An Efficient, Exact Algorithm for Solving Tree-Structured Graphical Games

Littman, Kearns, Singh
As presented by Rachael Affenit

Introduction

In this paper, the authors discuss algorithms for applying graph theory to multi-agent systems by representing them as graphical games in order to find their global Nash Equilibria (NE). The process of converting these systems to graphical games involves representing each agent as a single vertex, and each two-way interaction between those agents as an edge between the vertices involved in the interaction. While any multi-agent network can be represented as a graphical game, this paper and the algorithms it outlines are specifically designed for tree-structured graphical games with n players who each have two possible actions.

In a non-graphical game, each player has a payoff matrix $M_i(\vec{x})$ for its pure strategies, where $\vec{x}$ is its set of possible actions, {0,1}. This is an n-dimensional matrix, with 2^n entries because player has two choices. For mixed strategies, each player has a probability p_j of choosing strategy 1, and probability $1-p_j$ of choosing strategy 0, and the payoff matrix is modified accordingly.

For the purposes of this paper, we consider an n-player graph game that represents a multi-agent system, which contains a graph G with n vertices, each representing a player, and a set of n local payoff matrices M, each with indices representing each of k neighbors. So for local payoff matrix $M_i(\vec{x})$, there are 2^k entries where k is the number of neighbors of vertex i, including the vertex itself. For instance, consider a case where we have 7 players in the tree structure shown below in Figure 1. A payoff matrix for $M_0(\vec{x})$ has 3 dimensions, as vertex 0 has 3 neighbors, {0, 1, 2}.

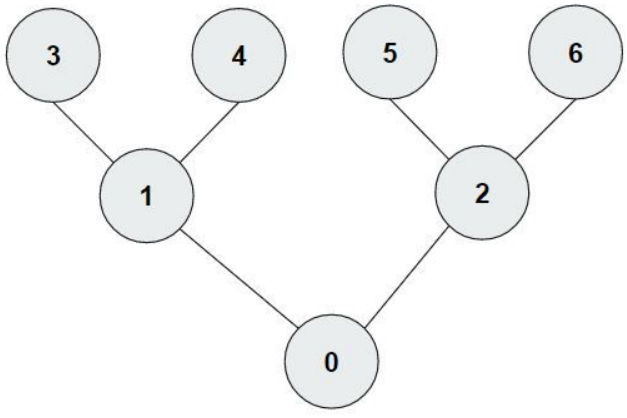

Figure 1. A tree-structured graph game representing a 7-agent network

The authors describe their previous work, which includes two algorithms. Their first algorithm approximated Nash Equilibria for tree-structured graph games, which was accomplished in polynomial time with respect to the number of agents/vertices and the accuracy of the Nash Equilibria approximation. The second computed all exact Nash Equilibria for tree-structured graph games, which was accomplished in exponential time with respect to the number of agents/vertices.

The newly proposed algorithm that the authors describe in this paper computes an exact global Nash Equilibrium for a tree-structured graph game, which can be accomplished in polynomial time with respect to the number of agents/vertices. The method they chose was to find partial Nash Equilibria, then prove that these partial equilibria extend to a global equilibrium where all players can agree on a best strategy from which none of them wish to deviate.

Algorithm

This algorithm has two components: a downstream pass of information, and an upstream pass of information. Leaf nodes are situated at the top of the graph, and the root node is situated at the bottom. As such, leaf nodes are upstream from a vector V of interest, and the root node is downstream.

One of the key elements of this algorithm is a data structure called a Breakpoint Policy, which defines a set of breakpoints and a set of corresponding values for vertex V. Breakpoint Policies define a mixed strategy for any vertex V based on its child's action (value) w, which is specified as follows:

> If the value played by child W falls between breakpoints, V must play a fixed value v_i given by the breakpoint policy.
>
> If the value played by child W is a breakpoint, V may play any value on the interval $[v_i, v_i + 1]$.

The breakpoint policies of all vertices are used to get all players to commit to a global Nash Equilibrium by passing them downstream, and assigning final values upstream.

Part 1: Downstream Pass

As a general overview of the downstream pass portion of the algorithm, any vertex V receives breakpoint policies from all of its parents, U_i. These breakpoint policies are used to calculate the breakpoint policy for V, which V then passes to its child node W. This process, shown in Figure 2 below, ensures upstream Nash Equilibrium (i.e. no vertex V wishes to change its value based on its children's values).

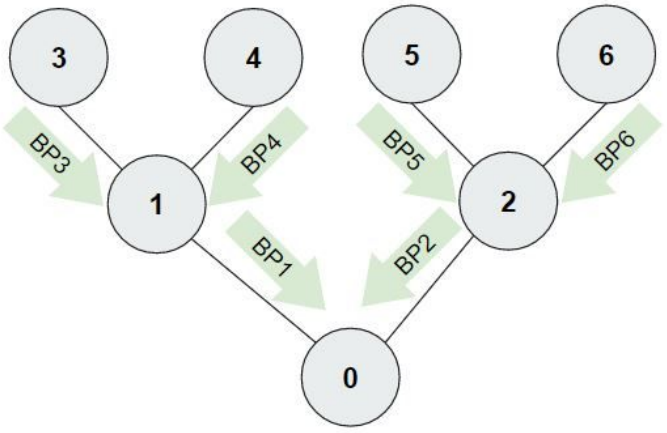

Figure 2. Shows pattern of breakpoint policies passed downstream

The first step of this part of the algorithm is to calculate the best response for vector V, the vector of interest. This function is defined as follows:

$$\Delta v(\vec{u}, w) \equiv M_V(1, \vec{u}, w) - M_V(0, \vec{u}, w)$$

From the function definition above, we can see that if the value of Δv is negative, then action 1 is the best response of vector V; if the value of Δv is positive, then action 0 is the best response. However, if the value of Δv is exactly 0, then any mixed strategy can be chosen for vector V. This is the condition of interest for the rest of this algorithm; we are interested in finding breakpoint policies for each vertex such that for a subgraph G^V which describes a subtree of the overall tree with a root at vertex V, an upstream Nash Equilibrium can be found by setting the value of child W equal to value w. This process was described extensively in one of the previous algorithms.

We wish to prove that the downstream pass produces an upstream Nash Equilibrium, so we must prove that we can commit to a value for each vertex as defined by its Nash breakpoint policy that extends to a global equilibrium for the overall tree.

Downstream Proof

In order to do so, we must prove by induction that computing breakpoint policies from a vertex's parents results in Nash breakpoint policies for all vertices in the graph, where those policies have no more than $2 + U_b$ breakpoints and U_b is the number of breakpoints in the breakpoint policy of vertex V's parent nodes. The base case for this examines a leaf node V. If we apply the equation of Δv above to this node, since this node has no parents and therefore no $\vec{u}$, the only parameter that exerts influence on the best response is the value of w. This means we only need a single breakpoint in our breakpoint policy, at the value of w that yields $\Delta v = 0$, where vertex V is indifferent and can choose any value along its assigned interval.

For the induction step of our proof, we assume that the breakpoints of vertex V's parents occur at different values. One parent U_i is assigned an interval from the breakpoint policy, and the rest are assigned fixed values. If value w is chosen as a breakpoint, then it is part of the value set W_l, which includes any value for child W that allows V to play any mixed strategy (i.e. that results in a $\Delta v = 0$, an indifferent best strategy). To ensure upstream nash, this set can only be one of the following:

- An empty set
- A single interval
- A union of two non-floating intervals

So with the definitions described above, we know that if W plays a value in set W_i, V can choose any strategy; if V plays a value that results in an indifferent best strategy for its parents, all of its parents U can choose any strategy. Therefore, if all children play values in set W_l, or in the set of values where 0 or 1 is V's best response, all parents have indifferent best strategies, and can choose any strategy assigned in order to satisfy the best strategy requirement.

Downstream Subproof: Nash Breakpoint Policies

In order to complete our induction proof, we need to know that every fixed value w is in either W_l or the set of values where 0 or 1 is the best response. If we look at all the open intervals between breakpoints w_i, we first observe that the leftmost interval has $\Delta v = pos$ which means action 0 is the best response. We also see that the rightmost interval has $\Delta v = neg$, meaning action 1 is the best response. Since our values range from 0 to 1, we know that we must have a point where vertex V's best response changes- this is a new breakpoint. This case would fall into the set W_l by definition, as its value for $\Delta v = 0$. Therefore, all these values are contained in the sets of interest.

We can now define two intervals for vertex V's breakpoint policy for each identified breakpoint: one is [0,a], and one is [a,1], where a is the breakpoint value. Values with overlap are assigned to one interval or the other. These together constitute our Nash breakpoint policy.

Downstream Subproof: Number of Breakpoints

Finally, we must ensure that the Nash breakpoint policies computed above have no more than $2 + U_b$ breakpoints and U_b is the number of breakpoints in the breakpoint policy of vertex V's parent nodes. We know that:

- Each interval contributes at most one new breakpoint.
- The first interval contributes an additional breakpoint.
- Fixed values of 0/1 at each end contribute one each.

- A breakpoint is removed for the final interval, per breakpoint policy definition

Since the first and final bullet cancel out each other's contributions, our total number of breakpoints for vertex V is equal to the number of breakpoints in U_b plus the two fixed values. This proves the final component of the induction proof, which ensures that the downstream pass yields Nash breakpoint policies that ensure upstream Nash Equilibrium.

Part 2: Upstream Pass

The upstream pass is similar to downstream, except instead of passing breakpoint policies, it passes assigned values. Each vertex V receives an assigned value from its child as well as the child's assigned value, and computes assigned values for its parents based on the Nash breakpoint policy of parent U_i to ensure that its assigned value remains a best response. Vertex V then passes its assigned value and the value it computed for each parent to the parent in question, as shown in Figure 3 below. A special case is assumed for the root node, where we initialize its value by assuming it has a "dummy" child with no influence on the best response and constant payoff, which chooses a random value for itself and chooses a value for the root node based on the root node's Nash Breakpoint Policy.

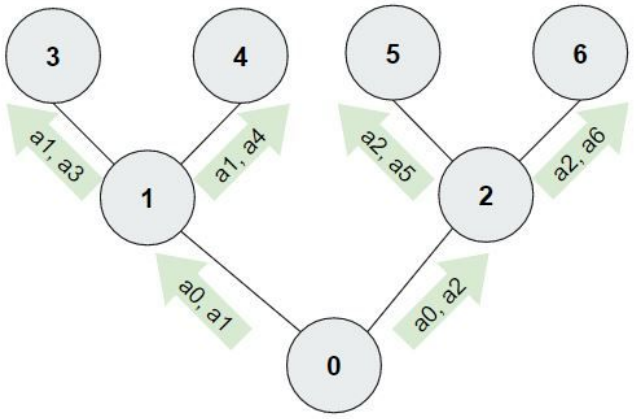

Figure 3. Shows pattern of breakpoint policies passed downstream

The computation of the value itself is based on the Nash breakpoint policy of parent U_i. If vertex V's assigned value v is not a breakpoint, its parent is assigned a fixed value corresponding to value v. If vertex V's assigned value v is in fact a breakpoint, which can only occur for one parent via the definition of a breakpoint policy, its parent is assigned a value that can be computed by setting the best response $\Delta v = 0$. We must prove by induction that our value assignment strategy results in downstream Nash Equilibrium (i.e. no vertex V wishes to change its value based on its parent's values).

Part 3: Total Algorithm Correctness

Vertex V knows that all its children have played values w from either set W_l or the set of values where 0 or 1 is V's best response. If the former is true, then vertex V having a value on its assigned interval would achieve a best response; if the latter is true, then vertex V having a fixed value of 0 or 1 would achieve a best responses. Since the upstream pass considers these parameters in assignment, we know that value v will be considered a best response with respect to its child.

Vertex V knows that if it plays its assigned value v, its child W will play value w, and value w will be considered a best response with respect to its parent. These parameters are ensured by the upstream pass.

Therefore, beginning at the root, all vertices in the graph do not wish to change their values based on their children or parents, and we have found a global Nash Equilibrium.

Running Time

If t is the number of breakpoints in the breakpoint policy for vertex V, which has k total parents, then the authors analyze the time complexity as follows:

When performing the downstream pass, sorting breakpoints should have a time complexity of $O(tlog(t))$, and finding the W_l sets and breakpoint policies should have a time complexity of $O(t2^k)$.

When performing the upstream pass, one breakpoint is taken at a time, so this step should have a time complexity of $O(log(t) + 2^k)$

The number of breakpoints is at most $2n$ where n is the number of vertices, which is given by the Number of Breakpoints subproof from the downstream component of the algorithm, giving a time complexity of $O(n^2log(n) + n^22^k)$. However, one more step can be taken- if the payoff can be simplified so that it only depends on whether action 1 is played by the neighbors of a vertex, then the 2^k term at the end becomes a k, and there is no longer an exponential dependency on k.

The final running time then, with this modification, is $O(n^2log(n) + n^2k)$.

Reference

[6] Michael L Littman, Michael J Kearns, and Satinder P Singh. An efficient, exact algorithm for solving tree-structured graphical games. In Advances in Neural Information Processing Systems, pages 817–823, 2002.