Structure Learning for Approximate Solution of Many-Player Games

Zun Li and Michael P. Wellman
University of Michigan, Ann Arbor
{lizun, wellman}@umich.edu

Abstract

Games with many players are difficult to solve or even specify without adopting structural assumptions that enable representation in compact form. Such structure is generally not given and will not hold exactly for particular games of interest. We introduce an iterative structure-learning approach to search for approximate solutions of many-player games, assuming only black-box simulation access to noisy payoff samples. Our first algorithm, K-Roles, exploits *symmetry* by learning a *role assignment* for players of the game through unsupervised learning (clustering) methods. Our second algorithm, G3L, seeks *sparsity* by greedy search over local interactions to learn a *graphical game* model. Both algorithms use supervised learning (regression) to fit payoff values to the learned structures, in compact representations that facilitate equilibrium calculation. We experimentally demonstrate the efficacy of both methods in reaching quality solutions and uncovering hidden structure, on both perfectly and approximately structured game instances.

1 Introduction

Many of the real-world multiagent systems we would like to understand strategically involve an enormous number of interacting (or potentially interacting) agents. For example, domains of multiagent research interest—such as ad auctions (Guo et al. 2019), financial markets (Nevmyvaka, Feng, and Kearns 2006), traffic routing (Bazzan 2009), and rumor spreading over social media (Yang et al. 2018)—all encompass (depending on the scope being considered) thousands or millions of participating agents. Straightforward game-theoretic representations of these systems do not scale well: A direct normal-form representation of an N-agent, M-action game is $O(NM^N)$, which is obviously not feasible to construct or reason about directly for even moderately large-scale multiagent systems. In response, AI researchers and others have identified various kinds of regularity that may be exhibited in such systems, particularly invoking some form of homogeneity (*symmetry*) or locality of interaction (*sparsity*) that can be exploited to develop a more compact game representation (Jiang, Leyton-Brown, and Bhat 2011; Kearns, Littman, and Singh 2001). However, it may not be apparent or easy to specify the exact structure that applies in a given multiagent scenario, and indeed it may be that no pre-identified structural simplification holds exactly for a problem of interest. We therefore investigate in this work the possibility of learning such structure, and moreover doing so in an iterative manner interleaved with game-theoretic reasoning about structured game models as they are developed.

We develop our methods in the framework of empirical game-theoretic analysis (EGTA) (Wellman 2016; Tuyls et al. 2018). EGTA assumes as input a representation of the game in terms of a payoff oracle (e.g., a simulator). The game analyst may sample this *simulation-based game* by querying the oracle to obtain data from which to estimate or induce a game model, called the *empirical game*. This framework makes sense for the problem at hand, because specifying simulation models does not suffer from the curse of dimensionality that inhibits scaling explicit game models to large numbers of agents. The challenge is to make effective use of the simulator to gain game-theoretic insights on such large multiagent systems.

Our hypothesis is that the game-learning process can be enhanced by a focus on the structure of agent populations and interactions. For population structure, we appeal to the partition of agents into distinct *roles*. Many applications have obvious role dichotomies: investors and traders in financial markets, commuters and vacationers in road traffic, brand and sales marketers in advertising. More generally, we expect that many multiagent systems will at least roughly support classification into broad roles. For interaction structure, we appeal to locality. For example, on a social network, one is directly influenced (by rumors, innovations, etc.) primarily through one's connections on the network. Both kinds of structure—role-organized symmetry and locality of influence—have been formalized in terms that support compact game representations. Whereas the formal requirements for compact representation may not strictly

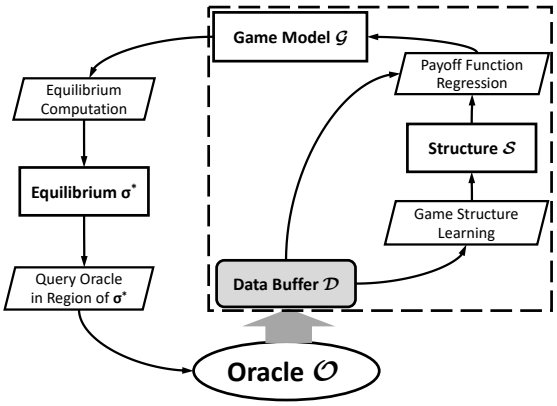


Figure 1: Iterative game model learning and solving. The dashed box encompasses the model learning components.

hold for games of interest, we expect that they will often hold approximately to a useful degree. If so, it is worth trying to identify that useful structure in payoff samples, thereby enabling induction of more compact and sample-efficient game representations, and accordingly supporting simpler and more reliable game-theoretic reasoning.

Our approach draws on a broad variety of machine learning techniques. Inspired by the model-based reinforcement learning framework (Sutton and Barto 2018, Sec. 8.2), we build our iterative learning-and-solving architecture diagrammed in Figure 1. The underlying game is represented by a simulator, $\mathcal{O}$, providing black-box oracle access. The only explicit game descriptors are the sets of agents and actions. Starting with an arbitrary guess solution $\boldsymbol{\sigma}^*$, on each iteration, the method first queries $\mathcal{O}$ in the region of $\boldsymbol{\sigma}^*$, obtaining by this online sampling process a new dataset $\mathcal{D}_{val}$, which is added to the data buffer $\mathcal{D}$. Through offline interaction with $\mathcal{D}$, we then learn and solve a game model to reach the next $\boldsymbol{\sigma}^*$. The learning process encompasses two steps: It first discovers the approximate hidden structure from payoff data, and then enables payoff regression by feeding both data and learned structure to function approximators. The sampling process across iterations is designed to concentrate the data buffer $\mathcal{D}$ around candidate solution candidates, thus prioritizing the quality of generalization on the most relevant regions. The method employs offline computation and storage with the aim of limiting online sample complexity, in service of effective reasoning about large-scale simulation-based games.

We propose two algorithms instantiating our framework: K-Roles for learning role-symmetry, and Greedy Graphical Game Learning (G3L) for learning graphical structure in a game model. We begin by introducing necessary background information in Section 2, followed by Section 3 reviewing related work on game model learning and solving. Algorithmic details of K-Roles and G3L are covered respectively in Sections 4 and 5. Section 6 presents our evaluation on both perfectly and approximately structured game instances. Conclusion and insights on methods developed are drawn in Section 7.

2 Preliminaries

2.1 Normal Form Games

In an N-player normal form game $\mathscr{G}$, player (or *agent*) $n \in \mathscr{N} = \{1, \ldots, N\}$ chooses its action $a_n \in \mathscr{A}_n$, and receives payoff $u_n(\boldsymbol{a})$ as a function of the agents' joint action or *action profile* $\boldsymbol{a}$.[1] We assume agents share a universal finite action set: $\forall n.\ \mathscr{A}_n = \mathscr{A} = \{1, \ldots, M\}$. Thus, $\boldsymbol{a} \in \mathscr{A}^N$, and for convenience, we write payoff functions in a form $u_n(a_n, \boldsymbol{a}_{-n})$ that separates the subject agent's action and the vector of other-agent actions in distinct arguments. A mixed strategy σ is a probability distribution induced over $\mathscr{A}$. The payoff for n under a joint mixed strategy $\boldsymbol{\sigma}$ is $u_n(\boldsymbol{\sigma}) \triangleq \mathbb{E}_{\boldsymbol{a} \sim \boldsymbol{\sigma}}[u_n(a_n, \boldsymbol{a}_{-n})]$. The *deviation payoffs* vector for n under $\boldsymbol{\sigma}$ is $\nabla_{\sigma_n} u_n(\boldsymbol{\sigma}) = \left(\frac{\partial u_n(\boldsymbol{\sigma})}{\partial \sigma_{n,1}}, \ldots, \frac{\partial u_n(\boldsymbol{\sigma})}{\partial \sigma_{n,M}}\right)^{\mathbf{T}}$, where the m^{th} element $\frac{\partial u_n(\boldsymbol{\sigma})}{\partial \sigma_{n,m}} = u_n(m, \boldsymbol{\sigma}_{-n}) \triangleq \mathbb{E}_{\boldsymbol{a}_{-n} \sim \boldsymbol{\sigma}_{-n}}[u_n(m, \boldsymbol{a}_{-n})]$, that is, the payoff of n choosing m while others act according to $\boldsymbol{\sigma}$.

2.2 Approximate Nash Equilibrium

Define the *regret* of agent n at $\boldsymbol{\sigma}$ as $\textsc{Regret}_n(\boldsymbol{\sigma}) \triangleq \max_{a_n} u_n(a_n, \boldsymbol{\sigma}_{-\boldsymbol{n}}) - u_n(\boldsymbol{\sigma})$. The overall regret of the game at $\boldsymbol{\sigma}$ is $\textsc{Regret}(\boldsymbol{\sigma}) \triangleq \max_n \textsc{Regret}_n(\boldsymbol{\sigma})$, and if $\textsc{Regret}(\boldsymbol{\sigma}) \leq \epsilon$, we call $\boldsymbol{\sigma}$ an ϵ-*Nash equilibrium*. We typically seek solutions to minimize $\textsc{Regret}$, or alternately $\textsc{NashConv}(\boldsymbol{\sigma})$, defined as the aggregate regret over agents: $\sum_n \textsc{Regret}_n(\boldsymbol{\sigma})$ (Lanctot et al. 2017; Srinivasan et al. 2018).

2.3 Succinct Games

Without further structural assumptions, the representation complexity of an N-agent, M-action game is $O(NM^N)$, which is generally intractable both for storing a game description and computing its solution. Therefore, extensive work has been directed at identifying *succinct game* representations (Daskalakis, Goldberg, and Papadimitriou 2009).

In an *anonymous game* (Daskalakis and Papadimitriou 2015), agent n's payoff depends only on its action and how many (or equivalently, what fraction of) agents choose each action: $u_n(a_n, \boldsymbol{a}_{-n}) = u_n(a_n, f_1, \ldots, f_M)$, with f_m counting the frequency of agents choosing m. If furthermore $\forall n.\ u_n = u$, the game is *symmetric*. A *role-symmetric game* generalizes this concept by introducing asymmetry: Agents are partitioned into different *roles*, where agents within the same roles are interchangeable from the view of others. Let $\mathcal{R}(n) \in \{1, \ldots, K\}$ denote the role for agent n. Then

[1] Actions here may correspond to complex strategies; we refer to action and strategy profiles interchangeably.

the payoff for agent n depends on its action and the action distribution within each role: $u_n(a_n, \boldsymbol{a}_{-n}) = u_n^{\mathcal{R}}(a_n, f_{1,1}, \ldots, f_{1,M}, \ldots, f_{K,M})$, where $f_{k,m}$ is the action frequency of m within role k. For a role-symmetric game, we are typically interested in role-symmetric profiles: $\forall n, n'.\ \mathcal{R}(n) = \mathcal{R}(n') \implies \boldsymbol{\sigma}_n = \boldsymbol{\sigma}'_n$. For finite role-symmetric games, equilibria are guaranteed to exist in role-symmetric profiles (Nash 1951).

A second category of succinct representations, *graphical games* (Kearns, Littman, and Singh 2001), capture *sparsity* in multiagent interaction. By assuming agent n's payoff depends only on the joint action profile over its neighborhood $\mathcal{N}(n)$ on an interaction graph, $u_n(a_n, \boldsymbol{a}_{-n}) = u_n(a_n, \boldsymbol{a}_{\mathcal{N}(n)})$, the representation complexity is reduced to $O(NM^{\kappa})$, where κ is the maximum size of a neighborhood.

2.4 Empirical Game Models

The methodology of *empirical game-theoretic analysis* (EGTA) employs simulation or sampling to induce a game model. This approach is called for when it is not feasible to express a game model in analytic form, either due to representation complexity or difficulty of manual specification. Formally, in EGTA the multiagent environment is represented by a *game oracle* $\mathcal{O}$ (e.g., a simulator), which can be queried to generate a dataset $\mathcal{D}$ of action-payoff tuples $(\boldsymbol{a}, \boldsymbol{u})$, where $\boldsymbol{u}$ is either an exact or noisy sample of the payoff vector associated with action profile $\boldsymbol{a}$. A normal-form game model induced from $\mathcal{D}$ is called an *empirical* game.

In most EGTA studies, the dominant cost is that of simulating action profiles (i.e., querying the oracle). Accordingly, several prior works have addressed the problem of controlling the sampling process to maximize analysis value while minimizing query costs (Jordan, Vorobeychik, and Wellman 2008; Walsh, Parkes, and Das 2003), and have obtained theoretical bounds in some cases (Viqueira et al. 2019; Goldberg and Turchetta 2017). Our work can be viewed in this line, distinguished by its focus on identifying structure both to improve generalization and facilitate reasoning.

2.5 Game Model Learning

Game model learning in our setting aims to induce a representation of a game, within a specified *hypothesis game space*, from limited payoff data using standard machine learning methods. The hypothesis spaces of interest correspond to succinct game formats where practical structure-exploiting Nash solvers exist. For example, one can apply *replicator dynamics* (Sandholm 2010, Section 4.3.1) or *function minimization* (McKelvey and McLennan 1996) to solve a role-symmetric game, and *homotopy method* (Blum, Shelton, and Koller 2006) or *hybrid refinement algorithm* (Vickrey and Koller 2002) to a graphical game. Given a hypothesis space $\mathcal{H}$, one preprocesses data point $(\boldsymbol{a}, \boldsymbol{u})$ to construct the *features* on the raw $\boldsymbol{a}$ according to $\mathcal{H}$. For a role-symmetric game it is to aggregate the action frequency over different roles, thus it can be viewed as *feature extraction*; while for a graphical game it is to eliminate agent dependency, thus it can be interpreted as *feature selection*. One therefore builds an empirical game by learning a mapping from the set of features to the set of payoffs.

3 Related Work

Computational Game Theory The idea of using iterated game approximation to find equilibrium dates back to the *homotopy method* (Govindan and Wilson 2003; Herings and Peeters 2010) in classic computational game theory. Homotopy method typically starts from a perturbation of the original game model with a trivial solution. By keeping track of the perturbation vector and equilibrium along a homotopy path, it is guaranteed to reach the equilibrium of the origin game. One elegant instantiation is the *iterated polymatrix approximation algorithm* (Govindan and Wilson 2004; Blum, Shelton, and Koller 2006), which approximates the original game as a sequence of polymatrix games and solves them by some efficient subroutine such as the Lemke-Howson algorithm.

Multiagent Simulation for Game Model Learning In simulation-based game model learning, the analyst samples a variety of strategy profiles and receives data in the form of (profile,payoff-vector) for model learning (Vorobeychik, Wellman, and Singh 2007).

The only prior work we are aware of that expressly exploits clustering for learning a normal form game model is by Ficici, Parkes, and Pfeffer (2008). Their approach starts by clustering agents via k-means ($k = 2$) according to the average payoff vector in the dataset. They then use linear regression to estimate mixed-strategy payoffs for the clusters. These regressors are in turn used to construct the pure-strategy payoff table of a reduced two-player approximation of the game. They compute a Nash equilibrium of this game, and ascribe the resulting mixed strategies to agents in the respective clusters. Our method for learning role-symmetry structure can be viewed as a variation of theirs that: (1) reduces dimensionality by clustering payoff deviation functions rather than payoff functions, (2) allows for more clusters, and (3) solves the role-symmetric game rather than a reduced game.

There also has been prior work on regression for role-symmetric games, for given role assignments (Wiedenbeck, Yang, and Wellman 2018; Sokota, Ho, and Wiedenbeck 2019). Duong et al. (2009) and Fearnley et al. (2015) studied algorithms for inducing structure of graphical games.

These works typically assume that training data collection through simulation is fully controlled by the analyst. This makes the setting akin to *active learning* (Settles 2009). Thus, the simulation-based approach could also be regarded as a semi-supervised style of game model learning.

Game Model Learning from Behavioral Data Another line of work employs behavioral data for game model learning. Here the data are typically assumed to be generated from approximate equilibria repeatedly played by bounded rational agents. In contrast to the simulation-based approach described above, the observational data here takes the form of actions rather than payoffs.[2] Thus, we classify this style of game model learning as *unsupervised*, and note that it can also be viewed as a multiagent form of inverse reinforcement learning (Ng and Russell 2000).

The focus of prior work in this area has been to recover underlying game structure. For example, Honorio and Ortiz (2015) adopt a specific generative model, and optimize the fit of key parameters to the available data. Other works also employ diverse optimization techniques to uncover the payoff matrix under a best-response constraint (Kuleshov and Schrijvers 2015; Waugh, Ziebart, and Bagnell 2011; Ling, Fang, and Kolter 2018; 2019). Models learned with such techniques have been shown to fit well to some real-world scenarios (Garg and Jaakkola 2016; 2017).

4 K-Roles: Learning Role Symmetry

We now describe a specific algorithm that employs structure learning, under the hypothesis that the game approximately exhibits role symmetry structure. The *K-Roles* algorithm follows the basic template of Figure 1, with structure defined by a mapping of players to roles, players within each role treated symmetrically. The method combines unsupervised methods (clustering) for structure learning, with supervised techniques (regression) for payoff estimation. Though the target game is in general not perfectly role-symmetric for $\hat{K} < N$, we may still expect it to exhibit approximate role structure for some reasonable number of roles.

4.1 Overview

As shown in Algorithm 1, our approach employs a hyperparameter, $\hat{K}$, denoting the number of roles in the game model. In each iteration the algorithm first augments the payoff dataset by sampling near the current candidate solution. It then estimates deviation payoff vectors for each player (DEVEST in Algorithm 1), and assigns players to roles through an unsupervised clustering method (DETCLUSTER). A new payoff function is then learned by regression (FITREGRESSOR), taking the derived role assignment as a constraint. With role symmetry, the regression essentially entails training payoffs for $\hat{K}$ separate "role agents", each representing its respective role as determined from the clustering operation. Finally, we compute a role-symmetric mixed Nash equilibrium (NASHSOLVER) for the game model at the current iteration.

[2]Gao and Pfeffer (2010) study game learning based both on payoff data and assumed rationality of action choice.

Algorithm 1: K-Roles

Input: Hyperparameter $\hat{K}$, Oracle $\mathcal{O}$.
Initial solution $\boldsymbol{\sigma}^*$, Data buffer $\mathcal{D} = \{\}$;
repeat
  $\mathcal{D}_{val} \leftarrow \text{QUERY}(\mathcal{O}, \boldsymbol{\sigma}^*)$;
  $\nabla_{\boldsymbol{\sigma}^*} \hat{u} \leftarrow \text{DEVEST}(\mathcal{D}_{val})$;
  $\mathscr{C} \leftarrow \text{DETCLUSTER}\left(\nabla_{\boldsymbol{\sigma}^*} \hat{u}, \mathcal{D}, \mathcal{D}_{val}\right)$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{val}$;
  $\{\nabla_{\boldsymbol{\sigma}} \mathscr{R}_k\}_{k \in [\hat{K}]} \leftarrow \text{FITREGRESSOR}(\mathscr{C}, \mathcal{D})$;
  $\boldsymbol{\sigma}^* \leftarrow \text{NASHSOLVER}\left(\{\nabla_{\boldsymbol{\sigma}} \mathscr{R}_k\}_{k \in [\hat{K}]}\right)$;
until $\boldsymbol{\sigma}^*$ *sufficiently close to equilibrium*;

By imposing role symmetry and a regression model, we transform the problem from a combinatorial optimization of dimension $O(NM^N)$ to a continuous optimization of dimension $O(M^2\hat{K}^2)$.

4.2 Structure Learning

We propose two heuristic agent clustering methods based on well-known partitional clustering algorithms: k-means and hierarchical clustering. We represent each agent by its individual point deviation payoffs based on the current model and the validation data set $\mathcal{D}_{val}$, where the deviation payoff estimator $\nabla_{\boldsymbol{\sigma}^*} \hat{u}_{n,m}$ is calculated by taking the average of the payoffs when agent n chooses action m in $\mathcal{D}_{val}$. If no data is available for action m, we simply take it as n's average payoff across all actions. The task here is to cluster these agents by similarity of strategic view into $\hat{K}$ roles.

The first method is similar to the one adopted by Ficici, Parkes, and Pfeffer (2008): We directly apply k-means ($k = \hat{K}$) on the vector embeddings to obtain a cluster assignment. Per the k-means procedure, on each iteration we calculate the centroids of each cluster based on the current assignment, then update the clustering by assigning each agent to the cluster to which it is closest based on these centroids.

In the second method we define distance measures between any pair of agents (i, j) and then perform hierarchical clustering. Here we use a weighted L^p-norm between deviation payoffs as the distance metric. Specifically we compute $\|u_i - u_j\|_{\boldsymbol{\sigma},p}$ as

$$\left(\sum_{a_i}\sum_{a_j} \sigma_{i,a_i}\sigma_{j,a_j} |u_i(a_i, \boldsymbol{\sigma}_{-i}) - u_j(a_j, \boldsymbol{\sigma}_{-j})|^p\right)^{1/p},$$

at the latest equilibrium point $\boldsymbol{\sigma} = \boldsymbol{\sigma}^*$ with $p \geq 1$. We use $\nabla_{\boldsymbol{\sigma}^*} \hat{u}_{i,a_i}$ to estimate $u_i(a_i, \boldsymbol{\sigma}^*_{-i})$. We adopt the version of agglomerative average linkage clustering: Starting from N singletons, in each iteration for any pair of current clusters we calculate the average inter-cluster distance, and merge the pair that minimizes that until we reach $\hat{K}$ clusters.

In the experiments described below, we employ both clustering methods for K-Roles. First we try hierarchical

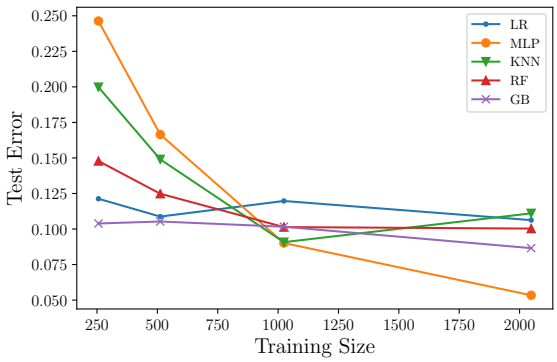


Figure 2: Performance of deviation payoff estimations for linear regression (LR), multilayer perceptron (MLP), k-nearest-neighbor (KNN) with $k = 5$, random forest (RF), and gradient boosting (GB). Training data are corrupted with Gaussian noise width 0.2^2.

agent clustering with $p = 2$. If any returned cluster is of size below 20, we discard the result and apply k-means clustering instead.

4.3 Payoff Function Regression

Given a cluster assignment $\mathscr{C}$ derived from the preceding step, we perform regression (FITREGRESSOR in Algorithm 1) to estimate deviation payoffs for each action of each role agent. Define $\nabla_{\boldsymbol{\sigma}}\mathscr{R}_k = (\nabla_{\boldsymbol{\sigma}}\mathscr{R}_{k,1}, \ldots, \nabla_{\boldsymbol{\sigma}}\mathscr{R}_{k,M})^{\mathbf{T}}$ as the vector of deviation payoffs to learn for role k, where the m^{th} element is the payoff for an individual of role k playing m while others act according to the role-symmetric strategy $\boldsymbol{\sigma}$. We compute that by aggregating the payoff information according to $\mathscr{C}$.

Specifically, given partition $\mathscr{C} = \{\mathcal{R}_1, \ldots, \mathcal{R}_{\hat{K}}\}$, we organize the data for each role as follows. For a raw data point $(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}$, we first calculate the action counts (empirical distribution) from $\boldsymbol{a}$ for each role and concatenate them as the feature vector $\boldsymbol{f}$, and then for each agent n of role k we store $(\boldsymbol{f}, (\boldsymbol{u})_n)$ as a data point for training $\nabla_{\boldsymbol{\sigma}}\mathscr{R}_{k,(\boldsymbol{a})_n}$. This corresponds to the *point method* proposed by Wiedenbeck, Yang, and Wellman (2018) for mixed payoff estimation.

We tried a variety of regression methods; Figure 2 plots their performance for a random role-symmetric game with $N = 300, M = 3, K = 3$, trained according to the ground-truth role partition. We sample 500 role-symmetric mixed strategies from a Dirichlet distribution as the test set, and define the test error to be the L2 loss between the regressor predictions and the true deviation payoffs, averaging across all roles and all actions. For each profile we estimate these target deviation payoffs via 1000 samples from the oracle. We find that generally gradient boosting and multilayer perceptron regression methods outperform the others in terms of accuracy and robustness against noise, while the latter generalize better with sufficient amount of training data.

4.4 NASHSOLVER

After we have trained the role agents $\{\mathscr{R}_1, \ldots, \mathscr{R}_{\hat{K}}\}$, they naturally define a role-symmetric game, where the action space for each role is an M-dimensional simplex consisting of all possible action distributions within that population. We resort to function minimization (McKelvey and McLennan 1996) to attain a role-symmetric equilibrium.

Remark 1 The regressed model supports $O(1)$ access to the deviation payoffs of role-symmetric mixed strategies needed for NASHSOLVER as defined here. This avoids the infeasible multiplication and summation over payoff matrices for deviation calculation employed in classical Nash algorithms.

Remark 2 The learned role-symmetric game model represented by differentiable function approximators is a special case of *differentiable game* (Balduzzi et al. 2018). Optimization techniques designed for that class could therefore also be applied in an alternative NASHSOLVER.

5 G3L: Learning Graphical Structure

5.1 Overview

Our second algorithm operates under the hypothesis that the game approximately exhibits graphical dependence structure. *G3L* (Algorithm 2) employs a refinement of the greedy loss minimization approach of Duong et al. (2009) for structure learning. The procedure also resembles score-based structure learning in probabilistic graphical models (Heckerman, Geiger, and Chickering 1995), and greedy forward feature selection in representation learning (Friedman, Hastie, and Tibshirani 2001, Sec. 3.3.2).

5.2 Structure Learning

For each iteration we maintain an estimated neighborhood set $\hat{\mathcal{N}}(n)$ for each agent n. All such $\hat{\mathcal{N}}(n)$ would define a directed graph for a graphical game. Initialized as $\{n\}$, we perform a sequence of local searches on $\hat{\mathcal{N}}(n)$ until convergence: Each time we either add a new neighbor or delete an old one from $\hat{\mathcal{N}}(n)$, such that the training loss (described next subsection) would decrease the most. Furthermore to control model complexity for efficient deviation computation during equilibrium calculation, we employ a regularizer $\hat{\kappa}$ to constrain the maximum neighborhood size.

5.3 Payoff Function Regression

For a given profile $\boldsymbol{a}_{\hat{\mathcal{N}}(n)}$ of the learned graphical game, we set $\hat{u}_n$ to the average of all $(\boldsymbol{u})_n$, such that $(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}$ and $\boldsymbol{a}_{\hat{\mathcal{N}}(n)}$ is contained in $\boldsymbol{a}$. If $\boldsymbol{a}_{\hat{\mathcal{N}}(n)}$ does not appear in any profile of $\mathcal{D}$, we just set $\hat{u}_n$ to the average payoff of n choosing a_n in $\mathcal{D}$. The training loss for a given

Algorithm 2: Greedy Graphical Game Learning

Input: Hyperparameter $\hat{\kappa}$, Oracle $\mathcal{O}$.

Initial solution $\boldsymbol{\sigma}^*$, Data buffer $\mathcal{D} = \{\}$;

repeat

  $\mathcal{D}_{val} \leftarrow \text{QUERY}(\mathcal{O}, \boldsymbol{\sigma}^*)$;

  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{val}$;

  for $n \in \mathcal{N}$ **do**

    $\hat{\mathcal{N}}(n) = \{n\}$;

    repeat

      $\hat{\mathcal{N}}(n) \leftarrow \underset{\substack{S: |S \triangle \hat{\mathcal{N}}(n)| \leq 1 \\ |S| \leq \hat{\kappa}}}{\arg\min} \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}} L(\hat{u}_n(S, \boldsymbol{a}), (\boldsymbol{u})_n)$;

    until $\hat{\mathcal{N}}(n)$ *converged*;

  end

  $\boldsymbol{\sigma}^* \leftarrow \text{NASHSOLVER}\left(\left\{\hat{u}_n(\hat{\mathcal{N}}(n)\right\}_{n \in \mathcal{N}}\right)$;

until $\boldsymbol{\sigma}^*$ *sufficiently close to equilibrium*;

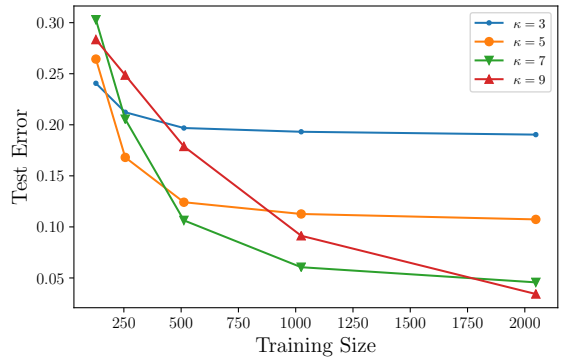


Figure 3: Performance of greedy forward learning for pure-strategy payoff estimation under different $\hat{\kappa}$. Training data are corrupted with Gaussian noise width 0.2^2.

$\hat{\mathcal{N}}(n)$ is defined as the average of L2 loss between payoff prediction $\hat{u}_n(\hat{\mathcal{N}}(n), \boldsymbol{a})$ and target payoff $(\boldsymbol{u})_n$ for all $(\boldsymbol{a}, \boldsymbol{u}) \in \mathcal{D}$.

We plot in Figure 3 the performance of greedy forward learning on a random graphical game, for different regularizer $\hat{\kappa}$. We sample 500 pure strategy profiles according to uniform mixed strategy as the test set, and define the test error as the L2 loss between the regressor predictions and these ground truth pure strategy payoffs. We find that in general the hyperparameter $\hat{\kappa}$ trade-offs prediction accuracy for sample complexity: A bigger $\hat{\kappa}$ represents a richer model class while requires more data to make good estimations.

5.4 NASHSOLVER

We implement an optimized version of the Govindan-Wilson algorithm (Blum, Shelton, and Koller 2006) to solve for exact mixed NE of the learned graphical game. The graphical structure is exploited through the efficient computation of deviation payoffs.

Remark With many players, the main computational bottleneck of the Govindan-Wilson algorithm is calculating the adjugate matrix for the payoff Jacobian. Here we resort to the method proposed by Stewart (1998), which utilizes perturbated decomposition to compute the adjugate as well as to handle the case when it is singular. We adopt the "wobble" trick and adaptive step size described by Blum, Shelton, and Koller (2006) to speed up convergence. Other parameters are the same as the implementations in GameTracer (Blum, Koller, and Shelton 2002).

6 Experiments

In this section, we evaluate our methods on both perfectly and approximately structured games. For games with perfect role structure, we generate the cluster assignment from a uniform distribution. For games with an underlying graphical model, we generate a directed random graph with expected number of neighbors 5. All payoff samples are added with Gaussian noise of width 0.2^2 when returned by the oracle.

For a game with a perfect role structure, we validate the model accuracy by the normalized Rand Statistics (Rand 1971) within range [0,1], between the clusters output by the algorithm and the ground-truth. The higher the score the more similar two partitions are.

For a game with an underlying graph $\{\mathcal{N}(n)\}_n$, we define the *graph score* for the learned graph $\{\hat{\mathcal{N}}(n)\}_n$ as $GS = \frac{1}{N} \sum_n \frac{|\mathcal{N}(n) \cap \hat{\mathcal{N}}(n)|}{|\mathcal{N}(n)|} \in [0, 1]$, measuring how well the learned graph resembles the original.

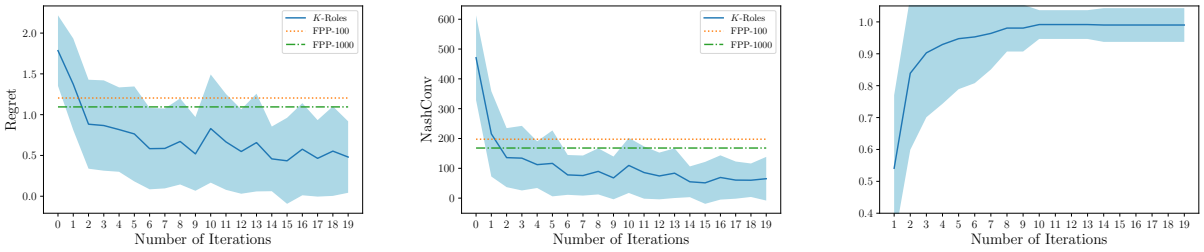
6.1 Random Role-Symmetric Games

We first test on a 300-agent 3-action, 3-role random role-symmetric game. We evaluate K-Roles choosing $\hat{K} = 3$ against the method of (Ficici, Parkes, and Pfeffer 2008) trained with 100 and 1000 data points, denoted as FPP-100 and FPP-1000 respectively. We maintain a data buffer of size 1000, and query 100 data points as $\mathcal{D}_{val}$ in each iteration. For regression of deviation function approximators, we use a neural network with two hidden layers of sizes 32 and 16.

As shown in Figures 4(a) and 4(b), K-Roles is able to find better solutions than FPP within a few iterations. Figure 4(c) shows that the procedure also succeeds in recovering the ground-truth role partition quickly, starting from a random initial assignment. The progress in finding better solutions with iteration can be attributed to both improved cluster representation and accumulation of data for payoff training.

6.2 Biased Voting Game

The biased voting game (Kearns et al. 2009) is a graphical game model designed to capture a tradeoff between



(a) Regret (b) NashConv (c) Rand Statistics with Ground-truth

Figure 4: Performance of K-Roles over iterations. The results are averaged over 30 runs.

Table 1: Performance of G3L versus IBR on biased voting game instances. The entry format in the second and third columns is REGRET(NASHCONV). The last column gives the graph scores for the graphs learned by G3L. All results are averaged over 20 runs.

Game Instance	G3L	IBR	G3L GS
$N = 100, M = 2$	0.165 (0.337)	0.315 (0.797)	0.907
$N = 100, M = 3$	0.298 (1.447)	0.513 (1.601)	0.698
$N = 200, M = 2$	0.075 (0.207)	0.661 (5.178)	0.915

maximizing one's own preferences and coordinating with neighbors. In an M-party biased voting game, agent n has a *preference score* $s_{n,m}$ for each party m. And if $f_{n,m}$ fraction of n's neighbor votes for party m, the payoff of n voting party m is $s_{n,m}f_{n,m}$.

We first test on three games with different degrees of solving difficulty. In each experiment we query 1000 data points for one shot. We adopt Iterated Best Response (IBR) as our benchmark with the same number of data points queried. In each round of IBR a player is randomly selected to make a best response to the current state. IBR is guaranteed to reach pure-strategy Nash equilibrium (PSNE) for network games with strategic complements (Jackson 2010, Section 9.3.3), however in general for network games and the biased voting game in particular, PSNE may not even exist.

We set $\hat{\kappa} = 6$ if $M = 2$ and $\hat{\kappa} = 4$ when $M = 3$. The results are shown in Table 1. We observe that choosing a large $\hat{\kappa}$ typically results in a fairly accurate graphical model, but since the size of payoff table for the learned game grows exponentially with $\hat{\kappa}$, we need to sacrifice model accuracy for efficient equilibrium computation when facing moderate M. Nevertheless the solutions returned by G3L exhibited better quality than IBR in all instances tested.

We then perform an iterated version of the experiment $N = 100, M = 2$, where buffer size is 1000 and 100 data points are queried in each iteration. As shown in Figures 5, G3L beats the baseline even at the first iteration, and its capability of finding a good solution as well as recovering the graph structure evidently improves over iterations.

6.3 Criminal Network Game

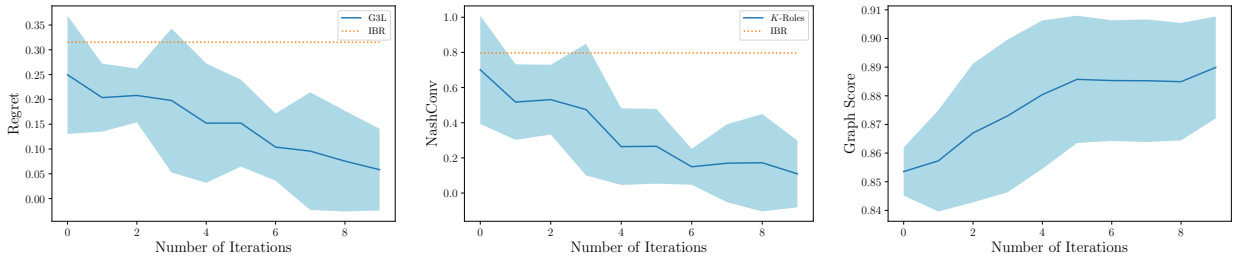
In the peer-effect *criminal network game* studied by Bramoullé, Kranton, and D'Amours (2014), each agent n is embedded in a graph G and has to choose a criminal level $a_n \in [0, 1]$. The payoff for agent n is defined as $u_n = y_n - \zeta \cdot x_n$. The symmetric game term y_n is a function of the total criminal levels of this network, capturing the competing effects and satisfying conditions $\frac{\partial y_n}{\partial a_n} \geq 0, \frac{\partial y_n}{\partial a_{n'}} \leq 0, \forall n' \neq n$. The graphical game term x_n measures peer-effects and satisfies $\frac{\partial x_n}{\partial a_n} \geq 0, \frac{\partial x_n}{\partial a_{n'}} \leq 0, \forall n' \in \mathcal{N}(n)$.

For our purposes, the most interesting feature of the criminal network game is that by varying the structure parameter $\zeta \geq 0$ for fixed y_n, x_n, we obtain a spectrum of games between perfect symmetry and perfect sparsity: With greater $\zeta \geq 0$ the game is closer to a graphical game as opposed to a symmetric game, and vice versa.

We let $M = 3$ by constraining the criminal level $a_n \in \{0, 0.5, 1\}$. We fix linear-quadratic functions for both y_n and x_n and vary ζ. The results of K-Roles and G3L are based on a single iteration of game learning with 1000 query samples. The solution qualities of the methods for game points of different structures are plotted in Figure 6. We find that K-Roles and G3L, respectively, outperform the others when the game is closer to the symmetry or the sparsity extreme. Interestingly, when the game approaches a graphical game, K-Roles is able to discover an approximate role structure when the sparsity increases, with solutions nearly as good as those found by G3L. This suggests that symmetry can arise from sparsity in a game, and validates the efficacy of K-Roles in revealing such structure.

7 Conclusion

Scaling game modeling to large numbers of players perhaps inevitably requires some structural regularity in the situation and interactions of the agent population. We proposed an approach to reasoning about many-player games based on iterative structure learning, given black-box access to noisy payoff samples for a target game. Starting from a basic structural hypothesis, our method learns a candidate structure and associated game model, then gathers additional training data based on



(a) Regret (b) NashConv (c) Graph Score with Ground-truth

Figure 5: Performance of G3L over iterations. The results are averaged over 10 runs

.

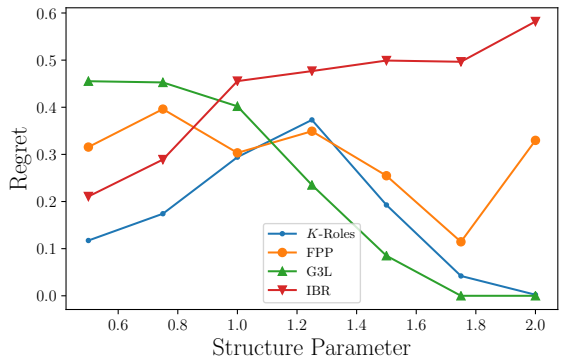


Figure 6: Performance of K-Roles, FPP, G3L, IBR on a 100-player, 3-action criminal network game instance. $\hat{K} = 2$ for K-Roles and FPP. Results are averaged over 10 runs.

this candidate to learn a refined model. We instantiated this approach for two very different forms of structural hypothesis: role symmetry, and locality of interaction. Our K-Roles algorithm, inspired by k-means clustering, combines supervised and unsupervised techniques to learn a role-symmetric game model. Our G3L iteratively learns a graphical game model.

We performed computational experiments on three relevant albeit stylized games, on instances with at least 100 agents. One game exhibits strict role structure, the second strict graph structure, and a third has structure but neither strictly. We found that both methods achieved good performance for structure learning in the models with clear structure, and both also demonstrated advantages of the iterative structure-learning approach to equilibrium seeking.

It is important to note that the identification of symmetry by K-roles assumes that we already have a common set of actions for the agents. In other situations we may have different action sets, which would just impose a constraint on the clustering process. In a more general version of the problem, identification of correspondences between actions of different agents would itself be something that would have to be learned; as yet we have not considered how to extend K-roles in that direction.

For graphical game learning, action correspondence is not a concern. A possible focus for improvement of G3L would be more explicit management of the tradeoffs in maximum neighborhood size $\hat{\kappa}$, considering simultaneously its affect on learning (i.e., use as a regularizer) and on game-theoretic computation with the resulting model.

Future work could encompass these issues as well as many other extensions to cover broader classes of games and additional forms of game structure.

Acknowledgment This work was supported in part by the US Army Research Office under MURI W911NF-18-1-0208.

References

Balduzzi, D.; Racaniere, S.; Martens, J.; Foerster, J.; Tuyls, K.; and Graepel, T. 2018. The mechanics of n-player differentiable games. In *ICML*.

Bazzan, A. L. C. 2009. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems* 18(3):342–375.

Blum, B.; Koller, D.; and Shelton, C. R. 2002. Game theory: Gametracer. http://dags.stanford.edu/Games/gametracer.html.

Blum, B.; Shelton, C. R.; and Koller, D. 2006. A continuation method for Nash equilibria in structured games. *Journal of Artificial Intelligence Research* 25:457–502.

Bramoullé, Y.; Kranton, R.; and D'Amours, M. 2014. Strategic interaction and networks. *American Economic Review* 104(3):898–930.

Daskalakis, C., and Papadimitriou, C. H. 2015. Approximate Nash equilibria in anonymous games. *Journal of Economic Theory* 156:207–245.

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

Duong, Q.; Vorobeychik, Y.; Singh, S.; and Wellman, M. P. 2009. Learning graphical game models. In *IJCAI*, 116–121.

Fearnley, J.; Gairing, M.; Goldberg, P. W.; and Savani, R. 2015. Learning equilibria of games via payoff queries. *Journal of Machine Learning Research* 16(1):1305–1344.

Ficici, S. G.; Parkes, D. C.; and Pfeffer, A. 2008. Learning and solving many-player games through a cluster-based representation. In *UAI*, 187–195.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2001. *The Elements of Statistical Learning*. Springer.

Gao, X. A., and Pfeffer, A. 2010. Learning game representations from data using rationality constraints. In *UAI*, 185–192.

Garg, V., and Jaakkola, T. 2016. Learning tree structured potential games. In *NeurIPS*.

Garg, V., and Jaakkola, T. 2017. Local aggregative games. In *NeurIPS*.

Goldberg, P. W., and Turchetta, S. 2017. Query complexity of approximate equilibria in anonymous games. *Journal of Computer and System Sciences* 90:80–98.

Govindan, S., and Wilson, R. 2003. A global Newton method to compute Nash equilibria. *Journal of Economic Theory* 110(1):65–86.

Govindan, S., and Wilson, R. 2004. Computing Nash equilibria by iterated polymatrix approximation. *Journal of Economic Dynamics and Control* 28(7):1229–1241.

Guo, X.; Hu, A.; Xu, R.; and Zhang, J. 2019. Learning mean-field games. In *NeurIPS*.

Heckerman, D.; Geiger, D.; and Chickering, D. M. 1995. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning* 20(3):197–243.

Herings, P. J.-J., and Peeters, R. 2010. Homotopy methods to compute equilibria in game theory. *Economic Theory* 42(1):119–156.

Honorio, J., and Ortiz, L. E. 2015. Learning the structure and parameters of large-population graphical games from behavioral data. *Journal of Machine Learning Research* 16(1):1157–1210.

Jackson, M. O. 2010. *Social and Economic Networks*. Princeton University Press.

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

Jordan, P. R.; Vorobeychik, Y.; and Wellman, M. P. 2008. Searching for approximate equilibria in empirical games. In *AAMAS*, 1063–1070.

Kearns, M.; Judd, S.; Tan, J.; and Wortman, J. 2009. Behavioral experiments on biased voting in networks. *Proceedings of the National Academy of Sciences* 106(5):1347–1352.

Kearns, M.; Littman, M. L.; and Singh, S. 2001. Graphical models for game theory. In *UAI*, 253–260.

Kuleshov, V., and Schrijvers, O. 2015. Inverse game theory: Learning utilities in succinct games. In *WINE*.

Lanctot, M.; Zambaldi, V.; Gruslys, A.; Lazaridou, A.; Tuyls, K.; Pérolat, J.; Silver, D.; and Graepel, T. 2017. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*.

Ling, C. K.; Fang, F.; and Kolter, J. Z. 2018. What game are we playing? End-to-end learning in normal and extensive form games. In *IJCAI*.

Ling, C. K.; Fang, F.; and Kolter, J. Z. 2019. Large scale learning of agent rationality in two-player zero-sum games. In *AAAI*.

McKelvey, R. D., and McLennan, A. 1996. Computation of equilibria in finite games. *Handbook of Computational Economics* 1:87–142.

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

Nevmyvaka, Y.; Feng, Y.; and Kearns, M. 2006. Reinforcement learning for optimized trade execution. In *ICML*, 673–680.

Ng, A. Y., and Russell, S. J. 2000. Algorithms for inverse reinforcement learning. In *ICML*.

Rand, W. M. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association* 66(336):846–850.

Sandholm, W. H. 2010. *Population Games and Evolutionary Dynamics*. MIT Press.

Settles, B. 2009. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Sokota, S.; Ho, C.; and Wiedenbeck, B. 2019. Learning deviation payoffs in simulation-based games. In *AAAI*, 2173–2180.

Srinivasan, S.; Lanctot, M.; Zambaldi, V.; Pérolat, J.; Tuyls, K.; Munos, R.; and Bowling, M. 2018. Actor-critic policy optimization in partially observable multiagent environments. In *NeurIPS*.

Stewart, G. 1998. On the adjugate matrix. *Linear Algebra and its Applications* 283(1-3):151–164.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press, second edition.

Tuyls, K.; Perolat, J.; Lanctot, M.; Leibo, J. Z.; and Graepel, T. 2018. A generalised method for empirical game theoretic analysis. In *AAMAS*, 77–85.

Vickrey, D., and Koller, D. 2002. Multi-agent algorithms for solving graphical games. In *AAAI*.

Viqueira, E. A.; Cousins, C.; Upfal, E.; and Greenwald, A. 2019. Learning equilibria of simulation-based games. *arXiv preprint arXiv:1905.13379*.

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

Walsh, W. E.; Parkes, D.; and Das, R. 2003. Choosing samples to compute heuristic-strategy Nash equilibrium. In *AMEC*.

Waugh, K.; Ziebart, B. D.; and Bagnell, J. A. 2011. Computational rationalization: The inverse equilibrium problem. In *ICML*, 1169–1176.

Wellman, M. P. 2016. Putting the agent in agent-based modeling. *Autonomous Agents and Multi-Agent Systems* 30:1175–1189.

Wiedenbeck, B.; Yang, F.; and Wellman, M. P. 2018. A regression approach for modeling games with many symmetric players. In *AAAI*, 1266–1273.

Yang, J.; Ye, X.; Trivedi, R.; Xu, H.; and Zha, H. 2018. Deep mean field games for learning optimal behavior policy of large populations. In *ICLR*.