}\in\vec{S}} \mathcal{M}(\vec{s} \mid n-1,\sigma)\, k_*(\vec{s})$. We next consider component i of this vector, $\tilde{k}_i$. Defining C_1 in terms of the mixture σ:

$$C_1 \equiv \begin{bmatrix} \sigma_1-\sigma_1^2 & -\sigma_1\sigma_2 & \cdots & -\sigma_1\sigma_n \\ -\sigma_2\sigma_1 & \sigma_2-\sigma_2^2 & \cdots & -\sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ -\sigma_n\sigma_1 & -\sigma_n\sigma_2 & \cdots & \sigma_n-\sigma_n^2 \end{bmatrix}$$

lets us use equation 4 to approximate $\tilde{k}_i$ with a Gaussian. Since we now have a continuous distribution, the summation can be approximated by an integration over the profile simplex.

$$\begin{aligned} \tilde{k}_i &\approx \sum_{\vec{s}\in\vec{S}} \mathcal{N}_{\vec{s}}(n\vec{\sigma}, C_1) k(\vec{s},\vec{x}_i) \\ \tilde{k}_i &\approx \int_{\vec{S}} \mathcal{N}_{\vec{s}}(n\vec{\sigma}, C_1) k(\vec{s},\vec{x}_i) d\vec{s} \qquad (7) \end{aligned}$$

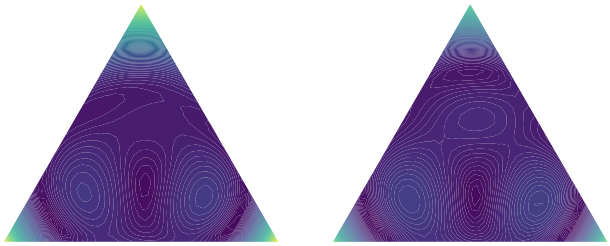


Figure 2: Comparing input data for regression in a 100-player, 3-strategy congestion game. Simplex coordinates specify a mixture, and color specifies error from true expected payoffs. Left: payoffs are estimated poorly near the corners of the simplex. Right: over-sampling edge profiles reduces this error.

We can further simplify equation 7 by re-writing the RBF kernel from equation 2 in the form of a Gaussian:

$$\begin{aligned} k(\vec{s},\vec{x}_i) &= c \cdot \exp\left(-\frac{1}{2l^2}(\vec{s}-\vec{x}_i)^T {C_2}^{-1}(\vec{s}-\vec{x}_i)\right) \\ &= c\left((2\pi)^{|S|}\det(C_2)\right)^{\frac{1}{2}} \mathcal{N}(\vec{s} \mid \vec{x}_i, C_2) \end{aligned}$$

where we define C_2 by:

$${C_2}^{-1} \equiv \begin{bmatrix} 2/l^2 & 1/l^2 & \cdots & 1/l^2 \\ 1/l^2 & 2/l^2 & \cdots & 1/l^2 \\ \vdots & \vdots & \ddots & \vdots \\ 1/l^2 & 1/l^2 & \cdots & 2/l^2 \end{bmatrix}$$

This gives us:

$$\begin{aligned} \tilde{k}_i &\approx \int_{\vec{S}} \mathcal{N}(\vec{s} \mid n\vec{\sigma}, C_1)\, c\left((2\pi)^{|S|}\det(C_2)\right)^{\frac{1}{2}} \mathcal{N}(\vec{s} \mid \vec{x}_i, C_2)\, d\vec{s} \\ &= c((2\pi)^{|S|}\det(C_2))^{\frac{1}{2}} \int_{\vec{S}} \mathcal{N}_{\vec{s}}(n\vec{\sigma}, C_1)\mathcal{N}_{\vec{s}}(\vec{x}_i, C_2) d\vec{s} \\ &= c((2\pi)^{|S|}\det(C_2))^{\frac{1}{2}} \int_{\vec{S}} \mathcal{N}(n\vec{\sigma} \mid \vec{x}_i, C_1+C_2) \cdot \\ &\qquad \mathcal{N}(\vec{s} \mid \mu', C')\, d\vec{s} \qquad (8) \\ &= c((2\pi)^{|S|}\det(C_2))^{\frac{1}{2}} \mathcal{N}(n\vec{\sigma} \mid \vec{x}_i, C_1+C_2) \cdot \\ &\qquad \int_{\vec{S}} \mathcal{N}(\vec{s} \mid \mu', C')\, d\vec{s} \qquad (9) \end{aligned}$$

Equation 8 makes use of the product-of-Gaussians identity from equation 3. This leaves us with the integration of a Gaussian distribution over the full profile simplex, which we can approximate by 1, simplifying equation 9 to:

$$\begin{aligned} \tilde{k}_i \approx& c((2\pi)^{|S|}\det(C_2))^{\frac{1}{2}} \mathcal{N}(n\vec{\sigma} \mid \vec{x}_i, C_1+C_2) \\ =& c\left[\frac{(2\pi)^{|S|}\det(C_2)}{(2\pi)^{|S|}\det(C_1+C_2)}\right]^{\frac{1}{2}} \cdot \\ & \exp\left((n\vec{\sigma}-\vec{x}_i)^T(C_1+C_2)^{-1}(n\vec{\sigma}-\vec{x}_i)\right) \\ =& c\left[\frac{\det(C_2)}{\det(C_1+C_2)}\right]^{\frac{1}{2}} \cdot \\ & \exp\left((n\vec{\sigma}-\vec{x}_i)^T(C_1+C_2)^{-1}(n\vec{\sigma}-\vec{x}_i)\right) \qquad (10) \end{aligned}$$

Equation 10 shows how we can approximate each element of the vector $\tilde{k}$, and therefore each component of equation 6 computationally. In equation 10, C_2 depends only on the length-scale l of the RBF kernel, so $\det(C_2)$ can be computed in advance and re-used for every expected payoff computation. C_1 changes with each mixture being evaluated, but given a mixture, it is the same for all pure strategies and each $\tilde{k}_i$. This means that each mixture considered in an equilibrium computation algorithm requires one inversion and one determinant calculation on an $n \times n$ matrix.

In principle, this approximation decays near the edges of the mixed-strategy simplex, because the Gaussian approximation to the multivariate distribution and the approximation of the full-simplex integral by 1 should both perform

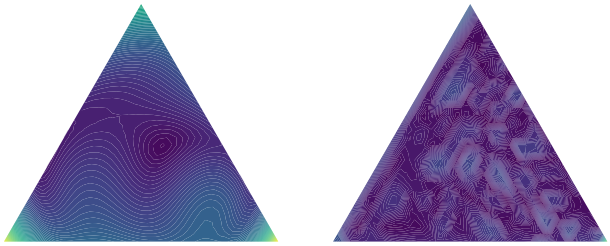


Figure 3: Comparing accuracy of expected payoff estimates by GPR and neural networks. Simplex coordinates specify a mixture, and color specifies error from true expected payoffs. Left: GPR. Right: 60-node neural network.

less well. In practice, however, we have found this decay to be small relative to the inherent inaccuracy of learning from small data sets. As shown in figure 1b, when the Gaussian process regression learns accurate payoffs, the integration methods estimates expected payoffs extremely accurately.

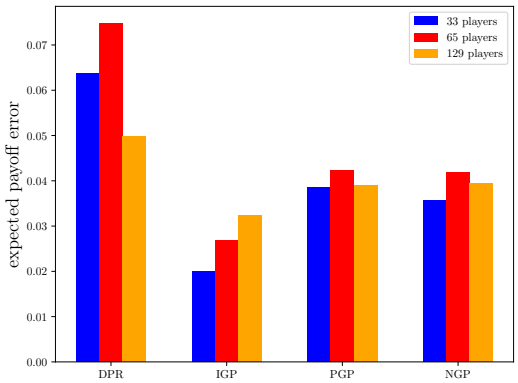
Experiments

In all of our experiments, we generate random large games, which we represent compactly as action-graph games with additive function nodes (Jiang, Leyton-Brown, and Bhat 2011). This ensures that we can compare the results of various approximation methods to a ground truth, checking the expected payoff of mixed strategies, and the regret of approximate equilibria. Previous work in approximating large games has used similar random data sets, but focused on much smaller games; we believe that our experiments on games with 100 or more players provide strong evidence that our methods can scale effectively.

In our first experiment, we isolate the effect of expected payoff estimation methods by constructing a 100-player, 2-strategy game, and providing exact payoffs for all 101 profiles as inputs to GPR. Figure 1 compares all three generic methods and the GPR-specific integration method for expected payoff estimation. Figure 1a shows that sampling (with $k = 100$) to be quite accurate, but noisy. Point estimation is smooth, but overshoots the movements of the true expected payoff. Neighbor estimation (with $d = 5$) falls somewhere in between, exhibiting moderate bias and less noise. However, note the stepped appearance of the neighbor estimates; we found that this occasionally prevented equilibrium computation from converging. Figure 1b shows excellent performance for the integration method, which is borne out through the remainder of our results.

In our second experiment, we compare different choices of input data in a 100-player, 3-strategy congestion game. In Figure 2, the simplex coordinates specify a mixed strategy: the corners correspond to all players choosing the same strategy with probability 1, and the center is the uniform distribution. Color plotted in the simplex gives the error relative to true-game expected payoffs: blue indicates low error, green indicates moderate error. expected payoffs come from GPR and point estimation in both plots. In the left-hand plot, input profiles have been spaced evenly throughout the

(a) Error vs. approximation method and game size:



(b) Error vs. approximation method and game type:

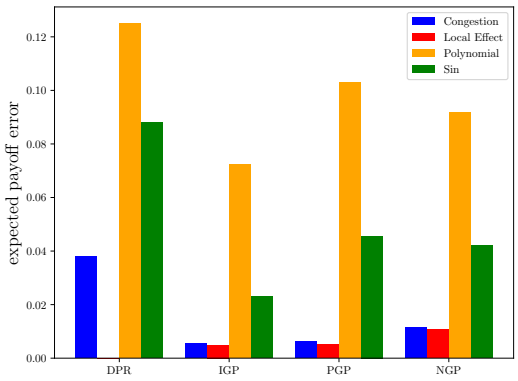


Figure 4: Learning methods compared to deviation-preserving reduction (DPR). IGP ≡ GPR + integration, PGP ≡ GPR + point, NGP ≡ GPR + neighbor. IGP performs best across all game sizes and most game types.

profile space $\vec{S}$. This results in insufficient data for accurate regression estimates near the extremes of the simplex. Because Nash equilibria often occur near edges of the simplex, especially in higher dimensions (more strategies), this can cause large errors in equilibrium estimates. In the right-hand plot, profiles have been re-allocated to the edges of the simplex, over-representing profiles with 1 or 0 players choosing a given strategy. This helps the regression to develop better estimates of the extreme profiles.

Our third experiment demonstrates that other regression methods can also be used. Figre 3 again shows errors in expected payoff estimates plotted on the probability simplex for a 100-player, 3-strategy congestion game. The left simplex of Figure 3 was created with the same settings as the right simplex of Figure 2, but a different randomly generated congestion game. The right simplex shows the accuracy of neural network learning with point-estimate expected payoffs on the same game. The neural network used hidden layers of 32, 16, 8, and 4 nodes, with sigmoid activation functions, 0.2 dropout probability and the Adam optimizer. The average error is comparable across the two methods, but the distribution of mistakes differs significantly. The neural network hyperparameters may not be sufficiently optimized.

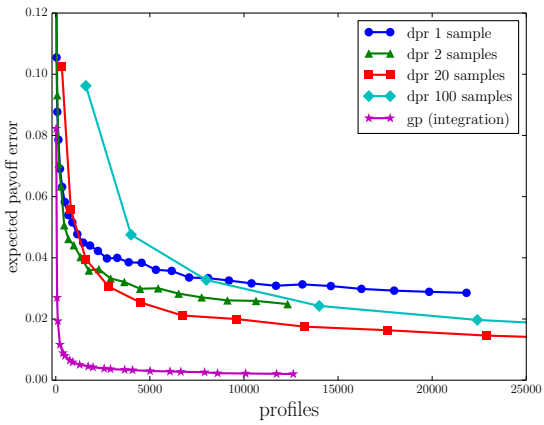
Our fourth experiment compares our regression method

against the best existing method for approximating large symmetric games, deviation-preserving reduction (Wiedenbeck and Wellman 2012). Figure 4 shows average results on a data set of 120 randomly-generated games. The games include 10 instances of each combination of parameters from (33, 65, or 129 players) and (congestion game, local effect game, action-graph game with polynomial function nodes, or action-graph game with sinusoidal function nodes). All methods were given the same amount of data, chosen to suit DPR. The number of players in the random games were also chosen to be optimal for DPR. The graph shows average error in estimating the expected payoff $u(s, \vec{\sigma})$, for a large set of symmetric mixtures, including a grid spaced evenly across the space of possible mixtures and a number of randomly generated mixtures. Despite having many parameters chosen advantageously, DPR was outperformed by the regression methods on all game sizes and nearly all game types. Among the regression methods, estimating expected payoffs by the continuous integral approximation was clearly superior. Neighbor estimation does not consistently out-perform point estimation, which suggests that it is probably not worth the extra computational burden of querying many more points. We also computed equilibria in these games, and while DPR closes the gap slightly in terms of measured true-game regret, GPR with integration remains the clear winner.

Our fifth experiment compares GPR with integration to DPR as a function of the number of profiles used as input, with noisy observations. For this experiment, we constructed a data set of 15 action-graph games with polynomial function nodes, and used rejection sampling to ensure that all games in the data set had only mixed symmetric equilibria. In the preceding experiments, all methods received correct payoff values for profiles in the data set. Here each data point has had normally-distributed noise added. In the presence of noise, it is common, before performing player reduction, to simulate the same profile multiple times for a better estimate of its payoffs. Our experiment shows that this can be a good use of simulation resources, as increasing the number of samples per profile reduces error and regret as a function of the total number of simulations over the range of 1–20 samples per profile. This effect tapers off eventually, and by 100 samples per profile, it would be better to sample more profiles fewer times.

Because regression is inherently robust to noisy inputs, our method has less need to resample the same profile repeatedly, and can sample a larger variety of profiles at the same simulation cost. As shown in Figure 5a, our method significantly outperforms all variants of DPR in terms of average error of expected payoff estimates. Our most important experimental result is shown in Figure 5b. This graph demonstrates that ϵ-Nash equilibria computed by replicator dynamics have significantly lower true-game regret under our method than under DPR.

(a) expected payoff estimation error vs. data set size:



(b) Regret of computed equilibria vs. data set size:

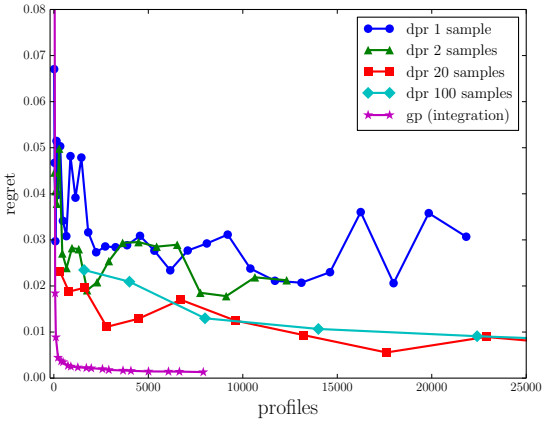


Figure 5: Learning compared to deviation-preserving reduction (DPR) with noisy payoff samples. IGP performs better in terms of error and regret whether DPR samples profiles repeatedly, or constructs a larger reduced game.

Conclusions

We have demonstrated a new method for computing approximate Nash equilibria in games with a large number of players. Our method uses player symmetries as the basis for regression-learning of pure-strategy payoffs. Using these regression estimates, we have shown that expected payoffs of mixed strategies can be estimated effectively, allowing low-regret symmetric mixtures to be identified. We provided strong experimental evidence that our methods outperform previous techniques for working with large normal-form games.

Future work on this topic should include extending the continuous regression approximation to games with multiple roles and/or to other regression methods. We would also like to combine our methods for generalizing over players with existing methods for generalizing over strategies (Vorobeychik, Wellman, and Singh 2007). The ability to use data about arbitrary profiles as input to regression opens the problem of choosing an appropriate set of profiles to simulate; it may be possible to interleave equilibrium computation and sample collection in useful ways. Finally, we hope to investigate the possibility of learning expected payoffs directly; if feasible, this could dramatically improve computational efficiency, and/or approximation performance.

References

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up limit hold'em poker is solved. *Science* 347(6218):145–149.

Cheng, S.-F.; Reeves, D. M.; Vorobeychik, Y.; and Wellman, M. P. 2004. Notes on equilibria in symmetric games. In *Sixth International Workshop on Game-Theoretic and Decision-Theoretic Agents*.

Dandekar, P.; Goel, A.; Wellman, M. P.; and Wiedenbeck, B. 2015. Strategic formation of credit networks. *ACM Transactions on Internet Technology* 15(1):3:1–41.

Duong, Q.; Vorobeychik, Y.; Singh, S.; and Wellman, M. P. 2009. Learning graphical game models. In *21st International Joint Conference on Artificial Intelligence*, 116–121.

Gintis, H. 2009. *Game Theory Evolving: A problem-centered introduction to modeling strategic behavior*. Princeton university press, second edition.

Heinrich, J.; Lanctot, M.; and Silver, D. 2015. Fictitious self-play in extensive-form games. In *32nd International Conference on Machine Learning*, 805–813.

Honorio, J., and Ortiz, L. 2015. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research* 16:1157–1210.

Jiang, A. X.; Leyton-Brown, K.; and Bhat, N. A. R. 2011. Action-graph games. *Games and Economic Behavior* 71(1):141–173.

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

Kearns, M. 2007. Graphical games. In Vazirani, V.; Nisan, N.; Roughgarden, T.; and Tardos, E., eds., *Algorithmic Game Theory*. Cambridge University Press. chapter 7, 159–180.

Monderer, D., and Shapley, L. S. 1996. Potential games. *Games and Economic Behavior* 14(1):124–143.

Moravčík, M.; Schmid, M.; Burch, N.; Lisỳ, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356(6337):508–513.

Nash, J. 1951. Non-cooperative games. *Annals of Mathematics* 286–295.

Papadimitriou, C. H., and Roughgarden, T. 2005. Computing equilibria in multi-player games. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms*, 82–91.

Petersen, K. B., and Pedersen, M. S. 2008. The matrix cookbook. Technical report, Technical University of Denmark.

Phelps, S.; Marcinkiewicz, M.; and Parsons, S. 2006. A novel method for automatic strategy acquisition in n-player non-zero-sum games. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems*, 705–712.

Porter, R.; Nudelman, E.; and Shoham, Y. 2008. Simple search methods for finding a Nash equilibrium. *Games and Economic Behavior* 63(2):642–662.

Rasmussen, C. E., and Williams, C. K. 2006. *Gaussian Processes for Machine Learning*, volume 1. MIT Press.

Severini, T. 2005. *Elements of Distribution Theory*. Cambridge Series in Statistica. Cambridge University Press.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Vorobeychik, Y.; Wellman, M. P.; and Singh, S. 2007. Learning payoff functions in infinite games. *Machine Learning* 67(1-2):145–168.

Wah, E., and Wellman, M. P. 2016. Latency arbitrage in fragmented markets: A strategic agent-based analysis. *Algorithmic Finance* 5:69–93.

Wah, E.; Hurd, D.; and Wellman, M. P. 2015. Strategic market choice: Frequent call markets vs. continuous double auctions for fast and slow traders. In *Third EAI Conference on Auctions, Market Mechanisms, and Their Applications*.

Walsh, W. E.; Das, R.; Tesauro, G.; and Kephart, J. O. 2002. Analyzing complex strategic interactions in multi-agent systems. In *AAAI-02 Workshop on Game-Theoretic and Decision-Theoretic Agents*.

Wellman, M. P.; Reeves, D. M.; Lochner, K. M.; Cheng, S.-F.; and Suri, R. 2005. Approximate strategic reasoning through hierarchical reduction of large symmetric games. In *20th National Conference on Artificial Intelligence*, 502–508.

Wellman, M. P.; Kim, T. H.; and Duong, Q. 2013. Analyzing incentives for protocol compliance in complex domains: A case study of introduction-based routing. In *12th Workshop on the Economics of Information Security*.

Wellman, M. P. 2006. Methods for empirical game-theoretic analysis (extended abstract). In *21st National Conference on Artificial Intelligence*, 1152–1155.

Wiedenbeck, B., and Wellman, M. P. 2012. Scaling simulation-based game analysis through deviation-preserving reduction. In *11th International Conference on Autonomous Agents and Multiagent Systems*, 931–938.