A Continuation Method for Nash Equilibria in Structured Games

Ben Blum
Stanford University
bblum@stanford.edu

Christian R. Shelton
Stanford University
cshelton@cs.stanford.edu

Daphne Koller
Stanford University
koller@cs.stanford.edu

Abstract

We describe algorithms for computing Nash equilibria in structured game representations, including both graphical games and multi-agent influence diagrams (MAIDs). The algorithms are derived from a continuation method for normal-form and extensive-form games due to Govindan and Wilson; they follow a trajectory through the space of perturbed games and their equilibria. Our algorithms exploit game structure through fast computation of the Jacobian of the game's payoff function. They are guaranteed to find at least one equilibrium of the game and may find more. Our approach provides the first exact algorithm for computing an exact equilibrium in graphical games with arbitrary topology, and the first algorithm to exploit fine-grain structural properties of MAIDs. We present experimental results for our algorithms. The running time for our graphical game algorithm is similar to, and often better than, the running time of previous approximate algorithms. Our algorithm for MAIDs can effectively solve games that are much larger than those that could be solved using previous methods.

1 Introduction

Game theory is a mathematical framework that describes interactions between multiple rational agents and allows for reasoning about their outcomes. However, the complexity of standard game descriptions grows exponentially with the number of agents involved. For many multi-agent situations, this blowup presents a serious problem. Recent work in artificial intelligence [La Mura, 2000; Kearns *et al.*, 2001; Koller and Milch, 2001] proposes the use of structured game representations that utilize a notion of locality of interaction; these representations allow a wide range of complex games to be represented compactly.

In this paper we consider the task of computing *Nash equilibria* for structured games. A Nash equilibrium is a strategy profile such that it is no agent's interest to deviate unilaterally. A naive approach to finding Nash equilibria is to convert the structured game into a standard game representation, and apply a standard game-theoretic solution algorithm [McKelvey and McLennan, 1996]. This approach is, in general, infeasible for all but the simplest games. We would like an algorithm that exploits the structure in these game representations for efficient computation.

In this paper, we describe a set of algorithms that use *continuation methods* for solving structured games. These algorithms follow a trajectory of equilibria of perturbed games until an equilibrium of the original game is found. Our algorithms are based on the recent work of Govindan and Wilson [2002; 2003a; 2003b] (GW hereafter), which apply to standard game representations (normal-form and extensive-form). We show how the structure of the games can be exploited to perform the key computational step of the algorithms of GW.

Our methods address both *graphical games* [Kearns *et al.*, 2001] and *multi-agent influence diagrams (MAIDs)* [Koller and Milch, 2001]. We present the first algorithm for finding exact equilibria in graphical games of arbitrary structure. We also provide the first algorithm that can take advantage of the fine-grained structure of MAIDs, and therefore can handle MAIDs which are significantly outside the scope of previous methods. We provide experimental results demonstrating the efficiency of our approach relative to previous methods.

2 Game Representations and Equilibria

2.1 Game Theory

We begin by briefly reviewing concepts from game theory used in this paper, referring to Owen [1995] for a good introduction. A game defines an interaction between a set N of agents. Each agent $n \in N$ has a set of available *strategies* Σ_n, where a strategy determines the agent's behavior in the game. The precise definition of the set Σ_n depends on the game representation, as we discuss below. A *strategy profile* σ defines a strategy $\sigma_n \in \Sigma_n$ for each $n \in N$. Given a strategy profile σ, the game defines a payoff $G_n(\sigma)$ for each agent $n \in N$. We use σ_{-n} to denote the strategy profiles of the agents $n' \in N - \{n\}$. Similarly, we use Σ_{-n} to refer to the set of all strategy profiles of agents in $N - \{n\}$.

A solution to a game is a prescription of a strategy profile for the agents. The agent's goal is to maximize its payoff. Thus, a basic desideratum for a solution profile is stability — it should not be in any agent's interests to deviate from it. More precisely, the fundamental notion of a *Nash equilibrium* [Nash, 1951] is defined to be a strategy profile σ such that, for all $n \in N$, $G_n(\sigma_n, \sigma_{-n}) \geq G_n(\sigma'_n, \sigma_{-n})$, for all other strategies $\sigma'_n \in \Sigma_n$. Thus, if an agent knew that the others were playing according to an equilibrium profile, it would have no incentive to deviate.

An ϵ-equilibrium is a strategy profile such that no agent can improve its expected payoff by more than ϵ by unilaterally changing its strategy. Unfortunately, finding an ϵ-equilibrium is not necessarily a step toward finding an exact equilibrium: the fact that σ is an ϵ-equilibrium does not guarantee the existence of an exact equilibrium in the neighborhood of σ.

Normal-Form Games

A general-sum normal-form game defines a simultaneous-move multiagent scenario in which each agent independently selects an action and then receives a payoff that depends on the actions selected by all of the agents. More precisely, let G be a normal-form game with a set N of agents. Each agent $n \in N$ has a discrete action set S_n and a payoff array G_n with entries for every action profile in $\prod_{n \in N} S_n$.

Equilibrium strategies often require that agents randomize their choice of action. A *mixed strategy* σ_n is a probability distribution over S_n. The set Σ_n is the set of all mixed strategies. The *support* of a mixed strategy is the set of actions in S_n that have non-zero probability. A strategy σ_n for agent n is said to be a *pure strategy* if it has only a single action in its support. The set Σ of mixed strategy profiles is $\prod_{n \in N} \Sigma_n$. A mixed strategy profile $\sigma \in \Sigma$ is thus an m-vector, where $m = \sum_{n \in N} |S_n|$. Every game is guaranteed to have at least one mixed-strategy equilibrium, and the number of equilibria may be exponential in the number of agents.

Extensive-Form Games

An extensive-form game is represented by a tree in which each node represents a choice either of an agent or of nature. Each of nature's choice nodes is associated with a probability distribution over its outgoing branches. Each leaf $z \in Z$ of the tree is associated with a vector of payoffs $G(z)$, where $G_n(z)$ denotes the payoff to agent n at leaf z. The choices of the agents and nature dictate which path of the tree is followed and therefore the payoffs to the agents.

The decision nodes belonging to each agent are partitioned into information sets, where each information set is a set of states among which the agent cannot distinguish. Thus, an agent's strategy must take the same action at all nodes in the same information set. We define an *agent history* $H_n(y)$ for a node y in the tree and an agent n to be a sequence containing the information sets traversed in the path from the root to y, and the action selected at each one. Thus, two nodes have the same agent-n history if the paths used to reach them are indistinguishable to n. (The paths may differ in other ways, such as nature's decisions or the decisions of other agents.) We make the common assumption of *perfect recall*: an agent does not forget information known nor the choices made at previous decisions. More precisely, if two nodes y, y' are in the same information set for agent n, then $H_n(y) = H_n(y')$.

We need a representation of a strategy for an extensive-form game; unlike the case of normal-form games, there are several quite different choices. For our purposes, the most appropriate representation is the *sequence form* [Koller and Megiddo, 1992; von Stengel, 1996]. Here, the strategy σ_n for an agent n is represented as a vector of real values of size h_n, one for each distinct history $H_n(z)$ for a leaf z in the tree. The number $\sigma_n(H_n(z))$, abbreviated $\sigma_n(z)$, is the product of the probabilities controlled by agent n along the history H_n. From this representation, we can easily derive the probability of taking an action at a particular information set.

The set of sequence form strategies for agent n is therefore a subset of $\mathbb{R}^{h_n}$, where h_n is at most the number of leaves in the tree. The set of legal sequence form strategies Σ_n for agent n is defined by a set of linear constraints on vectors in $\mathbb{R}^{h_n}$. The set of sequence form strategy profiles is then defined as $\Sigma = \prod_{n \in N} \Sigma_n$. The payoff to agent n in an extensive-form game can be shown to be

$$G_n(\sigma) = \sum_{z \in Z} G_n(z) \prod_{i \in N} \sigma_i(z) \ . \tag{1}$$

Thus, the payoffs are a sum, over the leaves in the tree, of the payoff at a leaf times the product of the sequence form parameters for that leaf.[1] Importantly, this expression has a similar multi-linear form to the payoff in a normal-form game, but using sequence form strategies rather than mixed strategies. In an extensive form game satisfying perfect recall, any mixed strategy equilibrium can be represented using an essentially equivalent sequence form strategy profile.

2.2 Structured Representations

Graphical Games

The size of the payoff arrays required to describe a normal-form game grows exponentially with the number of agents. Kearns *et al.* [2001] introduced the framework of *graphical games*, which provide a more structured representation based on probabilistic graphical models. Graphical games capture local structure in multi-agent interactions, allowing a compact representation for scenarios where each agent's payoff is only affected by a small subset of other agents. Examples of interactions where this structure occurs include agents that interact along organization hierarchies and agents that interact according to geographic proximity.

A graphical game is described like a normal-form game. The basic representation (slightly generalized) is a directed graph with one node for each agent. An edge from agent n' to agent n in the graph indicates that agent n's payoffs depend on the action of agent n'. More precisely, we define Fam_n to be the set of agents consisting of n itself and its parents in the graph. The agent's payoff G_n is an array indexed only by the actions of the agents in Fam_n. Thus, the description of the game is exponential in the in-degree of the graph and not in the total number of agents. In this case, we use Σ_{-n} and S_{-n} to refer to strategy profiles of the agent in $Fam_n - \{n\}$.

Multi-agent Influence Diagrams

The description length of extensive-form games often also grows exponentially with the number of agents. In many situations this large tree can be represented more compactly. *Multi-agent influence diagrams* (MAIDs) [Koller and Milch, 2001] allow a structured representation of games involving time and information by extending *influence diagrams* [Howard and Matheson, 1984] to the multi-agent case.

A MAID is represented as a directed acyclic graph over nodes of three types: chance, decision, and utility. (Utility

[1] For notational simplicity, $G_n(z)$ includes nature's probabilities.

nodes are assumed to have no children.) Chance nodes represent nature's actions, and a *conditional probability distribution (CPD)* is associated with each such node, describing the distribution over outcomes of that variable conditioned on the values of its parents. Each decision node is associated with a single agent. The parents of a decision node represent the variables whose values are known to the agent when making that decision. Thus, an agent's decision rule for a node can specify a different strategy for each assignment of values to the node's parents. In effect, each such assignment corresponds to an information set. A randomizing decision rule for a decision node is simply a CPD: a distribution over its values for each instantiation of its parents.

Each utility node is associated with an agent, and represents a component in that agent's payoff. The utility node takes real values as a deterministic function of its parents. Thus, that component of the agent's payoff depends only on a subset of the variables in the MAID. The agent's overall utility is the sum of the utilities obtained at its different utility nodes. It is easy to show that a MAID defines an extensive-form game. If we use CPDs and decision rules that are tree-structured, then the MAID representation is no larger than the corresponding extensive-form representation, and is exponentially smaller in many cases. Note that a graphical game is simply a MAID where each agent has a single decision node and a single utility node, and where the parents of an agent n's utility node are the decision nodes for the agents in Fam_n.

3 Continuation Methods

We begin with a high-level overview of continuation methods, referring the reader to [Watson, 2000] for a more detailed discussion. Continuation methods work by solving a simpler perturbed problem and then tracing the solution as the magnitude of the perturbation decreases, converging to a solution to the original problem.

More precisely, let λ be a scalar parameterizing a continuum of perturbed problems. When $\lambda = 0$, the perturbed problem is the original one; when $\lambda = 1$, the perturbed problem is one for which the solution is known. Let $\boldsymbol{w}$ represent the vector of real values of the solution. For any perturbed problem defined by λ, we characterize solutions by the equation $F(\boldsymbol{w}, \lambda) = \mathbf{0}$, where F is a real-valued vector function (so that $\mathbf{0}$ is a vector of zeros). The function F is such that if $F(\boldsymbol{w}, \lambda) = \mathbf{0}$ holds then $\boldsymbol{w}$ is a solution to the problem perturbed by λ.

The continuation method traces solutions along the manifold of solution pairs $(\boldsymbol{w}, \lambda)$ satisfying $F(\boldsymbol{w}, \lambda) = \mathbf{0}$. Specifically, if we have a solution pair $(\boldsymbol{w}, \lambda)$, we would like to trace that solution to adjacent solutions. Differential changes to $\boldsymbol{w}$ and λ must cancel out so that F remains equal to $\mathbf{0}$. Thus, locally, changes $d\boldsymbol{w}$ and $d\lambda$ along the path must obey $\nabla_{\boldsymbol{w}} F(\boldsymbol{w}, \lambda)\, d\boldsymbol{w} = -d\lambda\, \nabla_\lambda F(\boldsymbol{w}, \lambda)$, which is equivalent to the matrix equation

$$\begin{bmatrix} \nabla_{\boldsymbol{w}} F & \nabla_\lambda F \end{bmatrix} \begin{bmatrix} d\boldsymbol{w} \\ d\lambda \end{bmatrix} = 0 \ . \qquad (2)$$

If the matrix $[\nabla_{\boldsymbol{w}} F \ \nabla_\lambda F]$ has a null-space of rank 1 everywhere, the curve is uniquely defined. If properly constructed, the curve starting at $\lambda = 1$ is guaranteed to cross $\lambda = 0$, at which point the corresponding value of $\boldsymbol{w}$ is a solution to the original problem. A continuation method begins at the known solution for $\lambda = 1$. The null-space of the Jacobian ∇F at a current solution $(\boldsymbol{w}, \lambda)$ defines a direction, along which the solution is moved by a small amount. The process then repeats, tracing the curve until $\lambda = 0$. The cost of each step in this computation, given the Jacobian, is cubic in the size of the Jacobian, due to the required matrix operations.

Continuation methods involve the tracing of a dynamical system through the continuous variation of the parameter λ. For computational purposes, discrete steps must be taken. As a result, error inevitably accumulates as the path is traced. One can use several techniques to reduce the error, which we do not describe for lack of space. Unfortunately, these techniques can potentially send the algorithm into a cycle, and in practice they occasionally do. If the algorithm cycles, random restarts and a decrease in step size can improve convergence.

4 Continuation Methods for Games

We now review the work of Kohlberg and Mertens [1986] and GW on applying the continuation method to the task of finding equilibria in games. These algorithms form the basis for our extension to structured games, described in the next section. The continuation method perturbs the game by adding λ times a fixed bonus to each agent's payoffs, such that an agent's bonus depends only on its own actions. If the bonuses are large enough (and unique), the bonuses dominate the original game structure, and the agents need not consider their opponents' plays. Thus, for $\lambda = 1$, the perturbed game has only one equilibrium: each agent plays the action with the largest bonus. We then use the continuation method to follow a path in the space of λ and equilibrium profiles for the resulting perturbed game, decreasing λ until it is zero; at this point, the corresponding strategy profile is an equilibrium of the original game. We now make this intuition more precise.

4.1 Normal Form Games

In order to apply Eq. (2), we need to characterize the equilibria of perturbed games as the zeros of a function F. We first define an auxiliary function measuring the benefit of deviating from a given strategy profile. Specifically, $V(\sigma)$ is a vector payoff function of the payoff to agent n for deviating from the mixed strategy profile σ by playing each action s:

$$V_s(\sigma) = \sum_{\boldsymbol{t} \in S_{-n}} G_n(s, \boldsymbol{t}) \prod_{i \in N - \{n\}} \sigma_{\boldsymbol{t}_i} .$$

We now define a *retraction operator* $R : \mathbb{R}^m \to \Sigma$ to be an operator that maps arbitrary m-vectors $\boldsymbol{w}$ to the point in the space Σ of mixed strategies which is nearest to $\boldsymbol{w}$ in Euclidean distance. As outlined in the structure theorem of Kohlberg and Mertens [1986], an equilibrium σ is recoverable from $\sigma + V(\sigma)$ by the retraction operator R: $R(\sigma + V(\sigma)) = \sigma$. In fact, this condition is a full characterization of equilibria. Thus, we can define an equilibrium as a solution to the equation $\sigma = R(\sigma + V(\sigma))$. Conversely, if $\sigma = R(\boldsymbol{w})$ and $\boldsymbol{w} = \sigma + V(\sigma)$ we have the equivalent condition that $\boldsymbol{w} = R(\boldsymbol{w}) + V(R(\boldsymbol{w}))$. Thus, we can search for a point $\boldsymbol{w} \in \mathbb{R}^m$ which satisfies this equality, in which case $R(\boldsymbol{w})$ is guaranteed to be an equilibrium.

We now define the game perturbation and associated continuation method. For a target game G with payoff function V, we create an easily soluble perturbed game by adding an m-vector $\boldsymbol{b}$ of unique bonuses that agents receive for playing certain actions, independently of what all other agents do. In perturbing the game G by $\boldsymbol{b}$, we arrive at a new game $G + \boldsymbol{b}$ in which for each $s \in S_n$, and for any $\boldsymbol{t} \in S_{-n}$, $(G + \boldsymbol{b})_n(s, \boldsymbol{t}) = G_n(s, \boldsymbol{t}) + \boldsymbol{b}_s$. If we make $\boldsymbol{b}$ sufficiently large, then $G + \boldsymbol{b}$ has a unique equilibrium, in which each agent plays the pure strategy s for which $\boldsymbol{b}_s$ is maximal.

Now, let V be the payoff function for the target game G. The induced payoff function $V + \lambda\boldsymbol{b}$ is also perturbed from V by $\lambda\boldsymbol{b}$. Thus, the form of our continuation equation is:

$$F(\boldsymbol{w}, \lambda) = \boldsymbol{w} - R(\boldsymbol{w}) - (V(R(\boldsymbol{w})) + \lambda\boldsymbol{b}) \ . \tag{3}$$

We have that $V + \lambda\boldsymbol{b}$ is the payoff function for the perturbed game $G + \lambda\boldsymbol{b}$, so $F(\boldsymbol{w}, \lambda)$ is zero if and only if $R(\boldsymbol{w})$ is an equilibrium of $G + \lambda\boldsymbol{b}$. At $\lambda = 0$ the game is unperturbed, so $F(\boldsymbol{w}, 0) = \mathbf{0}$ iff $R(\boldsymbol{w})$ is an equilibrium of G.

The expensive step in the continuation method is the calculation of the Jacobian $\nabla_{\boldsymbol{w}} F$, required for the computation that maintains the constraint of Eq. (2). Here, we have that $\nabla_{\boldsymbol{w}} F = I - (I + \nabla V)\nabla R$, where I is the $m \times m$ identity matrix. The hard part is the calculation of ∇V. For pure strategies $s_1 \in S_{n_1}$ and $s_2 \in S_{n_2}$, the value at location (s_1, s_2) in $\nabla V(\sigma)$ is equal to the expected payoff to agent n_1 when it plays the pure strategy s_1, agent n_2 plays the pure strategy s_2, and all other agents act according to the strategy profile σ:

$$\nabla V_{s_1,s_2}(\sigma) = \sum_{\boldsymbol{t} \in S_{-n_1,-n_2}} G_{n_1}(s_1, s_2, \boldsymbol{t}) \prod_{i \in N - \{n_1, n_2\}} \sigma_{\boldsymbol{t}_i} \ . \tag{4}$$

Computing Eq. (4) requires a number of multiplications which is exponential in the game size; the sum is over the exponentially large space $S_{-n_1,-n_2} = \prod_{n \in N - \{n_1, n_2\}} S_i$.

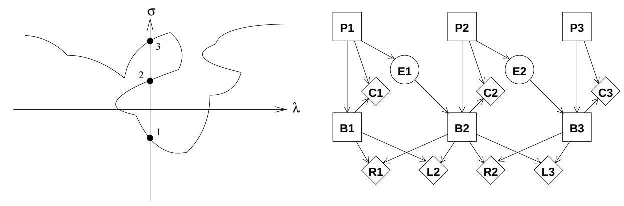
4.2 Extensive Form Games

The same method applies to extensive-form games, using the sequence form parameterization of strategies. We first need to define the bonus vector, and the retraction operator which allows us to characterize equilibria.

The bonus vector in the extensive-form adds a bonus for each sequence of each agent. GW show that a sufficiently large bonus guarantees a unique equilibrium for the perturbed game. The retraction operator R takes a general vector and projects it onto the valid region of sequence forms. As all of the constraints are linear, the projection amounts to a decomposable quadratic program (QP) that can be solved quickly. We employ standard QP methods. The Jacobian of the retraction is easily computable from the set of active constraints.

The solution for the sequence form is now surprisingly similar to that of the normal-form. The key property of the sequence form strategy representation is that the payoff function is a multi-linear function of the extensive-form parameters, as shown in Eq. (1). The elements of the Jacobian ∇V also have the same general structure. In particular the element corresponding to sequence s_1 for agent n_1 and sequence s_2 for agent n_2 is

$$\nabla V_{s_1,s_2}(\sigma) = \sum_{z \in Z_{s_1,s_2}} G_{n_1}(z) \prod_{i \in N - \{n_1, n_2\}} \sigma_i(z) \tag{5}$$



(a) (b)

Figure 1: (a) An abstract diagram of the path. The horizontal axis represents λ and the vertical axis represents the space of strategy profiles (actually multidimensional). The algorithm starts on the right at $\lambda = 1$ and follows the dynamical system until $\lambda = 0$ at point 1, where it has found an equilibrium of the original game. It can continue to trace the path and find the equilibria labeled 2 and 3. (b) Two-stage road building MAID for three agents.

where Z_{s_1,s_2} is the set of leaves that are consistent with the sequences s_1 (for agent n_1) and s_2 (for agent n_2). We take Z_{s_1,s_2} to be the empty set (and hence $\nabla V = 0$) if s_1 and s_2 are incompatible. Eq. (5) is precisely analogous to Eq. (4) for normal-form games. We have a sum, over outcomes, of the utility of the outcome multiplied by the strategy probabilities for all other agents. Note that this sum is over the leaves of the tree, which may be exponential in the number of agents.

Zero-probability actions in extensive-form games give rise to an additional subtlety. Such actions induce a probability of zero for entire trajectories in the tree, possibly leading to equilibria based on unrealizable threats and other undesirable phenomena. For us, they can also lead to bifurcations in the continuation path, preventing convergence. Thus, we constrain all sequence form parameters to be greater than or equal to ϵ for some small ϵ. This constraint ensures that the continuation path is a 1-manifold. The algorithm thus finds an equilibrium to a perturbed game, where agents have a small probability of choosing an unintended action. As ϵ tends to zero, these equilibria converge to *perfect equilibria* of the original game [Owen, 1995], a (nonempty) subset of all equilibria. For ϵ small enough, continuity implies that there is always an exact perfect equilibrium in the vicinity of the perturbed equilibrium, which can easily be found using local search.

4.3 Path Properties

In the case of normal-form games, the structure theorem of Kohlberg and Mertens [1986] implies that, with probability one over all choices for $\boldsymbol{b}$, the path of the algorithm is a one-manifold without boundary. GW provide an analogous structure theorem that guarantees the same property for extensive-form games. Figure 1(a) shows an abstract representation of the path followed by the continuation method. The equilibrium for large positive λ is unique, so the one-manifold cannot double back to the side of $\lambda = \infty$. Furthermore, the perturbed games along the path can have only a finite number of discrete equilibria, so the path cannot travel back and forth indefinitely. Therefore, it must cross the $\lambda = 0$ hyperplane at least once, yielding an equilibrium. In fact, the path may cross multiple times, yielding many equilibria in a single run. As the path must eventually continue to the $\lambda = -\infty$ side, it will find an odd number of equilibria when run to completion.

In both normal-form and extensive-form games, the path is piece-wise polynomial, with each piece corresponding to a different support set of the strategy profile. These pieces are called *support cells*. The path is not smooth at cell boundaries, due to discontinuities in the Jacobian of the retraction operator, and hence in $\nabla_{\boldsymbol{w}} F$, when the support changes. Thus, in following the path, care must be taken to step up to these boundaries exactly.

In the case of two agents, the path is piece-wise linear and, rather than taking steps, the algorithm can jump from "elbow" to "elbow" along this path. When this algorithm is applied to a two-agent game and a particular bonus vector is used, the steps from support cell to support cell that the algorithm takes are exactly equal to the pivots of the Lemke-Howson solution algorithm [Lemke and Howson, 1964] for two-agent games, and the two algorithms find precisely the same set of solutions. Thus, the continuation method is a strict generalization of the Lemke-Howson algorithm that allows different perturbation rays and games of more than two agents.

4.4 Iterated Polymatrix Approximation

Because perturbed games may themselves have an exponential number of equilibria, and the path may wind back and forth through any number of them, the continuation algorithm can take a while to trace its way back to a solution to the original game. We can speed up the algorithm using an initialization procedure based on the *iterated polymatrix approximation* (IPA) algorithm of GW. A *polymatrix game* is a normal-form game where the payoffs to a agent n are equal to the sum of the payoffs from a set of two-agent games, each involving n and another agent. Polymatrix games can be solved quickly using the Lemke-Howson algorithm [1964].

Given a normal-form game G and a strategy profile σ, we can construct a polymatrix game P_σ whose Jacobian at σ is the same as the Jacobian of G's payoff function V at σ. The game P_σ is a linearized approximation to G around σ, and can be computed efficiently from the Jacobian of G. GW provide an iterative algorithm that, in each step, takes a profile σ and improves it using the solution of the polymatrix approximation P_σ. This algorithm is not guaranteed to converge, but in practice, it quickly moves "near" a good solution. We then construct a perturbed game close to the original game for which this approximate equilibrium is an exact equilibrium. The continuation method is then run from this starting point to find an exact equilibrium of the original game.

5 Exploiting Structure

As mentioned above, the calculation of ∇V at each step of the algorithm consumes most of the time. Both in normal-form and (in the worst case) in extensive-form games, it requires time that is exponential in the number of agents. However, as we show in this section, when using a structured representation such as a graphical game or a MAID, we can effectively exploit the structure of the game to drastically reduce the computational time required.

5.1 Graphical Games

Consider the computation of the normal-form Jacobian in Eq. (4). The key insight is that the choice of strategy for an agent outside the family of n_1 does not affect G_{n_1}. This observation allows us to compute the n_1 entries in the Jacobian locally, considering only n_1's family. More precisely, we can consider two cases. If $n_2 \notin \mathit{Fam}_{n_1}$, then

$$\nabla V_{s_1,s_2}(\sigma) = \sum_{\boldsymbol{t} \in S_{-n_1}} G_{n_1}(s_1, \boldsymbol{t}) \prod_{i \in \mathit{Fam}_{n_1} - \{n_1\}} \sigma_{\boldsymbol{t}_i} \; . \tag{6}$$

Recalling that S_{-n_1} is a vector of actions only of $\mathit{Fam}_{n_1} - \{n\}$, we see that this computation is exponential only in the family size of n_1. Furthermore, this value does not depend on s_2 or n_2, and can therefore be calculated once and copied for all agents $n_2 \notin \mathit{Fam}_{n_1}$ and for all choices of their strategies. If $n_2 \in \mathit{Fam}_{n_1}$, then we simply have:

$$\nabla V_{s_1,s_2}(\sigma) = \sum_{\boldsymbol{t} \in S_{-n_1,-n_2}} G_{n_1}(s_1, s_2, \boldsymbol{t}) \prod_{i \in \mathit{Fam}_{n_1} - \{n_1,n_2\}} \sigma_{\boldsymbol{t}_i} \; . \tag{7}$$

Letting f be the maximal family size, and d the maximal number of actions per agent, we have that the computation of the Jacobian requires time $O(|N| f d^f + |N|^2)$.

5.2 MAIDs

The Jacobian for MAIDs

To find equilibria in MAIDs, we extend the sequence form continuation method of Section 4.2. As above, our key task is computing the Jacobian of Eq. (5). The Jacobian has an entry for each pair of sequences in the game (one for each agent). We therefore begin by noting that the sequence form representation for MAIDs with perfect recall is no larger than the agent's decision rules for that MAID. Due to perfect recall, the last decision node (in topological order) must have incoming edges from all of its previous actions and all parents of previous actions. Moreover, it must have an information set for any distinct agent history. Thus, the agent's decision rule for that final decision has the same size as the sequence form. Hence, the dimension m of the $m \times m$ Jacobian matrix is linear in the size of the MAID (where size, as usual, includes the size of the parameterization).

We next turn to the computation of the Jacobian entries. Eq. (5) can be rewritten as

$$\nabla V_{s_1,s_2}(\sigma) = \sum_{z \in Z_{s_1,s_2}} \frac{G_{n_1}(z) \prod_{i \in N} \sigma_i(z)}{\sigma_{n_1}(z)\sigma_{n_2}(z)} \; . \tag{8}$$

A leaf node z in the extensive-form game is simply an assignment $\boldsymbol{x}$ to all of the variables in the MAID, and $G_{n_1}(z)$ is n_1's utility given $\boldsymbol{x}$. The sequence probability $\sigma_n(z)$ is the product of the probabilities for the decisions of agent n in the assignment $\boldsymbol{x}$. Thus, Eq. (8) is an expectation of $G_{n_1}(z)/[\sigma_{n_1}(z)\sigma_{n_2}(z)]$. The expectation is over the distribution defined by the Bayesian network $\mathcal{B}_\sigma$ whose structure is the same as the MAID, and where the agents' decision nodes have CPDs determined by σ.

The agent's utility $G_{n_1}(z)$ is the sum of its utility nodes. Due to linearity of expectation, we can perform the computation separately for each of the agent's utility nodes, and then simply add up the separate contributions. Thus, we assume from here on, without loss of generality, that n_1 has only a single utility node U.

The value of $\sigma_{n_i}(z)$ $(i = 1, 2)$ depends only on the values of the set of nodes $\boldsymbol{D}_{n_i}$ consisting of n_i's decision nodes and their parents. Thus, instead of computing the probabilities for all assignments to all variables, we need only to compute the marginal joint distribution over U, $\boldsymbol{D}_{n_1}$, and $\boldsymbol{D}_{n_2}$ From this distribution, we can compute the expectation in Eq. (5).

Using Bayesian Network Inference

Our analysis above reduces the required computations significantly. We need only compute one joint distribution for every pair of agents n_1, n_2. This joint distribution is the one defined by the Bayesian network $\mathcal{B}_\sigma$. Naively, this computation requires that we execute Bayesian network inference $|N|^2$ times: once for each ordered pair of agents n_1, n_2. Fortunately, we can exploit the structure of the MAID to perform this computation much more efficiently.

The basis for our method is the *clique tree algorithm* of Lauritzen and Spiegelhalter [1998]. A clique tree for a Bayesian network $\mathcal{B}$ is a data structure defined over an undirected tree over a set of nodes $\mathcal{C}$. Each node $C_i \in \mathcal{C}$ is a subset of the nodes in $\mathcal{B}$ called a *clique*. The clique tree satisfies certain important properties. It must be *family preserving*: for each node X in $\mathcal{B}$, there exists a clique $C_i \in \mathcal{C}$ such that X and its parents are a subset of C_i. It also satisfies a *separation* requirement: if C_2 blocks the path from C_1 to C_3, then, in the distribution defined by $\mathcal{B}$, we have that the variables in C_1 are conditionally independent of those in C_3 given those in C_2.

Each clique maintains a data structure, called a *potential*, which is an unnormalized distribution over the variables in C_i. The size of the potential for C_i is therefore exponential in $|C_i|$. The clique tree inference algorithm proceeds by passing messages from one clique to another in the tree. The messages are used to update the potential in the receiving clique. After a process in which messages have been sent in both directions over each edge in the tree, the tree is said to be *calibrated*; at this point, the potential of every clique C_i contains precisely the joint distribution over the variables in C_i according to $\mathcal{B}$.

We can use the clique tree algorithm to perform inference over $\mathcal{B}_\sigma$. Now, consider the final decision node for agent n_i. Due to the perfect recall assumption, all of n_i's previous decisions and all of their parents are also parents of this decision node. The family preservation property therefore implies that $\boldsymbol{D}_{n_i}$ is fully contained in some clique. Thus, the expectation of Eq. (8) requires the computation of the joint distribution over three cliques in the tree: the one containing U, the one containing $\boldsymbol{D}_{n_1}$, and the one containing $\boldsymbol{D}_{n_2}$. We need to compute this joint distribution for every pair of agents n_1, n_2.

The first key insight is that we can reduce this problem to one of computing the joint marginal distribution for all pairs of cliques in the tree. Assume we have computed $P_{\mathcal{B}}(C_i, C_j)$ for every pair of cliques C_i, C_j. Now, consider any triple of cliques C_1, C_2, C_3. There are two cases: either one of these cliques is on the path between the other two, or not. In the first case, assume without loss of generality that C_2 is on the path from C_1 to C_3. In this case, by the separation requirement, we have that $P_{\mathcal{B}}(C_1, C_2, C_3) = P_{\mathcal{B}}(C_1, C_2)P_{\mathcal{B}}(C_2, C_3)/P_{\mathcal{B}}(C_2)$. In the second case, there exists a unique clique C^* which blocks the paths between any pair of these cliques. Again, by the separation property, C^* renders these cliques conditionally independent, so we can use a similar method to compute $P_{\mathcal{B}}(C_1, C_2, C_3)$.

Thus, we have reduced the problem to one of computing the marginals over all pairs of cliques in a calibrated clique-tree. We can use dynamic programming to execute this process efficiently. We construct a table that contains $P_{\mathcal{B}}(C_i, C_j)$ for each pair of cliques C_i, C_j. We construct the table in order of length of the path from C_i to C_j. The base case is when C_i and C_j are adjacent in the tree. In this case, we have that $P_{\mathcal{B}}(C_i, C_j) = P_{\mathcal{B}}(C_i)P_{\mathcal{B}}(C_j)/P_{\mathcal{B}}(C_i \cap C_j)$. The probability expressions in the numerator are simply the clique potentials in the calibrated tree. The denominator can be obtained by marginalizing either of the two cliques. For cliques C_i and C_j that are not adjacent, we let C_k be the node adjacent to C_j on the path from C_i to C_j. The clique C_k is one step closer to C_i, so, by construction, we have already computed $P(C_i, C_k)$. We can now apply the separation property again:

$$P_{\mathcal{B}}(C_i, C_j) = \sum_{C_i - C_k} \frac{P_{\mathcal{B}}(C_i, C_k)P_{\mathcal{B}}(C_k, C_j)}{P_{\mathcal{B}}(C_k)} . \tag{9}$$

Let ℓ be the number of cliques in the tree, and d be size of the largest clique (the number of entries in its potential). The cost of calibrating the clique tree for $\mathcal{B}_\sigma$ is $O(\ell d)$. The cost of computing Eq. (9) for all pairs of cliques is $O(\ell^2 d^3)$. Finally, the cost of computing the s_i, s_j entry of the Jacobian is $O(d^4)$. In games where interactions between the agents are highly structured, the size d of the largest clique can be a constant even as the number of agents grows. In this case, the complexity grows only quadratically in the number of cliques, and hence also in the number of agents.

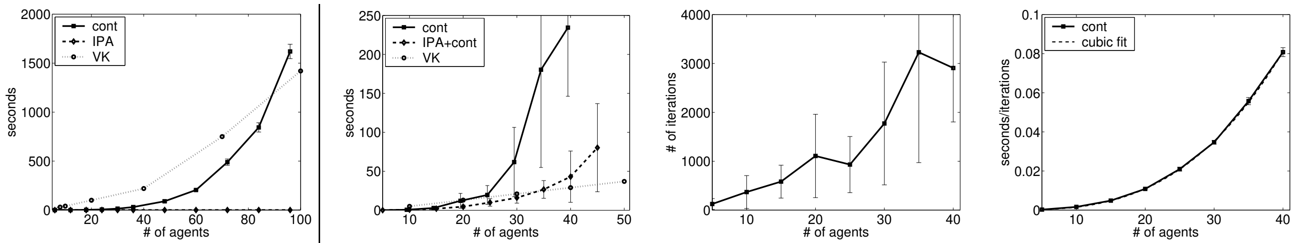
6 Results

6.1 Graphical Games

We compared two versions of our algorithm: cont, the simple continuation method, and IPA+cont, the continuation method with the IPA initialization. We compared our results to the published results of the the algorithm of Vickrey and Koller [2002] (VK hereafter). The VK method only returns ϵ-equilibria, but their approach is the only one that applies to graphical games whose interaction structure is not a (biconnected) tree and for which timing results are available. The VK paper contains several different algorithms. We compared against the algorithm which had the smallest approximation error for a given problem.

Following VK, our algorithms were run on two classes of games, of varying size. The Road game, denoting a situation where agents must build in land plots along a road, is played on a 2-by-L grid; each agent has three actions, and its payoffs depend only on the actions of its (grid) neighbors. Following VK, we constructed a game where the payoff for an agent is simply the sum of payoffs of games played separately with its neighbors, and where each such subgame has the payoff structure of rock-paper-scissors. This game is, in fact, a polymatrix game, and hence is very easy to solve using our methods. We also experimented with a ring graph with three actions per agent and random payoffs.

For each class of games, we chose a set of game sizes to run on. For each, we selected, randomly in cases where the payoffs were random, a set of at least ten (and up to one hundred for MAIDs) test games to solve. We then solved each game



(a) (b) (c) (d)

Figure 2: Results for graphical games: (a) Running time for road game with rock-paper-scissors payoffs. Results for ring game with random payoffs: (b) running time; (c) number of iterations of cont; (d) average time per iteration of cont.

with a different random perturbation vector, $\boldsymbol{b}$, and recorded the time and number of iterations necessary to reach the first equilibrium. We then averaged over test cases. The error bars show the variance due to the choice of perturbation vector and, for random games, the choice of game. For smaller games, the algorithms always converged to an equilibrium. In about 40% of the larger games (more than 20 agents), the algorithms did not converge on the first trial; in these cases, we restarted the same game with a different random perturbation vector. On average, about 2 restarts were sufficient for these difficult games. In a few large graphical games (*e.g.* 9% of games with 45 agents), IPA did not converge after 10 restarts; in these cases we did not record results for IPA+cont. In the other restarted cases, we recorded the time for the converging run. Our results are shown in Figures 2(a,b).

In all cases, our algorithm found an equilibrium with error at most 10^{-12}, essentially machine precision. In the Road games, we compared against the times for VK using their hill climbing method which found ϵ-equilibria with error of 10^{-4}. In these games, the cont method is more efficient for smaller games, but then becomes more costly. Due to the polymatrix nature of this game, the IPA+cont solves it immediately with the Lemke-Howson algorithm, and is therefore significantly less expensive than VK.

In the random-payoff ring games, VK had an equilibrium error of about 0.01 using their cost minimization method with a grid discretization of $1/3$. Here, our algorithms are more efficient than VK for smaller games (up to 20–30 agents), with IPA+cont performing considerably better than cont. However, the running time of our algorithms grows more rapidly than that of VK, so that for larger games, they become impractical. Nevertheless, our algorithms performed well in games with up to 50 agents and 3 actions per agent, games which were previously intractable for exact algorithms.

Here, we also plotted the number of iterations and time per iteration for cont in Figures 2(c,d). The number of iterations varies based both on the game and perturbation ray chosen. However, the time per iteration is almost exactly cubic as predicted. We note that, when IPA is used, the continuation method converges almost immediately (within a second).

6.2 MAIDs

Koller and Milch [2001] define a *relevance graph* over the decision nodes in a MAID, where there is an edge from D_1 to D_2 if the decision rule at D_1 impacts the choice of decision rule at D_2. They show that the task of finding equilibria for a MAID can be decomposed, in that only decision nodes in the same strongly connected component in the relevance graph must be considered together. However, their approach is unable to deal with structure within a strongly connected component, and they resorted to converting the game to extensive form, and using a standard equilibrium solver. Our approach addresses the complementary problem, dealing specifically with this finer-grained structure. Thus, we focused our experiments on MAIDs with cyclic relevance graphs.

We ran our algorithms on two classes of games, with varying sizes. The first, a simple chain, alternates between decision and chance nodes with each decision node belonging to a different agent. Each agent has two utility nodes, each connected to its decision node and to a neighbor's (except for the end agents who have one utility node for their single neighbor). All probability tables and payoff matrices are random. Our second example is shown in Figure 1(b). It is an extension of the graphical road game from above. Each agent must submit plans simultaneously (P_1–P_n) for the type of building (home or store) that they will build along a road. However, building (B_1–B_n) proceeds from left to right and so before committing to a build, an agent can see a noisy estimate of the plans of the agent to its left (E_1-E_n). Agents would prefer to be the first one to start a new type of building (*i.e.*, be different than their left neighbors, but the same as their right neighbors). They also take a penalty if their building and plan decisions differ. Carefully chosen payoffs ensure non-trivial mixed strategies.

Figures 3(a,b) show the running times for computing an equilibrium as the number of agents is increased for both types of games. We compared our results to those achieved by converting the game to extensive-form and running Gambit, a standard equilibrium computation package. Our timing results for Gambit do not include the time for the conversion to extensive-form.

Figures 3(c,d) show the number of iterations and running time per iteration for the case of the two-stage road game. The running time per iteration is once again well fit by a cubic. The variance is mainly due to the execution of the retraction operator whose running time depends on the number of strategies in the support.

7 Discussion and Conclusions

In the last few years, several papers have addressed the issue of finding equilibria in structured games. For graphical games, the exact algorithms proposed so far apply only to the very restricted class of games where the interaction structure

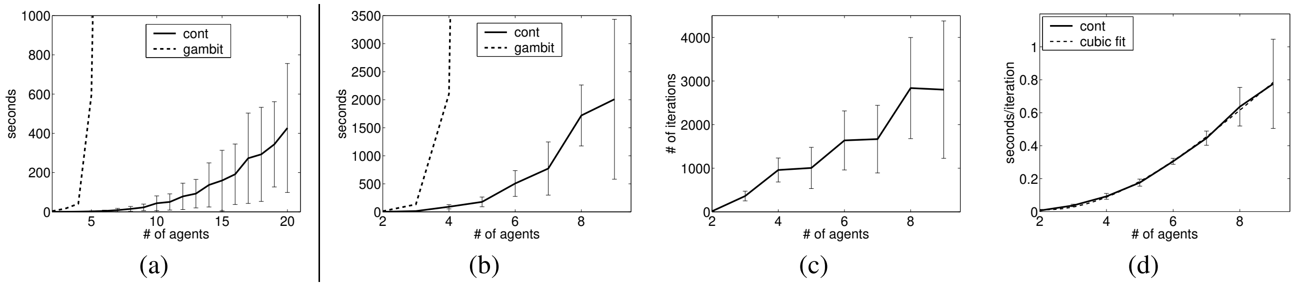


Figure 3: Results for MAIDs: (a) Running times for the chain MAID. Results for two-stage Road MAID: (b) running time; (c) number of iterations; (d) time per iteration.

is an undirected tree, and where each has only two possible actions [Kearns *et al.*, 2001; Littman *et al.*, 2002].

There have been several algorithms proposed for the computation of ϵ-equilibria in general graphical games, most of which (implicitly or explicitly) define an equilibrium as a set of constraints over a discretized space of mixed strategies, and then use some constraint solving method: Kearns *et al.* [2001] use a tree-propagation algorithm; Vickrey and Koller [2002] use variable elimination methods (VK1); and Ortiz and Kearns [2003] use arc-consistency constraint propagation followed by search. Vickrey and Koller [2002] also propose a gradient ascent algorithm (VK2). The running times of KLS and VK1 both depend on the tree-width of the graph, whereas the running times of our algorithm, VK2, and OK depend on the degree of the graph. However, these latter three algorithms all require multiple iterations and no bounds are currently known on the number of iterations required.

For MAIDs, Koller and Milch [2001] (KM) define a notion of independence between agents' decision, and provide an algorithm that can decompose the problem based on fairly coarse independence structure. Our algorithm is able to exploit a much finer-grained structure, resolving an open problem left by KM. La Mura [2000] (LM) proposes a continuation method for finding one or all equilibria in a G net, a representation which is very similar to MAIDs. This proposal only exploits a very limited set of structural properties (a strict subset of KM and of our algorithm). The proposal was also never implemented, and several issues regarding non-converging paths seem unresolved.

We have presented an algorithm for computing exact equilibria in structured games. Our algorithm is based on the methods of GW, but shows how the key computational steps in their approach can be performed much more efficiently by exploiting the game structure. Our method allows us to provide the first exact algorithm for general graphical games, and the first algorithm that takes full advantage of the independence structure of a MAID. Our methods can find exact equilibria in games with large numbers of agents, games which were previously intractable for exact methods.

Acknowledgments. This work was supported by ONR MURI Grant N00014-00-1-0637, and by Air Force contract F30602-00-2-0598 under DARPA's TASK program.

References

[Govindan and Wilson, 2002] S. Govindan and R. Wilson. Structure theorems for game trees. *Proc. Natl Academy of Sciences*, 99(13):9077–9080, 2002.

[Govindan and Wilson, 2003a] S. Govindan and R. Wilson. Computing Nash equilibria by iterated polymatrix approximation. *J. Economic Dynamics and Control*, 2003. to appear.

[Govindan and Wilson, 2003b] S. Govindan and R. Wilson. A global Newton method to compute Nash equilibria. *J. Economic Theory*, 2003. to appear.

[Howard and Matheson, 1984] R. A. Howard and J. E. Matheson. Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*, volume 2, pages 719–762. Strategic Decision Group, 1984. article dated 1981.

[Kearns *et al.*, 2001] M. Kearns, M. L. Littman, and S. Singh. Graphical models for game theory. In *Proc. UAI*, 2001.

[Kohlberg and Mertens, 1986] E. Kohlberg and J.-F. Mertens. On the strategic stability of equilibria. *Econometrica*, 54(5):1003–1038, September 1986.

[Koller and Megiddo, 1992] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Bahavior*, 4:528–552, 1992.

[Koller and Milch, 2001] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. In *Proc. IJCAI*, pages 1027–1034, 2001.

[La Mura, 2000] P. La Mura. Game networks. In *Proc. UAI*, pages 335–342, 2000.

[Lauritzen and Spiegelhalter, 1998] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society*, B 50(2):157–224, 1998.

[Lemke and Howson, 1964] C. E. Lemke and J. T. Howson, Jr. Equilibrium points in bimatrix games. *J. Society of Applied Mathematics*, 12(2):413–423, June 1964.

[Littman *et al.*, 2002] M. L. Littman, M. Kearns, and S. Singh. An efficient exact algorithm for singly connected graphical games. In *NIPS-14*, volume 2, pages 817–823, 2002.

[McKelvey and McLennan, 1996] R. D. McKelvey and A. McLennan. Computation of equilibria in finite games. In *Handbook of Computational Economics*, vol. 1, pages 87–142. Elsevier, 1996.

[Nash, 1951] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 52(2):286–295, September 1951.

[Ortiz and Kearns, 2003] L. E. Ortiz and M. Kearns. Nash propagation for loopy graphical games. In *NIPS-15*, 2003. to appear.

[Owen, 1995] G. Owen. *Game Theory*. Academic Press, UK, 1995.

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

[von Stengel, 1996] B. von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14, 1996.

[Watson, 2000] L. T. Watson. Theory of globally convergent probability-one homotopies for nonlinear programming. *SIAM J. on Optimization*, 11(3):761–780, 2000.