Graphical models for game theory and their application in calculating Nash equilibria

Jan Lönnberg, 51101M

23rd April 2002

Abstract

This document expands on an article by Kearns, Littman and Singh on graphical models for game theory, adding examples, clarifications and applications. The article was done for the seminar course T-79.194 at the Helsinki University of Technology in Spring 2002.

1 Introduction

1.1 Background

In many game theory problems, one wishes to find an equilibrium in which no player can improve his expected *payoff* by unilaterally changing his strategy. This form of equilibrium is called a *Nash equilibrium* after its discoverer, John F. Nash, who introduced the concept in an article [3] in 1951. Nash equilibria are quite useful when modelling situations where the participants can not trust each other to act as agreed; as a player has nothing to gain by deviating from a Nash equilibrium, it is quite unlikely that he will try to do so. Because of this property, stable solutions for many real-life problems in trade and resource allocation can be found when they are modelled as games.

Unfortunately, the most commonly used representation for game theory problems, *tabular* form, is woefully inefficient for many interesting problems. In tabular form, each player has a matrix (which in this article can mean a table of one, two or more dimensions) containing the payoff to the player for each possible combination of all players' actions. In other words, each player has a payoff matrix with an index for every player in the game, and this index has possible values corresponding to the possible actions for the player to which the index belongs. For an n-player game where each player has 2 possible actions, we get n payoff matrices containing 2^n items each. The large amount of data in the representation of the problem is reflected in the time used to find the equilibria.

1.2 Graphical models

To make it easier to compute Nash equilibria in many large problems, Kearns, Littman and Singh [2] describe in their article *Graphical Models for Game Theory* (which is the focus of this document) *graphical models* for multiplayer game theory and algorithms to calculate Nash equilibria in them efficiently in certain cases. These models and algorithms are the primary focus of this article. The graphical models are based on the assumption that in most interesting real life games, each player does not interact with all the other players; rather, they interact only with a few *neighbours* and their payoff depends solely on their own actions and those of their neighbours.

1.3 Applications of graphical models

Kearns et al. [2] suggest that the graph topology of the graphical game could represent the physical distribution and interactions of, e.g., salespeople (the neighbours of a salesperson would be the salespeople with whom he shares potential clients), the organisational structure of a company (each person interacting only with his direct superiors and subordinates) or, e.g., the topology of a computer network (with each machine e.g. balancing load). An interesting variation on the last example involves processors in a parallel computer attempting to optimise access to shared memory and I/O resources; in many massively parallel architectures (with hundred of processors) a block of memory may be shared by only a few processors.

However, it is important to note that the graph structure of the game need not be directly derived from a structure inherent in the game; it merely shows which players' actions affect each player.

2 Games, strategies and Nash equilibria

For the sake of simplicity, this article will be limited to n-player, 2-action games when describing the findings of Kearns, Littman and Singh. These games are traditionally described in tabular form as a set of matrices M_i, where $1 \leq i \leq n$ and $i \in \mathbb{N}$ and each matrix has n indices, each of which is either 0 or 1. $M_i(\vec{x}) = M_i(x_1, x_2, \ldots, x_n)$ specifies the payoff to player i when player 1 plays x_1, player 2 plays x_2 and so on. The joint action $\vec{x}$ specifies the actions chosen by all players.

To keep the mathematics simple, each payoff is assumed to have an absolute value less than or equal to 1. It is easy to convert a problem into this form by finding the payoff (for any player) with the largest absolute value and dividing all the payoffs with this absolute value.

Each player i can either choose a *pure strategy*, in which case he selects either action 0 or action 1, or a *mixed strategy*, in which case he has probability p_i of choosing 0 and therefore probability $1 - p_i$ of choosing 1.

A combination of strategies for all players in a game is called a *joint strategy*. A joint mixed strategy $\vec{p}$ is equivalent to each player i having a mixed strategy p_i.

The expected payoff $M_i(\vec{p})$ to player i under the mixed strategy $\vec{p}$ is defined as the expected value of $M_i(\vec{x})$ when the probability distribution of $\vec{x}$ is determined by the mixed strategy $\vec{p}$. The probability of a joint action $\vec{x}$ being chosen under probability distribution $\vec{p}$ is:

$$\alpha(\vec{x}) = \prod_{j=1}^{n} (p_j)^{1-x_j}(1-p_j)^{x_j}. \tag{1}$$

Thus:

$$M_i(\vec{p}) = \sum_{\vec{x}\in\{0,1\}^n}^{n} \alpha(\vec{x}) M_i(\vec{x}). \tag{2}$$

For brevity, $\vec{p}[i : p'_i]$ is defined to be the vector $\vec{p}$ with its ith component replaced by p'_i. A *Nash equilibrium* can then be defined as a mixed strategy $\vec{p}$ such that $\forall i \in \{1, 2, \ldots, n\}$: $\forall p'_i \in [0, 1] : M_i(\vec{p}) \geq M_i(\vec{p}[i : p'_i])$; in other words, no player can gain anything by deviating unilaterally (i.e. on his own, with everyone else keeping their choices the same) from a Nash equilibrium. For any game, there is at least one Nash equilibrium [3].

Approximate Nash equilibria are usually defined in a similar fashion. A mixed strategy $\vec{p}$ is an *ϵ-Nash equilibrium*, if $\forall i \in \{1, 2, \ldots, n\} : \forall p'_i \in [0, 1] : M_i(\vec{p}) + \epsilon \geq M_i(\vec{p}[i : p'_i])$; in other words, a player can gain at most ϵ by deviating unilaterally from the approximate equilibrium.

3 Graphical game model

3.1 Model description

In this section, the graphical game model proposed by Kearns, Littman and Singh [2] will be presented. Basically, in their model, a game is defined as a *graphical game* $(G, \mathcal{M})$, where G is an undirected graph with n vertices and $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ is the set of *local game matrices.*

Player i is represented by a vertex labelled i in the graph G. It is connected by undirected edges in G to its *neighbours*, $N_G(i)$, which are the vertices that represent the players with which each player interacts. The payoff to a player i is determined solely by the actions of its neighbours; for this reason, $i \in N_G(i)$ by convention.

Each local game matrix M_i has an index for each player in $N_G(i)$, each of which (in the 2-action case) is either 0 or 1 (in the multiple-action case, each possible action has a corresponding index value). $M_i(\vec{x})$ then represents the payoff to i when its neighbours play $\vec{x}$.

Of course, if the graph is not connected (in other words, it can be divided into components that are not connected to each other), the game can be split into separate subgames.

3.2 Relationship between graphical and tabular form

It is important to note that games in tabular form and graphical games are equivalent both in semantics and in expressive scope. Kearns et al. [2] note that every game in tabular form can be trivially expressed in graphical form as a graph with a vertex for each player, edges between all pairs of vertices and the same payoff matrices as the original tabular form. This is, of course, not the most compact notation for many games. Similarly, a graphical game can be converted into tabular form by taking its payoff matrices and adding indices for the non-neighbouring players, essentially copying the matrices for each possible action with no effect.

Based on the conversion from graphical to tabular form, a more efficient conversion from tabular to graphical form is possible. After performing the trivial conversion described above, redundant matrix indices and their corresponding edges are removed. Specifically, if matrices M_i and M_j have the properties that, for all possible values of $\vec{x}$, $\vec{y}$, x'_k and y'_l: $M_i(x_1, x_2, \ldots, x_k, \ldots, x_p) = M_i(x_1, x_2, \ldots, x'_k, \ldots, x_p)$, $M_j(y_1, y_2, \ldots, y_l, \ldots, y_q) = M_j(y_1, y_2, \ldots, y'_l, \ldots, y_q)$, j is the kth neighbour of i and i is the lth neighbour of j, we can remove the kth index from M_i, the lth index from M_j and the edge between i and j. More intuitively, if j's payoff doesn't depend on what i does and vice versa, we can remove the corresponding indices from both matrices and the edge between i and j.

3.3 Example

Imagine a group of students in a hurry to complete an assignment. Each student needs a workstation to do his work, but not every workstation is suitable for the assignment. To keep the example simple, there are 4 students and 4 workstations. Student 1 can use workstations a and d, student 2 can use workstations a and b, student 3 can use b and c and student 4 can use c and d for his work. The students are in a hurry, so they are only concerned about getting a workstation so they can get on with their job.

If we represent the students with unfilled circles and the workstations with filled circles, we get can draw a graph where each student is connected to the workstations he can use. This graph is in Figure 1.

Assuming that the payoff for a student is equal to his probability of getting the workstation he tries to get and that two students trying to get the same workstation each have a 50 % chance of getting that workstation, the situation can be expressed as a game:

Player 1's payoffs are in Table 1. The other players have similar payoffs, but with players and workstations shifted.

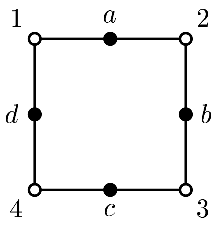

Figure 1: Example students and workstations

4:c	2:a	2:b
1:a	0.5	1
1:d	1	1

4:d	2:a	2:b
1:a	0.5	1
1:d	0.5	0.5

Table 1: Example payoff for player 1

The corresponding graph for this game is in Figure 2.

Intuitively, the optimal Nash equilibria are those where every student is guaranteed to get a machine, for example (1:a, 2:b, 3:c, 4:d).

4 Abstract algorithm for computation of Nash equilibria in trees

Kearns et al. [2] also describe approximate and exact algorithms to find Nash equilibria in graphical games, where the graph is a tree. Both are based on an "abstract" algorithm, which will be presented in this section.

The algorithm is based on the idea of computing all the Nash equilibria for a subtree with root V based on the Nash equilibria of the subtrees whose roots are children of V.

4.1 Notation

G is a tree with the root r (which can be chosen arbitrarily). In traditional tree terminology, each vertex except r then has a single parent and any number of children (Kearns et al. confusingly define parent and child the other way around). Kearns et al. refer to the ancestors for V as being *downstream* from V and its descendants as being *upstream*. The set of nodes upstream from V (including V itself) will be denoted $\mathrm{Up}_G(V)$, and the subgraph of G induced by the vertices in $\mathrm{Up}_G(V)$ will be denoted G^V. Assuming that $v \in [0,1]$ is a mixed strategy for player V, and V is the parent of U, $\mathcal{M}^U_{V=v}$ denotes the subset of $\mathcal{M}$ containing the matrices corresponding to the vertices in $\mathrm{Up}_G(U)$ in which M_U is collapsed by one index by fixing the strategy for V to v. The collapsed matrix is the same as M_U, except that the index corresponding to V is removed and the contents of the collapsed matrix are those of M_U with the removed index fixed at v.

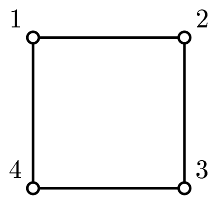

Figure 2: Graph for the example game

4.2 Downstream pass

The following description refers to any vertex V in the graphical model, which has k children $U_1, U_2, \ldots, U_k$ and parent W.

The algorithm does most of its work in the downstream pass, in which each child sends its parent a table containing the Nash equilibria for its subtree for every possible strategy chosen by the parent (the vertices are processed in depth-first order). Specifically, each child U_i sends to V a binary-valued table $T_{U_i}^V$ (denoted $T(v, u_i)$ by Kearns et al., which causes clashes in their notation), which is indexed by the possible values for the mixed strategies for V and U_i. For any pair (v, u_i), $T_{U_i}^V(v, u_i) = 1$ iff a Nash equilibrium exists for $(G^{U_i}, \mathcal{M}_{V=v}^{U_i})$ in which $U_i = u_i$. Note that the table is indexed by two real values between 0 and 1 (inclusive), and is therefore somewhat hard to store finitely. This is why this algorithm is called the abstract algorithm, and solutions to this problem will be presented later on in this article.

For each vertex V, the table to send to its parent W is calculated based on the tables received from its children. For leaf nodes, this computation is straightforward; $T_V^W(w, v) = 1$ iff strategy v is a best response for V to W choosing strategy w. For non-leaf nodes, $T_V^W(w, v) = 1$ iff a vector of mixed strategies (called a *witness* by Kearns et al.) $\vec{u} = (u_1, u_2, \ldots, u_k)$ (where each u_i is the strategy for U_i) exists such that:

1. $T_{U_i}^V(v, u_i) = 1$ for all $1 \leq i \leq k$
2. v is a best response for V to V's children choosing $\vec{u}$ and W choosing w.

In other words, each vertex V gets its subtree's Nash equilibria by making best responses to the equilibria of its children's subtrees for each of W's possible strategies. In addition, the algorithm keeps track of the witnesses $\vec{u}$ for each pair (w, v) for which $T_V^W(w, v) = 1$. For convenience, I will denote the list of witnesses for $T_V^W(w, v) = 1$ as $\omega_V^W(w, v)$. Again, the computation as described here is not straight-forward to implement, and will be described later. The root of the tree is a special case; its non-existent parent can be assumed to only have one possible fixed strategy (meaning that the table for the root is in practice one-dimensional).

It is not obvious that the above computation correctly calculates Nash equilibria for each subtree, but it can easily be explained. If the computation yields $T_V^W(w, v) = 1$ for some (w, v) and $\vec{u}$ is a witness for (w, v), then the first condition ensures (by induction) that if V plays v, there are Nash equilibria for the subtrees of the children in which each U_i plays u_i. Furthermore, the second condition guarantees that v is a best response to the choices of the parent and children. Thus, we have an equilibrium in the subtree rooted by V. Similarly, if $T_V^W(w, v) = 0$, there can be no Nash equilibrium where W plays w and V plays v (v is not a best response to any case where W plays w; the subgames all have at least one equilibrium for every possible v).

Finding the best responses to a fixed set of choices made by the parent and children is easy, as the expected payoff is a linear function of the choice made by the player; from Equation (2) we get:

$$\begin{aligned} M_i(\vec{p}) &= \left(\sum_{\substack{\vec{x} \in \{0,1\}^k \\ x_{j_i} = 0}} \prod_{\substack{j=1 \\ j \neq j_i}}^{k} \left(p_j^{1-x_j} (1-p_j)^{x_j} \right) M_i(\vec{x}) \right) p_{j_i} \\ &+ \left(\sum_{\substack{\vec{x} \in \{0,1\}^k \\ x_{j_i} = 1}} \prod_{\substack{j=1 \\ j \neq j_i}}^{k} \left(p_j^{1-x_j} (1-p_j)^{x_j} \right) M_i(\vec{x}) \right) (1 - p_{j_i}). \end{aligned}$$

Here, both sums are constants with respect to p_{j_i}, making the whole expression a linear function of p_{j_i} (the player's own choice) if all other choices are assumed to be constant.

Finding a maximum in a linear function is left as an exercise to the reader.

4.3 Upstream pass

After the downstream pass, the upstream algorithm proceeds down the tree in reverse depth-first order. It begins at the root, choosing any strategy z for the root for which $T_Z(z) = 1$ and any witness $(y_1, y_2, \ldots, y_k)$ corresponding to z. It then sends z and y_i to each child Y_i. In other words, Z chooses to play z and instructs Y_i to play y_i.

By induction, when vertex V receives a value v from its parent W and the value w that W is playing, $T(w, v) = 1$, and V can choose a witness $\vec{u}$ to (w, v), and send to each of its children U_i v and the value u_i from the witness. As $T(w, v) = 1$, v must be a best response to the choices of the parent and children.

To generate all Nash equilibria, we systematically choose every possible strategy and witness for each vertex, and then continue down the tree, backtracking when we have exhausted all possibilities for a node. This is easiest done using recursion.

4.4 Abstract algorithm pseudocode

```
TREENASH(TreeGame (G, M)):
    Calculate a depth-first ordering D of the vertices in G.
    // Downstream pass:
    for every vertex V in D
        W ← V.parent
        Fill T_V^W with 0 and ω_V^W with empty lists.
        if V is a leaf
            for all w, v ∈ [0, 1]
                if v is a best response for V to W playing w
                // Best response calculations based on M_V
                    T_V^W(w, v) ← 1
        else
            U⃗ ← V.children
            k ← U⃗.length
            for all w, v ∈ [0, 1] and all u⃗ ∈ [0, 1]^k
                if v is a best response for V to W playing w and U⃗ playing u⃗
                    if T_{U_i}^V(v, u_i) = 1 for all i = 1, 2, ..., k
                        T_V^W(w, v) ← 1
                        Add u⃗ to ω_V^W(w, v).
    // Upstream pass:
    Reverse D.
    Y⃗ ← Z.children
    Z ← G.root
    for every z for which T_Z(z) = 1
        C_Z ← z
        for every u⃗ ∈ ω_Z(z)
            for i ← 1 ... Y⃗.length
                C_{Y_i} ← u_i
            UPSTREAM(successor to Z in D)
UPSTREAM(Vertex V):
    U⃗ ← V.children
    W ← V.parent
    w ← C_W
    v ← C_V
        for every u⃗ ∈ ω_V^W(w, v)
            for i ← 1 ... U⃗.length
                C_{U_i} ← u_i
```

if V last in D
    for every vertex i in G
        Output i, C_i.
    Output separator/terminate.
else
    UPSTREAM(successor to V in D)

4.5 Intuitive interpretation of algorithm

Intuitively speaking, the downstream pass of the algorithm can be interpreted as follows: the leaf nodes start by working out which are their best responses to each possible choice their parent can make; this provides equilibria upstream from these leaf nodes for every possible choice the parent can make. The internal nodes then take all combinations of their children's upstream equilibria and all choices their parent can make, and work out the best responses to that. This continues up to the root node, which simply calculates its own best reponses to all equilibria upstream of its children.

The upstream pass can then be considered to be equivalent to the following: the root chooses one of its best responses to some combination of equilibria upstream from its children, then orders its children to choose the action corresponding to that equilibrium. The internal nodes then, on receiving an order from their parent, find choices for their children, to which the choice they were ordered to make is a best response, that lead to equilibria upstream from the children.

Based on this interpretation, it is pretty straightforward to determine how the processing of the algorithm can be distributed among the players so that the players only need to communicate with their neighbours in the tree and need only be aware of their own payoffs. If the processing is distributed in this way, it makes sense to assume that a rational, selfish root node will take advantage of its opportunity to choose actions for both itself and its children to select the equilibrium that maximises its own payoff. Similarly, each node can choose actions for its children that maximise its payoff within the limits of the choices made by its ancestors.

5 Approximation algorithm

5.1 Description of approximation algorithm

This section explains how Kearns et al. modify the abstract algorithm to get an implementable algorithm to calculate approximate Nash equilibria.

Each player is constrained to play a *discretised* mixed strategy that is a multiple of τ. The value of τ is determined by the desired precision (i.e. the amount ϵ by which a player at most may be able to unilaterally improve his payoff). Let $\tau m = 1$ for some $m \in \mathbb{N}$.

By restricting the choices of the players to $\frac{1}{\tau} + 1$ different alternatives $(0, \tau, 2\tau, \ldots, 1)$, we can limit the size of the tables used in TREENASH to $(\frac{1}{\tau} + 1)^2$ entries.

The algorithm to calculate approximate Nash equilibria in tree games, APPROXIMATETREENASH, differs from TREENASH only in the following:

- APPROXIMATETREENASH takes an additional argument ϵ, from which τ is calculated.
- For every vertex U with parent V, T_U^V has entries $T_U^V(v, u)$ only for $0 \leq u, v \leq 1$, $u = i\tau$, $v = j\tau$, $i, j \in \mathbb{N}$.
- Computations of best responses in TREENASH are replaced in APPROXIMATETREENASH by computations of ϵ-best responses. This is done by iterating through all allowed values of w and $\vec{u}$ (if applicable), and finding the ϵ-best responses for V for each combination of w and $\vec{u}$.

5.2 Requirements on τ

We can ensure that ApproximateTreeNash computes ϵ-Nash equilibria correctly (only outputs true ϵ-Nash equilibria, and finds at least one ϵ-Nash equilibrium with a distance of τ (in $L1$ norm) for each exact Nash equilibrium). This requires that:

$$\tau \leq \min(\epsilon/(2^{k+2}(k+\log_2(k))), 2/(k\log_2^2(k/2))). \tag{3}$$

Here, k is the size of the largest neighbourhood in the graph; $k = \max_{i\in G}|N_G(i)|$.

As the most computation-intensive part of the algorithm takes $O(\frac{1}{\tau^{2k}})$ time, and τ is bounded as described in Equation (3), the running time of the algorithm is polynomial in $\frac{1}{\epsilon}$, n and 2^k, the running time is polynomial in the size of the problem.

5.3 Correctness of approximation algorithm

Three things must be proven:

- An ϵ-best response always exists on the grid.
- The end results of the algorithm are ϵ-Nash equilibria.
- For every Nash equilibrium, the algorithm finds an ϵ-Nash equilibrium within a distance of τ (in L_1 norm) from it.

The second statement is based on the proof for the abstract algorithm (and the truth of the first statement); by ensuring that for every player, set of upstream ϵ-equilibria and possible choices by the parent player, the ϵ-best responses for the player are found, no player can improve his result by more than ϵ, and the result is therefore an ϵ-Nash equilibrium.

The other two statements needs some extra mathematics.

5.3.1 Lemma 1

If for all $\vec{p},\vec{q}\in[0,1]^k$, $|p_i-q_i|\leq\tau$ for all $1\leq i\leq k$ and $\tau\leq 2/(k\log_2^2(k/2))$ and $k\geq 2$, then:

$$\left|\prod_{i=1}^{k} p_i - \prod_{i=1}^{k} q_i\right| \leq (2k\log_2 k)\tau. \tag{4}$$

Kearns et al. [2] prove this by induction on k assuming that k is a power of two. The assumption that k is a power of two is said not to lead to a loss of generality, but there is no obvious way to generalise the result for k being a power of two back to k being any positive integer except 1.

The induction step is:

$$\begin{aligned}
\prod_{i=1}^{k} q_i &= \left(\prod_{i=1}^{\frac{k}{2}} q_i\right)\left(\prod_{i=\frac{k}{2}+1}^{k} q_i\right) \\
&\leq \left(\prod_{i=1}^{\frac{k}{2}} p_i + k\left(\log_2\frac{k}{2}\right)\tau\right)\times\left(\prod_{i=\frac{k}{2}+1}^{k} p_i + k\left(\log_2\frac{k}{2}\right)\tau\right) \\
&\leq \left(\prod_{i=1}^{k} p_i\right) + 2k\left(\log_2\frac{k}{2}\right)\tau + \left(k\left(\log_2\frac{k}{2}\right)\tau\right)^2 \\
&= \left(\prod_{i=1}^{k} p_i\right) + 2k((\log_2 k)-1)\tau + \left(k\left(\log_2\frac{k}{2}\right)\tau\right)^2
\end{aligned}$$

$$\leq \left(\prod_{i=1}^{k} p_i\right) + (2k\log_2 k)\tau.$$

And, similarly, with p and q switching places.

The first inequality is based on the lemma being true for $\frac{k}{2}$, the second on $|p_i| \leq 1$ and the last relies on $\tau \leq 2/(k\log_2^2(\frac{k}{2}))$.

To remove the need for the "assumption without loss of generality" described above, the induction step can be generalised to work for odd values of k (which means that the induction proves the lemma for all $k \geq 4$, assuming it has been proven for $k = 2$ and $k = 3$, which is quite straightforward).

$$\begin{aligned}
\prod_{i=1}^{k} q_i &= \left(\prod_{i=1}^{\frac{k-1}{2}} q_i\right)\left(\prod_{i=\frac{k+1}{2}}^{k-1} q_i\right) q_k \\
&\leq \left(\prod_{i=1}^{\frac{k-1}{2}} p_i + (k-1)\left(\log_2 \frac{k-1}{2}\right)\tau\right) \times \left(\prod_{i=\frac{k+1}{2}}^{k-1} p_i + (k-1)\left(\log_2 \frac{k-1}{2}\right)\tau\right) q_k \\
&\leq \left(\prod_{i=1}^{k} p_i\right) + \tau + 2(k-1)\left(\log_2 \frac{k}{2}\right)\tau + \left(k\left(\log_2 \frac{k}{2}\right)\tau\right)^2 \\
&\leq \left(\prod_{i=1}^{k} p_i\right) + 2k\left(\log_2 \frac{k}{2}\right)\tau + \left(k\left(\log_2 \frac{k}{2}\right)\tau\right)^2 \\
&= \left(\prod_{i=1}^{k} p_i\right) + 2k(\log_2 k - 1)\tau + \left(k\left(\log_2 \frac{k}{2}\right)\tau\right)^2 \\
&\leq \left(\prod_{i=1}^{k} p_i\right) + (2k\log_2 k)\tau
\end{aligned}$$

The additional inequalities used here are $2\log_2(k/2) \geq 1$ (true if $k \geq 3$), $q_k \leq 1$ and $q_k \leq p_k + \tau$.

This obviously places the same limitations on τ as the case where k is even, as the final result is the same.

5.3.2 Lemma 2

Using the previous lemma, we can show that if mixed strategies $\vec{p}$ and $\vec{q}$ for $(G, \mathcal{M})$ and τ satisfy the requirements above, $|M_i(\vec{p}) - M_i(\vec{q})| \leq 2^{k+1}(k\log_2 k)\tau$. Specifically:

$$\begin{aligned}
|M_i(\vec{p}) - M_i(\vec{q})| &\leq \sum_{\vec{x}\in\{0,1\}}^{k} \left|\prod_{j=1}^{k} p_j^{1-x_j}(1-p_j)^{x_j} - \prod_{j=1}^{k} q_j^{1-x_j}(1-q_j)^{x_j}\right| |M_i(\vec{x})| \\
&\leq \sum_{\vec{x}\in\{0,1\}}^{k} (2k\log_2 k)\tau \leq 2^{k+1}(k\log_2 k)\tau.
\end{aligned}$$

Here, the first inequality is from Equation (2) (the absolute value of the difference of sums being no greater than the sum of absolute values of differences) and the second uses the previously proven lemma (noting that $x_j \in 0, 1$ and $p_j \in [0, 1]$) and the limit on the payoff.

This means that for every Nash equilibrium $\vec{p}$, moving to the nearest (in L_1 metric) strategy $\vec{q}$ on the τ-grid decreases each player's payoff by at most $2^{k+1}(k\log_2 k)\tau$.

5.3.3 Final proof

Lemma 2 directly shows that there is a $2^{k+1}(k \log k)\tau$-best response near each best response on the τ-grid.

Kearns et al. [2] go on to show that $\vec{q}$ as described above is an ϵ-Nash equilibrium. Let r_i be a best response for player i to $\vec{q}$, and thus $M_i(\vec{q}[i : r_i]) \geq M_i(\vec{q})$.

The last lemma implies that $|M_i(\vec{q}[i : r_i]) - M_i(\vec{p}[i : r_i])| \leq 2^{k+1}(k \log_2 k)\tau$, and because $\vec{p}$ is an equilibrium, $M_i(\vec{p}) \leq M_i(\vec{p}[i : r_i])$. This means that $M_i(\vec{q}[i : r_i] \leq M_i(\vec{p}) + 2^{k+1}(k \log_2 k)\tau$. The same lemma also implies that $M_i(\vec{q}) \geq M_i(\vec{p}) - 2^{k+1}(k \log_2 k)\tau$. Subtracting these two statements from each other, we get:

$$M_i(\vec{q}[i : r_i]) - M_i(\vec{q}) \leq 2^{k+2}(k \log_2 k)\tau.$$

In other words, $\vec{q}$ is a $2^{k+2}(k \log_2 k)\tau$-Nash equilibrium, and so τ must be less than $\epsilon/(2^{k+2}(k + \log_2 k))$.

By choosing $\tau \leq \min(\epsilon/(2^{k+2}(k + \log_2 k)), 2/(k \log_2^2(k/2)))$, both the requirements on τ used in this proof are both valid. This means that for each Nash equilibrium, at least one ϵ-Nash equilibrium is found near it, at a distance of at most τ (in L_1 norm).

5.4 Example

As an example of the execution of ApproximateTreeNash, Kearns et al. [2] have a game with 8 players, each with two actions, which is reproduced here as Figure 3. In this game, the "M" players get a higher payoff by matching their parent's choice, while the "O" players get a higher payoff by choosing the opposite action to their parent. The relative payoff for matching or unmatching is also modulated by the values chosen by the children and also varies between vertices of the same type (in an unspecified way).

When executed with $\tau = 0.01$ and $\epsilon = 0.05$, the downstream pass produces the tables for each vertex and its parent in depth-first order, i.e. 0 and 1 before 4, 2 and 3 before 5, 4 and 5 before 6, and 6 before 7 (the exact order depends on the depth-first search used, so e.g. ascending numerical order is one possible order). Each table is shown graphically next to the corresponding edge, with 1 shown as black and 0 as grey, and the probability of choosing action 0 increasing to the right (the child) and down (the parent).

From this, the upstream pass can calculate an approximate Nash equilibrium starting at the root by choosing any value for which the root's table is 1, and then proceeding down the tree, choosing any witness (i.e. suitable choices for the children) to the values chosen by the parent.

For example, 7 can start off by deciding to play 0, which means that 6 must play 0 (or a close approximation; let us assume that we prefer pure strategies to keep things simple). If 6 chooses 0, we must find values for 4 and 5 that are black in their tables. These are 0 for 4 and 1 for 5 (this can be seen in the tables for 4 and 5 by looking for black segments extending to the bottom of the table, where the chance of 6 choosing 0 is 1). Repeating this process of choosing a witness for nodes 4 and 5, we get 0 for 0, 1 for 1, 1 for 2 and 0 for 3, which gives us the approximate equilibrium $(0, 1, 1, 0, 0, 1, 0, 0)$. Similarly, we can generate equilibria like $(1, 0, 0, 1, 1, 0, 1, 1)$, by choosing 1 for 7 and sticking to pure equilibria. By choosing a mixed strategy for 7, lots of mixed equilibria can be generated.

6 Exact algorithm

This section explains how Kearns et al. modify the abstract algorithm to get an implementable algorithm to calculate exact Nash equilibria.

6.1 Data structures

In the exact algorithm, each table $T_{U_i}^{V}$ passed from U_i to V is represented as a finite union of axis-parallel rectangles (or degenerate variants of these, such as line segments

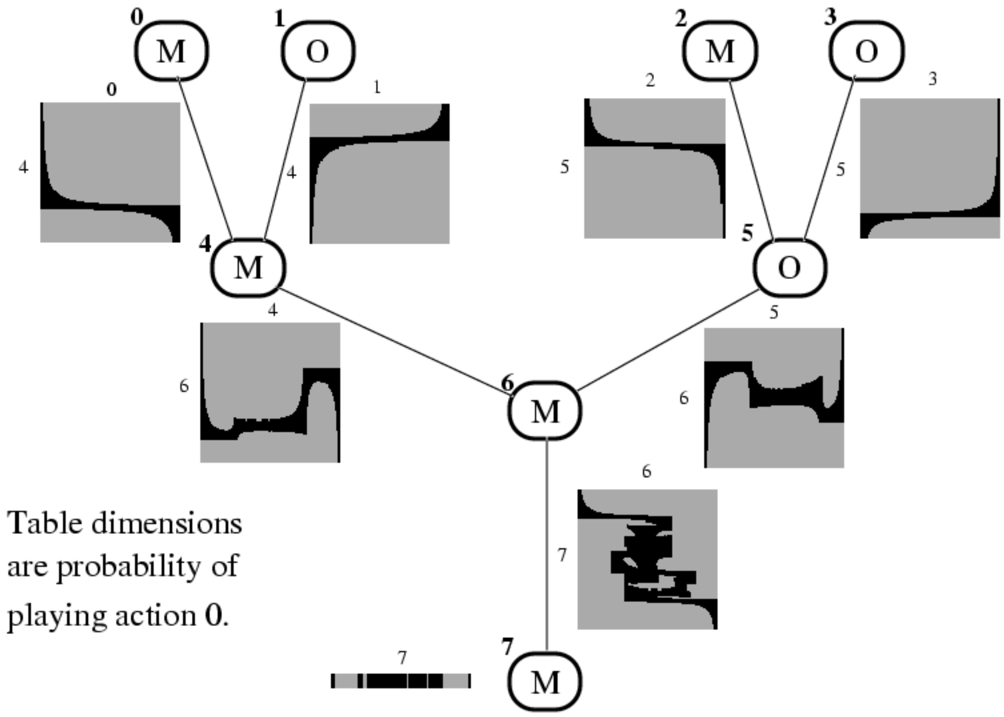

Figure 3: Example game with tables produced by APPROXIMATETREENASH

and points). Each $T_{U_i}^V$ is given by an ordered list called a *v-list*, consisting of $v + 1$ real numbers such that $0 = v_1 \leq v_2 \leq \ldots \leq v_{m-1} \leq v_m = 1$. The v-list splits the mixed strategy v into intervals $[v_l, v_{l+1}]$, where $1 \leq l \leq m$. Each interval and child U_i is associated with a set of (disjoint) intervals $I_j^{i,l} \subseteq [0,1]$. $T_{U_i}^V(v, u_i) = 1$ iff l and j exist such that $v \in [v_l, v_{l+1}]$ and $u_i \in I_j^{i,l}$. Each $[v_l, v_{l+1}]$ and $I_j^{i,l}$ can be considered to represent a rectangle in $T_{U_i}^V$ with the value 1.

By merging the v-lists for all tables of the form $T_{U_i}^V$, we can get a common v-list for all these tables.

The next section will in explaining the computation also show why this representation is sufficient.

6.2 Computation

The computation of T_V^W in the downstream pass is somewhat complicated. It is based on finding the set $\mathcal{W}$ of strategies w for W for which some $v \in [v_l, v_{l+1}]$ is a best response to $\vec{u}$ and w, where $\vec{u}$ ranges over some rectangular region $I_{j_i}^{1,l} \times I_{j_1}^{2,l} \times \cdots \times I_{j_k}^{k,l}$. This is repeated for all v-intervals and possible choices of the indices $j_1, \ldots, j_k$.

In the case where $v_l \neq 0$ and $v_{l+1} \neq 1$, a value in $[v_l, v_{l+1}]$ can be a best response to $\vec{u}$ and w only if the payoff for V playing 0 is identical to the payoff for V playing 1, in which case all of V's strategies are best responses. Thus, T_V^W will be 1 across the region $\mathcal{W} \times [v_l, v_{l+1}]$ and the union of all such subsets of $w \times v$ across all $m-1$ possible choices of v-interval and all choices of indices $j_1, \ldots, j_k$ completely defines where $T_V^W(w, v) = 1$.

Let $M_V(v, \vec{u}, w)$ be the expected payoff to V under mixed strategies v for V, $\vec{u}$ for $\vec{U}$ and w for W, and $\Delta(\vec{u}, w) = M_V(0, \vec{u}, w) - M_V(1, \vec{u}, w)$. Let $I_1, \ldots, I_k$ each be continuous intervals in $[0,1]$ and $\mathcal{W} = \{w \in [0,1] : \exists \vec{u} \in I_1 \times \cdots \times I_k : \Delta(\vec{u}, w) = 0\}$. From Equation (2) we get:

$$\Delta(\vec{u}, w) = \sum_{\substack{\vec{x}\in\{0,1\}^k \\ y\in\{0,1\}}} (M_V(0, \vec{x}, y) - M_V(1, \vec{x}, y)) \left(w^{1-y}(1-w)^y \prod_{i=1}^{k} \left((u_i)^{1-x_i}(1-u_i)^{x_i}\right) \right). \tag{5}$$

As each x_i is always 0 or 1 (for each i), each term of the sum above includes only either u_i or $1 - u_i$, meaning that $\Delta(\vec{u}, w)$ is a linear function of u_i, and therefore a monotonic function of u_i.

$\Delta(\vec{u}, w)$ is also continuous in w, so $w \in \mathcal{W}$ iff $w \in \mathcal{W}_\geq \cap \mathcal{W}_\leq$, where:

$$\mathcal{W}_\geq = \{w \in [0,1] : \exists \vec{u} \in I_1 \times \cdots \times I_k : \Delta(\vec{u}, w) \geq 0\},$$

$$\mathcal{W}_\leq = \{w \in [0,1] : \exists \vec{u} \in I_1 \times \cdots \times I_k : \Delta(\vec{u}, w) \leq 0\}.$$

$w \in \mathcal{W}_\geq$ iff $\max_{\vec{u}\in I_1\times\cdots\times I_k} \Delta(\vec{u}, w) \geq 0$. Since $\Delta(\vec{u}, w)$ is monotonic of each u_i, this maximum must occur at one of the 2^k extremal points (vertices) of the region $I_1 \times \cdots \times I_k$. By denoting $I_j = [l_j, r_j]$ and defining the extremal set $E = \{l_1, r_1\} \times \cdots \times \{l_k, r_k\}$, we get:

$$W_\geq = \bigcup_{\vec{u}\in E} \{w : \Delta(\vec{u}, w) \geq 0\}. \tag{6}$$

The linearity of $\Delta(\vec{u}, w)$ means that $\{w : \Delta(\vec{u}, w) \geq 0\}$ (for any fixed $\vec{u}$) must be of the form $[0, x]$ or $[x, 1]$, and so $W_\geq$ (and $W_\leq$ as well, as the same reasoning applies to it) is either empty, an interval, or the union of two intervals (all intervals of the form $[0, x]$ can be combined to one, as well as all intervals of the form $[x, 1]$). This also applies to $\mathcal{W} = \mathcal{W}_\geq \cap \mathcal{W}_\leq$.

The cases where $v = 0$ or $v = 1$ are simpler, as $v = 0$ is a best response if $\Delta(v, \vec{u}, w) \geq 0$ and $v = 1$ is a best response if $\Delta(v, \vec{u}, w) \leq 0$. In other words, we want $\mathcal{W}_\geq$ and $\mathcal{W}_\leq$ respectively.

In other words, we can compute $\mathcal{W}$ as described above in time exponential in k by exhaustively searching the extremal set E. Also, each rectangular region splits into at most 2 rectangular regions.

Additionally, we can use similar reasoning to show that the tables at the leaves have at most 3 rectangular regions (we can have one interval of w where $v = 0$ is the best response, one region where $v = 1$ is the best response, and if both of these exist, they are separated by a value of w where v does not affect the payoff; to prove this, note that Δ for leaf nodes is just a linear function of w). This limits the number of rectangular regions in T_U^V to $2^{a_U}3^{b_U}$, where a_U is the amount of internal nodes and b_U the amount of leaf nodes in the subtree rooted at U.

The witness lists need not be stored, as they can be generated from the tables. If $T(w, v) = 1$, we need merely find a $\vec{u}$ such that $T(v, u_i) = 1$ for all i.

6.3 Example

In Figure 4, which is reproduced from Kearns et al. [2], the table generated by the exact algorithm for the game used in the example for the approximation algorithm for the edge between vertices 6 and 7. Here, values where the approximate table is 1 are shown in grey and values where the exact table is 1 are shown in black.

7 Generalisations and extensions

Kearns et al. [2] have verified the following generalisations for the approximation algorithm, and were at the time of writing [2] working on generalising their results to the exact algorithm.

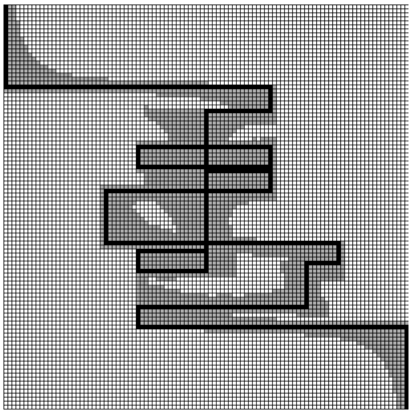

Figure 4: Example of table produced by ExactTreeNash

7.1 Multiple actions

By representing the mixed strategy v for each player V as a vector $\vec{v} = (v_0, v_1, \ldots, v_{c-2})$, where c is the amount of possible actions for player V (each entry u_i in the vector represents the probability of V playing action i, and the probability of playing action $c-1$ is determined by subtracting the sum of the probabilities of the other actions from 1), the approximation algorithm can be extended to handle multiple actions.

7.2 Vertex merging

The extension to multiple actions allows the algorithm to be further extended to arbitrary (i.e. also non-tree) graphs. Vertices are merged (in some manner) until the graph becomes a tree. The players in the merged vertices must be playing approximate best responses to each other as well as to their upstream and downstream neighbours.

As the running time is exponentially dependent on the size of the merged vertices, minimising this cluster size is important for the efficiency of the algorithm.

With these two extensions the approximation algorithm can be used to calculate approximate Nash equilibria for any single-stage multiplayer game.

7.3 Special equilibria

By appropriately augmenting the tables used in the downstream pass of the algorithm, Nash equilibria can be found that have certain special properties, such as:

- *Player optimum*: The expected reward to a specific player is maximised. In fact, this can be done without auxiliary data structures by defining the player whose reward should be maximised as the root of the tree, and selecting actions for him and his children that maximise his payoff.
- *Social optimum*: The total expected reward is maximised.
- *Welfare optimum*: The minimum expected reward (among all the players) is maximised.

8 Applications of the algorithms

As noted earlier, the algorithms described here can be generalised to find (approximate) Nash equilibria in any single-stage multiplayer game, and do so reasonably efficiently in the case of games with sparse graphical structures.

The algorithms are particularly efficient for games that can be expressed as trees with little or no vertex merging. Also, if the game can be expressed as a tree, the only communication required is between neighbours in the tree. This means that these algorithms are useful when trying to find stable solutions to games where communications and interaction are very limited, preferably to a tree-like structure.

For example, the game could be about allocation of funds in an organisation (which probably has a inherent tree structure), with each department represented as a player. Each player's set of actions would then be the set of possible distributions of the funds between different tasks and/or subdepartments. The "order-giving" aspect of the upstream pass would then amount to the director of each department giving orders to his subordinates, and by defining the payoff for each department as the total expected profit generated by the department (or some other suitable function for non-profit organisations), finding an equilibrium that maximises the payoff to the root would imply finding the best division of funds such that no department by deviating from the plan could increase their profit at the expense of another. The problem here is generating the payoff matrices in the first place; this involves estimating gains from many different plans.

Nash equilibria are generally speaking useful in resource allocation problems, but these seldom exhibit the tree-like structure necessary for efficient use of the aforementioned algorithms. One could imagine a set of processes on a large computing system (massively parallel or distributed), each trying to complete as quickly as possible, playing a game against other processes, interacting directly with the processes with which it has a usable resource in common. This graph should very sparse and have small clusters as processes with similar requirements would be competing (mostly) for the same resources.

A similar application that may have global importance in the not-too-distant future is the "Worldwide Computer" described in a similarly titled article by Anderson and Kubiatowicz in the March 2002 issue of Scientific American [1]. Here, each user of the system can be considered a player, his payoff a function of how quickly his processing job, download or whatever is completed, his possible actions which combination of servers to use, and his neighbours in the game graph the other (potential) users of the servers he can use to complete his job. Here, some sort of social optimum equilibrium is probably the most desirable, although the question remains of how to weight the importance of different players. Possible models include weighting by size of contribution to the system in resources, or by some form of "priority credit" which the user is allowed to distribute himself among the jobs he runs on the system.

9 Conclusion

The graphical game models and algorithms to find Nash equilibria in them described here are quite efficient. In the future, they may be applied to resource allocation problems in networks with large amounts of independent services and users to maximise efficiency while preventing users from hogging resources.

References

[1] D.P. Anderson, J. Kubiatowicz: The Worldwide Computer, Scientific American, March 2002, Volume 286, Number 3, 28-35, ISSN 0036-8733.

[2] M. Kearns, M.L. Littman, S. Singh: Graphical Models for Game Theory, Proceedings of UAI 2001, 253-260.

[3] J.F. Nash: Non-cooperative games, Annals of Mathematics, 54, 286-295, 1951.