Learning Graphical Game Models

Quang Duong[*] **Yevgeniy Vorobeychik**[†] **Satinder Singh**[*] **Michael P. Wellman**[*]

[*]Computer Science & Engineering
University of Michigan
Ann Arbor, MI 48109-2121 USA

[†]Computer & Information Science
University of Pennsylvania
Philadelphia, PA 19104 USA

Abstract

Graphical games provide compact representation of a multiagent interaction when agents' payoffs depend only on actions of agents in their local neighborhood. We formally describe the problem of learning a graphical game model from limited observation of the payoff function, define three performance metrics for evaluating learned games, and investigate several learning algorithms based on minimizing empirical loss. Our first algorithm is a branch-and-bound search, which takes advantage of the structure of the empirical loss function to derive upper and lower bounds on loss at every node of the search tree. We also examine a greedy heuristic and local search algorithms. Our experiments with directed graphical games show that (i) when only a small sample of profile payoffs is available, branch-and-bound significantly outperforms other methods, and has competitive running time, but (ii) when many profiles are observed, greedy is nearly optimal and considerably better than other methods, at a fraction of branch-and-bound's running time. The results are comparable for undirected graphical games and when payoffs are sampled with noise.

1 Introduction

Game theory is a formal framework for reasoning about outcomes of strategic interactions among self-interested players. This framework requires a complete specification of the players' *payoff functions* (*payoff matrices* when sets of actions are finite), which model the dependence of player utilities on the actions of the entire agent pool. To exploit common situations where interactions are localized, Kearns et al. [2001] introduced *graphical games*, which capture dependence patterns as a graph structure. Graphical representations of games can induce substantial compression of the game representation, and significant speedup in computing or approximating game-theoretic equilibria or other solution concepts [Duong et al., 2008].

We consider situations where there is some underlying game which can be represented compactly by a graphical model, but the payoff functions of players are unknown. We are given a data set of payoff realizations for specific strategy profiles, which are drawn according to some fixed probability distribution. For example, the game may be defined by an underlying simulator, from which we have a limited budget of payoff observations. The goal is to learn a graphical representation of the game based on this payoff experience.

Learning graphical structures has been explored in many other contexts, notably that of inducing Bayesian network structure from data. The general version of this problem is NP-hard, and so heuristics are commonly employed, particularly variants of local search [Heckerman et al., 1995].[1] Inducing a compact representation of a joint distribution over random variables is generally formulated as an unsupervised learning problem, since we are given data reflecting the distribution rather than a label specifying the distribution itself. In contrast, in our setting we induce a compact representation of a payoff function given a sample of action profile and payoff tuples. Thus, like previous work on learning payoff functions given similar input [Vorobeychik et al., 2007], our problem falls in the supervised learning paradigm. A similar direction was pursued by Ficici et al. [2008], who learn a cluster-based representation of games that takes advantage of game-theoretic symmetries wherever these exist.

We formally define the problem of learning graphical games and introduce several techniques for solving it. One technique, a branch-and-bound search, computes an optimal solution to the empirical loss minimization problem. Other techniques are local heuristics, ranging from greedy to several variants of simulated annealing. We find that branch-and-bound can effectively take advantage of the problem structure and runs very fast (competitive with the heuristic algorithms) when the data set is not too large. On the other hand, with a large data set the greedy heuristic performs nearly as well as branch-and-bound at a fraction of running time.

We provide an overview of game theory and graphical games in Section 2. Section 3 gives a formal description of the graphical game learning problem and presents algorithms for tackling it. In Section 4 we introduce three evaluation metrics for learned graphical games and present our experimental results.

[1] Schmidt et al. [2007] present an alternative heuristic method based on L_1-regularized regression.

2 Preliminaries

2.1 Game Theory

A game in normal form is given by $[I, \{A_i\}, \{u_i(a)\}]$, where I is the set of players with $|I| = N$, A_i is the set of *pure strategies* (actions) of player $i \in I$. We use $A = A_1 \times \cdots \times A_N$ to denote the set of pure joint strategies, or *profiles*. The payoff function $u_i(a)$ represents the value to player i of the outcome when strategy profile $a \in A$ is played. In this work we focus on *finite* games, in which the set of players I, as well as the pure strategy sets A_i, $i \in I$ are finite. We use a_{-i} to denote the joint strategic choice of all players other than i.

In a normal-form game, players make one-shot decisions (in effect) simultaneously and accrue payoffs, upon which the game ends. In this setting, an agent would ideally play its best strategy given a fixed joint strategy of everyone else. A configuration in which all strategies are mutual best responses constitutes a *Nash equilibrium*.

Definition 1. *Strategy profile s is a (pure strategy)* Nash equilibrium *of the game $[I, \{A_i\}, \{u_i(a)\}]$ if for every player $i \in I$ and any available strategy $a'_i \in A_i$, $u_i(a) \geq u_i(a'_i, a_{-i})$.*

A useful related notion is *regret*, denoted $\epsilon(a)$, which evaluates stability of a profile to deviations:

$$\epsilon(a) = \max_{i \in I} \max_{a'_i \in A_i} u_i(a'_i, a_{-i}) - u_i(a). \tag{1}$$

Note that a is a Nash equilibrium iff $\epsilon(a) = 0$.

2.2 Graphical Games

Whereas the normal form is a rather general representation, it fails to capture any structure that may be present in the actual game. One common form of structure that has a long history in its own right is a network of relationships among the agents in the game. Specifically, it may be that an agent does not affect all the others' payoffs, but rather each agent is strategic only vis-a-vis a relatively small set of *neighbors*.

To capture this notion, Kearns et al. [2001] introduced a graphical representation, where a link (directed or undirected) between two players represents a payoff dependence between them. Formally, a graphical game model is a tuple $[I, \{A_i\}, \{J_i\}, \{u_i(\cdot)\}]$, where I and A_i are as before, J_i is a collection of players connected to i, and $u_i(a_i, a_{J_i})$ is the payoff to player i playing $a_i \in A_i$ when the players J_i jointly play a_{J_i}. The notation $a_{J_i} \subset a_{-i}$ means that each $j \in J_i$ plays its assigned strategy from a_{-i}. We define A_{J_i} to be the set of joint strategies of players in i's neighborhood. This game structure is captured by a graph $G = [V, E]$ in which $V = I$ (i.e., each node corresponds to a player) and there is an edge $e = (j, i) \in E$ iff $j \in J_i$. In this presentation we focus on directed graphical models (noting extensions and comparisons to the undirected case on occasion), where payoff dependence between players need not be mutual.

3 Learning Graphical Games

3.1 Problem Definition

Suppose that we are given a partial specification of a game in normal form, $[I, A]$, that is, players and action sets but not payoff functions. We refer to such a specification as a *game form*. Additionally, we are given a data set $D = \{(a^1, U^1), \ldots, (a^n, U^n)\}$ of profiles and corresponding payoffs (or noisy samples from the true payoff function).

Our goal is to induce the graphical game structure. Specifically, we seek an algorithm that takes as input data set D and game form $[I, A]$, and outputs a graphical game $\hat{G} = [I, \{A_i\}, \{\hat{J}_i\}, \{\hat{u}_i(\cdot)\}]$ with $\hat{J}_i$ the estimated collection of neighbors of player i and $\hat{u}_i(\cdot)$, i's estimated utility function.

One of the basic ways to design and evaluate learning algorithms is by defining a *loss function*. Given true payoffs u, we define the *true* (quadratic) loss of a learned graphical game $\hat{G}$ for player i to be

$$L_i(\hat{J}_i, \hat{u}_i) = E_{a \sim P}[(u_i(a) - \hat{u}_i(a_i, a_{\hat{J}_i}))^2], \tag{2}$$

where P is some distribution over profiles[2], and $a_{\hat{J}_i}$ is the vector of actions played by the players in $\hat{J}_i$. Whereas the true loss is a measure of learning performance, learning algorithms are typically designed to minimize *empirical* loss. Define the empirical loss for player i, given data set D, to be

$$\hat{L}_i(\hat{J}_i, \hat{u}_i) = \frac{1}{n} \sum_{d=1}^{n} (U_i^d - \hat{u}_i(a_i^d, a_{\hat{J}_i}^d))^2. \tag{3}$$

Given the definition of loss for a particular player, we define the total true loss of a learned graphical game $\hat{G}$ by $L(\{\hat{J}_i\}, \hat{u}) = \frac{1}{N} \sum_i L_i(\hat{J}_i, \hat{u}_i)$. The total empirical loss is defined analogously.

The loss functions above are defined given the utility estimates $\hat{u}_i(\cdot)$. A natural estimator of $\hat{u}_i$ at $(a_i, a_{\hat{J}_i})$ is a sample mean over all profiles in D consistent with that partial profile:

$$\hat{u}_i(a_i, a_{\hat{J}_i}) = \frac{1}{|D_{(a_i, a_{\hat{J}_i})}|} \sum_{(a^l, U^l) \in D_{(a_i, a_{\hat{J}_i})}} U^l, \tag{4}$$

where $D_{(a_i, a_{\hat{J}_i})}$ denotes the set $\{(a^l, U^l) \in D \mid a_i = a_i^l, a_{\hat{J}_i} \subset a_{-i}^l\}$.[3] To see that this is a sensible utility estimator, we note in the following lemma[4] that it has a natural invariance property with respect to edges not in the underlying graph.

Lemma 1. *Suppose that D contains all profiles $a \in A$, and payoffs are obtained with no noise. For any i, $\hat{J}_i$, and $j \notin J_i$, $\hat{u}_i(a_i, a_{\hat{J}_i}) = \hat{u}_i(a_i, a_{\hat{J}'_i})$ for all $a \in A$, where $\hat{J}'_i = \hat{J}_i \cup j$.*

Henceforth, we assume that $\hat{u}$ is obtained using (4) and use the shorthand notation $\hat{L}_i(\hat{J}_i)$, or simply $\hat{L}_i$.

Definition 2. *The empirical loss function $\hat{L}_i$ is* monotone *if $\hat{J}_i \subseteq \hat{J}'_i$ implies that $\hat{L}_i(\hat{J}_i) \geq \hat{L}_i(\hat{J}'_i)$.*

We establish that adding one player to the neighborhood of i (i.e., adding one edge to the graph) reduces empirical loss.

Theorem 2. *Let $\hat{L}_i$ and $\hat{u}_i$ be as defined in* (3) *and* (4) *respectively. Then for any $\hat{J}_i$, j, $\hat{L}_i(\hat{J}_i) \geq \hat{L}_i(\hat{J}_i \cup j)$.*

[2] In the evaluation section we assume a uniform distribution.

[3] Observe that this estimator is undefined when $|D_{(a_i, a_{\hat{J}_i})}| = 0$. We address this issue in Section 4.2 and assume it away for now.

[4] Proofs of all results are provided in the extended version.

Corollary 3. *The empirical loss function is monotone.*

Note that neither this monotonicity nor Lemma 1 need hold under alternate definitions of approximate payoffs.

3.2 Constrained Loss Minimization

Given monotonicity, minimizing empirical loss is trivial unless we have some constraint or cost on graph complexity. To control complexity, we impose degree bounds M_i on the number of neighbors that a player i can have. Our problem (for the directed case) becomes

$$\min_{\{\hat{J}_i\}_{i\in I}} \sum_i \hat{L}_i(\hat{J}_i) \quad \text{s.t.} \quad |\hat{J}_i| \leq M_i \; \forall \, i \in I.$$

Alternative complexity constraints, such as on the total number of edges, tree-width, or other measure, could be incorporated in a similar manner (with algorithmic implications). Learning undirected graphs entails including an additional symmetry constraint, $j \in \hat{J}_i$ iff $i \in \hat{J}_j$. In the directed case, since the objective is additive in i and the neighbor constraints are independent, we can decompose the problem into N separate minimization problems.

3.3 Learning Algorithms

We present four algorithms for learning the graphical structure and payoff function of a game from data. The descriptions below cover directed graphical games; extension to the undirected case is conceptually straightforward. The first algorithm is a complete branch-and-bound search in graph space, and the remaining three are heuristic methods: greedy edge selection and two versions of stochastic local search. Since we focus on the decomposable problem of learning directed graphical games with degree constraints, we describe each algorithm in terms of learning the neighborhood structure for a fixed player i.

Branch-and-Bound Search

The branch-and-bound search algorithm, BB, constructs a search tree in which each node corresponds to a partial specification of $\hat{J}_i$, the neighborhood of i. Formally, each node at depth h is associated with a bit vector, $b = \langle b_1, \ldots, b_h \rangle$, with $b_j = 1$ representing an assignment of j to the neighborhood, $j \in \hat{J}_i$. The root (depth 0) of the search tree represents the null assignment. The two children of the root node expand the vector to one element, b_1, the left child assigning $b_1 = 0$ and the right $b_1 = 1$. Similarly, at depth h, the left child expands the assignment with $b_h = 0$ and the right with $b_h = 1$. The leaves (at depth $N-1$) of the search tree represent the 2^{N-1} possible vectors that fully specify player i's neighborhood.

The key to a branch-and-bound search is to provide upper and lower bounds on the loss function at every node. Let $\langle b_1, \ldots, b_h \rangle$ be a partial assignment at depth h. Monotonicity allows us to establish an upper bound by setting all $b_{h'}$, $h' > h$, to 0. We improve this bound by sequentially adding as many edges as the problem's constraints allow. To obtain a simple lower bound, we can again take advantage of the monotonicity of empirical loss by setting $b_{h'}$, $h' > h$, to 1. We can improve on this by subsequently removing a single edge (j^*, i) with smallest $\Delta_{\hat{J}_i, j^*} \hat{L}_i = \hat{L}_i(\hat{J}_i) - \hat{L}_i(\hat{J}_i \cup j^*)$, that is, with smallest impact on empirical loss.

Lemma 4. *Let $\hat{l}$ be the empirical loss of the graphical game induced by $\langle b_1, \ldots, b_h, 1, \ldots, 0, \ldots, 1 \rangle$, where 0 takes the place of the b_{j^*} with minimal $\Delta_{\hat{J}_i, j^*} \hat{L}_i$. $\hat{l}$ is then a lower bound on the loss of any leaf of $\langle b_1, \ldots, b_h \rangle$ if $\sum_{k=1}^{h} b_k + N - 1 - h \geq M_i$.*

A subtree at a node is pruned if the lower bound on loss is strictly greater than the upper bound at *some* other node. In addition to pruning the tree based on upper and lower bounds, we can also naturally prune subtrees due to the complexity constraints: for example, once $|\hat{J}_i| = M_i$, the remaining unassigned bits must be zero. Furthermore, by monotonicity, we can prune a subtree at level h if $\sum_{k=1}^{h} b_k + N - 1 - h < M_i$ (i.e., even adding every remaining edge would not reach the neighborhood constraint). We can implement the search as the standard breadth- or depth-first search, both of which would be complete with our branch-and-bound, since there is only finite depth. Branch-and-bound together with either breadth-first search or depth-first search is complete.

Greedy Loss Minimization

The greedy algorithm selects among potential edges (j, i) based on the amount of loss reduction due to introducing the edge, given the edges added so far. That is, given a current configuration $\hat{J}_i$, the algorithm adds an edge from j to i that maximizes $\Delta_{\hat{J}_i, j} \hat{L}_i$. We add one edge at a time until $|\hat{J}_i| = M_i$.

Local Search

We explore two versions of local search for loss-minimizing neighborhoods. Let $b_t = \langle b_{1,t}, \ldots, b_{N-1,t} \rangle$ be a vector in iteration t indicating which nodes are included in the neighborhood of i and suppose w.l.o.g. that $\sum_{i=1}^{N-1} b_{i,t} \leq M_i$ (otherwise edges can be removed arbitrarily to satisfy the constraint). Let $\hat{J}_{i,t}$ be the corresponding estimated set of neighbors of i and $\hat{L}_{i,t} = \hat{L}_i(\hat{J}_{i,t})$ its corresponding empirical loss.

The first search method ($SA\text{-}Simple$) uses a standard simulated annealing procedure to obtain the next iterate b_{t+1}. It generates a candidate b' by uniformly selecting among the vectors differing from b_t by one bit, and satisfying the degree constraint. The next iterate b_{t+1} is then chosen to be b' with probability $p_t(\hat{L}_{i,t}, \hat{L}'_i)$, and b_t otherwise, with

$$p_t(\hat{L}_{i,t}, \hat{L}'_i) = \begin{cases} \exp\left[-\frac{\hat{L}'_i - \hat{L}_{i,t}}{T_k}\right] & \text{if } \hat{L}'_i > \hat{L}_{i,t} \\ 1 & \text{otherwise,} \end{cases}$$

where T_k is a schedule of "temperatures" that govern the probability with which inferior candidates are explored.

The second local search method ($SA\text{-}Advanced$) is also based on simulated annealing. However, instead of randomly selecting a single candidate b', it examines all (feasible) neighbors of b_t and associates each neighbor $b_{j,t}$ with a probability measure:

$$p_{j,t}(\hat{L}_{i,t}, \hat{L}_{j,t}) \propto \begin{cases} \exp\left[(\hat{L}_{i,t} - \hat{L}_{j,t})T_0\right] & \text{if } \hat{L}_{j,t} \leq \hat{L}_{i,t} \\ \exp\left[\frac{\hat{L}_{i,t} - \hat{L}_{j,t}}{T_k}\right] & \text{if } \hat{L}_{j,t} > \hat{L}_{i,t} \;\&\; \\ & \nexists c : \hat{L}_{c,t} \leq \hat{L}_{i,t} \\ 0 & \text{otherwise,} \end{cases}$$

where T_0 is the initial value of the declining temperature schedule T_k. Each candidate $b_{j,t}$ is chosen as the next iterate b_{t+1} with probability $p_{j,t}$. Note that as T_0 goes to infinity, this version of local search approaches the behavior of the greedy algorithm. However, the ability to bias the choice of next configuration in favor of greater loss-reducing candidates comes at the increased cost of computing empirical loss for all candidates differing on one edge.

4 Evaluation

We evaluate our graphical-game learning methods with computational experiments on randomly generated games. We begin with the case where the data set includes all profiles $a \in A$, and each player's utility is sampled with no noise. We then evaluate the algorithms in more interesting and realistic instances where only a subset of profiles have been sampled and provided as training data or where payoff observations are noisy.

Our experiments are mainly concerned with directed graphical games, though we also present some results for small undirected games. Note that learning the graphical representation may be useful even when data are available to completely describe the game. For example a graphical model can considerably speed up equilibrium computation or other game-theoretic analysis. We may also wish to understand the graphical structure for its own sake, for example, in analysis of social network structure. These goals are complementary to our underlying problem of learning the graphical model by minimizing measurable loss.

4.1 Performance Metrics

We evaluate our learning methods using metrics that correspond to the three central goals of learning graphical games.

Approximating True Loss The first and most natural metric is the one we used to set up the problem: approximating the true loss, as defined in (2). This is a typical goal of machine learning in any domain, although perhaps the least important in our case. Rather, we use a standard learning setup as a means to an end, where the end in our case is, perhaps, better captured by the two metrics that follow.

Approximating Graphical Structure Our second metric focuses exclusively on the graphical structure of the learned games. Given the graphs of the underlying game $G = (V, E)$, and the learned game $\hat{G} = (V, \hat{E})$, we define their *structural similarity* as:

$$\frac{|E \cap \hat{E}|}{|E|}.$$

Structural similarity measures how much the learned graph $\hat{G}$ resembles the underlying graph G. When all profiles $a \in A$ are included in the input data set and the payoff for each is exact, the proposed greedy heuristic is in fact optimal according to this metric for learning directed graphs. To see this, note that a direct consequence of Lemma 1 is that adding an edge (j, i) which is not in the actual graph never decreases empirical loss. As a result, the greedy heuristic will only select edges which are in the underlying graph, until these are exhausted (i.e., until there is no other edge that decreases empirical loss). Thus, $\hat{E} \subset E$ and, hence, $E \cap \hat{E} = \hat{E}$. When we learn graphical games, $|\hat{E}| = \sum_i \min(M_i, |E_i|)$ both for the greedy heuristic and for any optimal algorithm, where $E_i = \{(j, i) \mid j \in J_i\}$. Given the graphical learning literature's context, this result is surprising even in the limited sense in which it holds.

In the undirected case, this argument no longer applies even if the conditions of Lemma 1 are satisfied.

Approximating Nash Equilibria We adopt the notion of regret defined in (1) as the basis for our third performance metric. Recall that for a strategy profile a, regret is the maximum gain any player would achieve by deviating from action a_i. Let G be the actual game. Suppose that $\hat{Q}$ is a set of pure-strategy Nash equilibria of the learned graphical game $\hat{G}$. In cases where there exist no exact pure equilibria, $\hat{Q}$ would comprise the profiles with smallest regret. We define the regret-based metric to be the average regret of profiles in $\hat{Q}$ with respect to G:

$$\frac{\sum_{a \in \hat{Q}} \epsilon(a)}{|\hat{Q}|}.$$

4.2 Empirical Results

We generated game instances for our experiments using the GAMUT game generation engine [Nudelman et al., 2004]. Specifically, we investigated three graphical structures: 10-player random graphs with half of the edges of a fully connected graph, 13-player tree games with maximum degree 4, and 10-player star games. We sampled 25 random game instances of each type of graphical game, with payoffs sampled uniformly randomly on $[0, 1]$. We considered games where players had two actions each.

We evaluated our algorithms in three settings. In the first, we observe exact payoffs for every strategy profile in the game, and we are merely interested in learning a compact representation. The second setting is the same, except that payoffs are sampled with additive zero-mean normal noise. In the final setting only a small subset of strategy profiles is available for learning. As the results for star games are similar to those for tree games, we omit these for lack of space.

Complete Game Data

We first consider a setting in which the data set includes exact payoff realization for every $a \in A$. The results for learning a directed graphical model on such data are presented in Figure 1. As we can readily observe, the greedy algorithm (GR) is nearly optimal and considerably better than other heuristics, a result which is robust across the metrics and game classes we considered. Furthermore, Figure 5 shows that GR has a substantially lower running time than BB, and is faster than $SA\text{-}Advanced$, the most competitive heuristic. Above we had already noted that in this setting GR is in fact optimal vis-a-vis the structural similarity metric, and so it comes as little surprise that it is also quite good on other metrics. We note that since GR is actually a special case of

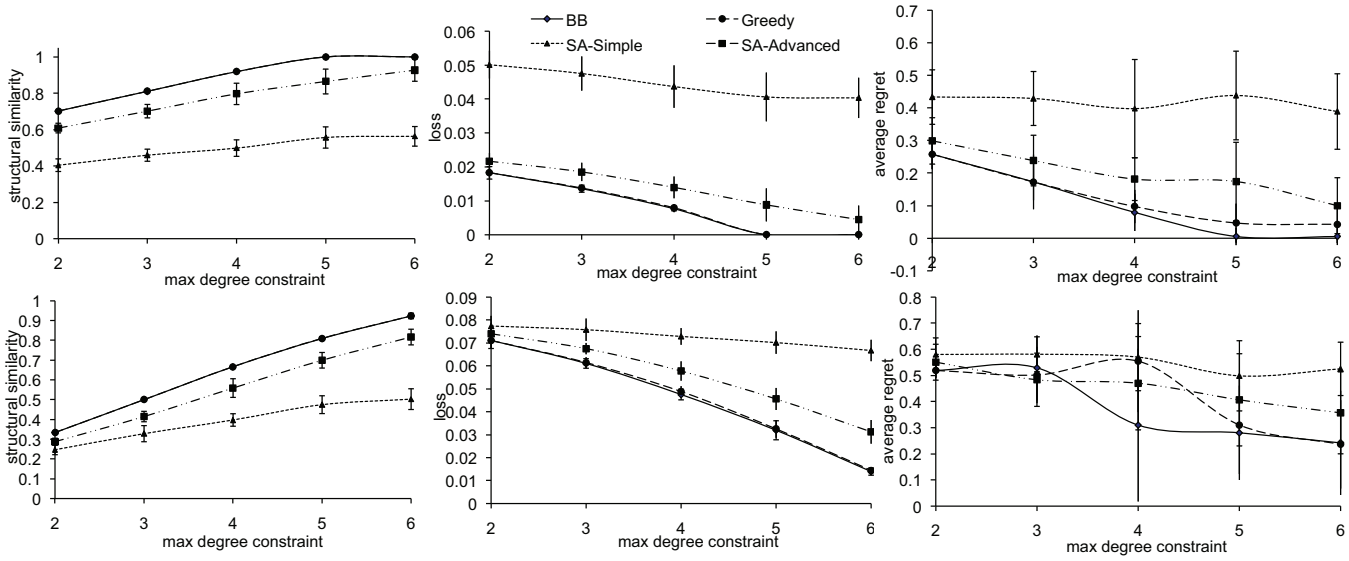

Figure 1: Directed-tree (top) and random-graph (bottom) games learned from complete game data.

$SA\text{-}Advanced$, the parameters of $SA\text{-}Advanced$ were clearly set suboptimally. However, the point is that $SA\text{-}Advanced$ is a more complex algorithm that requires a significant amount of tuning for its several parameters, and here the added complexity does not pay off. Nevertheless, $SA\text{-}Advanced$ does outperform the simpler and faster $SA\text{-}Simple$.

We next consider undirected graphical games with five players.[5] We observe that GR is now suboptimal in the structural similarity metric. However, across all three metrics, GR is still nearly as good as BB, and outperforms the other approximate approaches, although its advantage is somewhat smaller here. A comparison of the methods on the loss metric is shown in Figure 2.

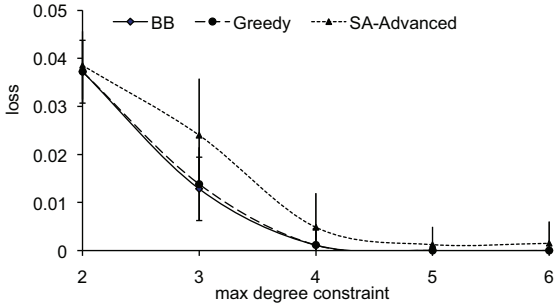

Figure 2: Undirected random-graph games learned from complete game data.

Finally, we consider tree games in which we generated payoffs with additive noise distributed according to $N(0, 1)$. In Figure 3 we plot the results for only the loss metric; the results for other metrics are qualitatively similar. We can see that the relative ranking of the methods is unchanged: GR is still nearly as good as BB, whereas $SA\text{-}Advanced$ tends to be significantly worse. We believe that a part of the reason that GR appears so robust to noise is that the actual payoffs are uniformly randomly distributed, so $N(0, 1)$ noise does not significantly distort observations on average.

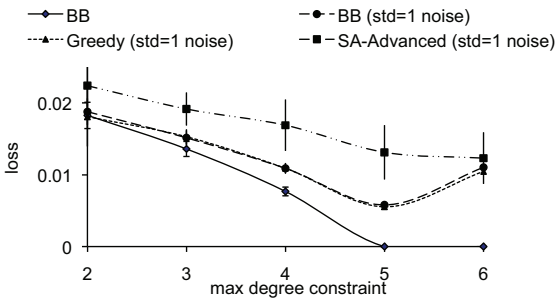

Figure 3: Loss when the entire game is sampled and added noise for tree graph cases

Partial Game Data

We now evaluate the performance of our learning algorithms when only a small number of strategy profiles have been sampled, assuming that the exact payoff vector is obtained for each. Another problem arising in this context is how to estimate payoffs for profiles in the learned graphical model for which no data is available. We sidestep this issue here by using an average over the payoffs for all observed profiles.[6] In Figure 4, we present performance results for our learning methods when only 50 profiles are observed. When the size of the data set is much smaller than game size, BB performs substantially better than GR (this is true for all three metrics; we show only the results for actual loss). Additionally, the ad-

[5]The reduction in number of players was necessary to accommodate the increased expense of branch-and-bound search.

[6]We tried other techniques also, but the results seem to vary little.

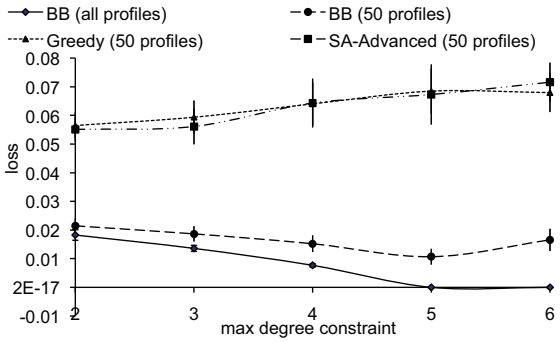

Figure 4: Loss when 50 profiles (out of 1024) are sampled in tree graphical games.

vantage of GR over $SA\text{-}Advanced$ dissipates. As we can observe from Figure 5, there is no longer a significant speedup offered by GR over BB. Since Lemma 1 does not hold for the case of partial game data, both GR and BB may now include edges not present in the actual graph. Thus, the order in which GR adds edges becomes especially significant here, which, together with its policy of permanently adding edges, yields empirical loss well above optimal and above BB's.

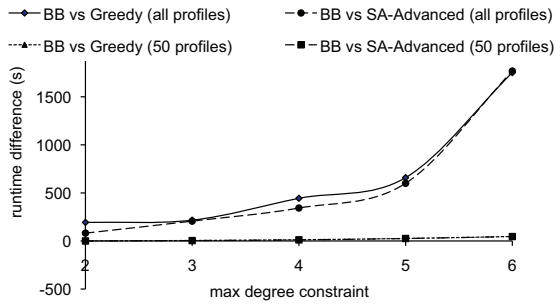

Figure 5: Runtime differences between branch-and-bound and approximate algorithms to learn tree graphical games

5 Conclusion

We have formally introduced the problem of learning graphical games from payoff data, and investigated a family of learning methods. We define a loss function over candidate game structures, based on a default estimator for local payoff functions given the learned neighborhood structure. The learning methods we investigate search for graphical game structures that minimize empirical loss. Given the varied goals of graphical game learning, we define evaluation metrics corresponding to each of three possible aims: (i) learning the actual graphical structure (e.g., social network) among the players, (ii) computing game-theoretic solutions, and (iii) learning the payoff functions to better understand structural properties of the game (perhaps with the purpose of developing better and simpler stylized models). We proceed to empirically evaluate four proposed learning methods: branch-and-bound, greedy, and two versions of simulated annealing. Our findings suggest that when a small subset of the profiles in the game is available, one should employ branch-and-bound (BB), as it considerably outperforms other methods we tried and has competitive running time.[7] On the other hand, as the size of the data set becomes large, the advantage of BB over the greedy algorithm dissipates while its computation time increases considerably, particularly in learning undirected graphs. Our results are similar when noise is added to payoff realizations. In future work, we intend to explore active learning techniques, and apply our algorithms to learning the graphical structure of games in specific domains for which we have simulation models.

References

Q. Duong, M. P. Wellman, and S. Singh. Knowledge combination in graphical multiagent models. In *Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 153–160, 2008.

S. G. Ficici, D. C. Parkes, and A. Pfeffer. Learning and solving many-player games through a cluster-based representation. In *Twenth-Fourth Conference on Uncertainty in Artificial Intelligence*, pages 187–195, 2008.

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

M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 880–887, 2004.

M. Schmidt, A. Niculescu-Mizil, and K. Murphy. Learning graphical model structure using L1-regularization paths. In *Twenty-Second National Conference on Artificial Intelligence*, pages 1278–1283, 2007.

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

[7] We observe that pruning generates better than a factor of two reduction in BB running time.