A Polynomial-time Nash Equilibrium Algorithm for Repeated Games*

Michael L. Littman
Dept. of Computer Science
Rutgers University
Piscataway, NJ 08854-8019 USA
mlittman@cs.rutgers.edu

Peter Stone
Dept. of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188 USA
pstone@cs.utexas.edu

ABSTRACT

With the increasing reliance on game theory as a foundation for auctions and electronic commerce, efficient algorithms for computing equilibria in multiplayer general-sum games are of great theoretical and practical interest. The computational complexity of finding a Nash equilibrium for a one-shot bimatrix game is a well known open problem. This paper treats a closely related problem, that of finding a Nash equilibrium for an average-payoff *repeated* bimatrix game, and presents a polynomial-time algorithm. Our approach draws on the "folk theorem" from game theory and shows how finite-state equilibrium strategies can be found efficiently and expressed succinctly.

Categories and Subject Descriptors

F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Algorithms, Theory

Keywords

Repeated games, complexity analysis, Nash equilibrium, computational game theory

1. INTRODUCTION

The Nash equilibrium is one of the most important concepts in game theory, forming the basis of much recent work in multiagent decision making and electronic marketplaces. As such, efficiently computing Nash equilibria is one of the most important problems in computational game theory.

The central result of this paper is a polynomial-time algorithm for computing a Nash equilibrium for repeated 2-player (bimatrix) games, under the average-payoff criterion. This result stands in contrast to the problem of computing a Nash equilibrium in a one-shot game, the complexity of which remains an important and long-standing open problem (Papadimitriou 2001). The idea behind our algorithm echoes that of the well known "folk theorem" (Osborne and Rubinstein 1994), which shows how the notion of *threats* can stabilize a wide range of payoff profiles in repeated games. The contribution of this paper is to show how the threat idea can be used to create an efficient equilibrium-finding algorithm.

In the rest of the paper, we formally describe the problem (Section 2) and our algorithm for solving it (Section 3), and conclude with a set of illustrative examples (Section 4).

2. PROBLEM STATEMENT

A repeated bimatrix game is played by two players, 1 and 2, each with a set of action choices of size n^1 and n^2, respectively. The game is played in rounds, with the two players simultaneously making a choice of action at each round. If Player 1 chooses action $1 \leq i^1 \leq n^1$ and Player 2 chooses $1 \leq i^2 \leq n^2$, they receive payoffs of $P^1_{i^1 i^2}$ and $P^2_{i^2 i^1}$, respectively[1]. In a repeated game, players select their actions, possibly stochastically, via a *strategy*—a function of the history of their interactions.

The objective of each player in a repeated game is to adopt a strategy that maximizes its expected average payoff (limit of the means criterion). A pair of strategies is a *Nash equilibrium* if each strategy is optimized with respect to the other—neither player can improve its average payoff by changing strategies unilaterally (Nash 1951).

This paper considers the following computational problem. Given a game specified by payoff matrices P^1 and P^2, return a pair of strategies that constitutes a Nash equilibrium for the average-payoff repeated bimatrix game. The running time of the algorithm should be a polynomial function of the size of the input.

To fully specify the equilibrium-computation problem, we must be concrete about the input and output representations. The input representation is relatively straightforward. For $(p, q) \in \{(1, 2), (2, 1)\}$, the function P^p is an $n^p \times n^q$ matrix. To bound the size of the numbers in these matrices, we assume they are rational numbers, specified as integer numerator and natural denominator of no more than k bits.

*

EC'03, June 9–12, 2003, San Diego, California, USA.
Copyright 2003 ACM 1-58113-679-X/03/0006 ...$5.00.

[1] For cleanliness of notation, we deviate from common practice and write matrices so that a player always chooses the row of its own payoff matrix, while the opponent always chooses the column.

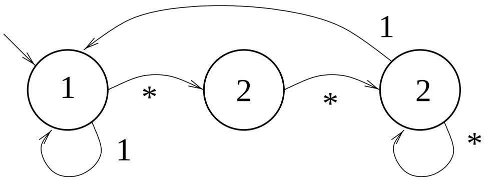

Figure 1: A strategy expressed as a three-state finite-state machine.

So, the running time of our algorithm needs to be a polynomial function of n^1, n^2, and k.

Note that the representation size of an integer is roughly its logarithm in base two and the representation size of a rational number is the sum of the sizes of its numerator and denominator. A *polynomial-size number* is one with representation size bounded by a polynomial function of the input size. Multiplying, dividing, adding or subtracting two polynomial-size rational numbers produces a polynomial-size result, as does solving a polynomial-size system of linear equations or linear program (Schrijver 1986).

The output of an equilibrium computation is a pair of strategies. It is well known that every bimatrix game has at least one pair of strategies that is a Nash equilibrium. However, strategies in repeated games can be infinitely large objects, so it is necessary to use some finite representation for strategies when computing Nash equilibria. In this paper, we consider two strategy representations, classical finite-state machines and a counting-node extension in which actions can be repeated a prespecified number of times. Both represent finite-state strategies, but the counting-node machine can result in exponentially smaller representations, as described next.

A finite-state machine strategy for a player p against an opponent q is a labeled directed graph. One node of the graph is the designated starting node. Each node of the graph is labeled with a probability distribution over action choices for p. Outgoing edges are labeled with actions for q, with no two edges from a single node sharing the same label; in particular, transitions are not influenced by the player's own actions. One outgoing edge for each node is labeled "*" to designate a default edge, taken if player q's action does not match any of the other labels. The size of a finite-state machine strategy is roughly the sum of the nodes and edges in the graph.

Figure 1 illustrates an example finite-state machine strategy for a (2×2)-action game. The player p starts off at the left node and selects Action 1. Then, if the opponent q selects Action 1, p returns to the left node to continue choosing Action 1. However, on any other action choice for q, a transition is made to the middle node, where Action 2 is chosen. Following this, any choice for q results in a transition to the rightmost node, in which Action 2 continues to be chosen until q chooses Action 1. At this point, p returns to the left node again.

The strategy expressed in the figure is "two tits for a tat" in prisoner's dilemma (Axelrod 1984); if Action 1 is cooperate and Action 2 is defect, the player defects twice in response to defection, but otherwise cooperates.

While finite-state machines provide a simple and broad language for expressing strategies, some basic strategies become cumbersome to write down as finite-state machines. Consider, for example, "2^b tits for a tat". A finite-state machine representation requires a graph with 2^b nodes and therefore an exponentially large representation. We introduce a *counting node* into the representation to make it easier to express simple repetitions of this type. Note that the exact form of this extension was selected to be sufficient for our algorithm in Section 3; more general and elegant extensions are also possible.

A counting node is depicted by a double circle with a repeat count c written beneath it. Like a standard node, it is labeled with a probability distribution over actions. This distribution is used independently to select actions c times consecutively. The counting node includes a default outgoing edge, which is taken after the c repetitions are complete. However, a counting node can also have up to one other edge, labeled with one of the opponent's actions, i^q. This edge is taken if and only if the opponent selected i^q all c times while the player was executing its actions in the counting node. The size of a counting-node machine strategy is the sum of the nodes and edges in the graph plus the number of bits in all the repeat counts.

Figure 2 shows an example of a generic counting node and its equivalent representation as a set of nodes in a finite-state machine.

Counting nodes can be used to express "2^b tits for a tat" by replacing the center node in Figure 1 with a counting node with a repeat count of $2^b - 1$. Since the number 2^b can be represented easily using b bits, the strategy can be expressed via a counting-node machine with size $\Theta(b)$. This succinctness is important in the construction in our algorithm in the next section.

3. ALGORITHM DESCRIPTION

Zero-sum games (von Neumann and Morgenstern 1947) are those in which $P^1 + (P^2)^T = 0$. Equilibria in these games can be computed in polynomial time via linear programming and the resulting equilibrium strategies and the expected payoffs to the two players are polynomial-size numbers (Schrijver 1986). An equilibrium strategy in a zero-sum game can be seen as simultaneously a defensive strategy that maximizes a player's expected payoff in the face of the least desirable strategy of the opponent, and also an attack strategy that forces the opponent to its minimum expected payoff.

While an efficient algorithm for zero-sum games appears to handle a zero-measure set of the possible games, we can use this efficient computation as an important component of an algorithm for general-sum games. At the highest level, we can quickly find that either the game is essentially a zero-sum game *or* that there are strategies that both players can play to achieve a better average payoff than what they can guarantee themselves in the worst case, stabilized by the threat of being forced to this lower payoff.

In the rest of this section, we describe the computation, argue that it can be carried out in polynomial time, and prove that the resulting strategies constitute a Nash equilibrium.

3.1 Attack and Defensive Strategies

Once again, let $(p, q) \in \{(1, 2), (2, 1)\}$. We will use $1 \le i^p \le n^p$ and $1 \le i^q \le n^q$ to represent action choices for the two players and π^p and π^q to be probability distributions

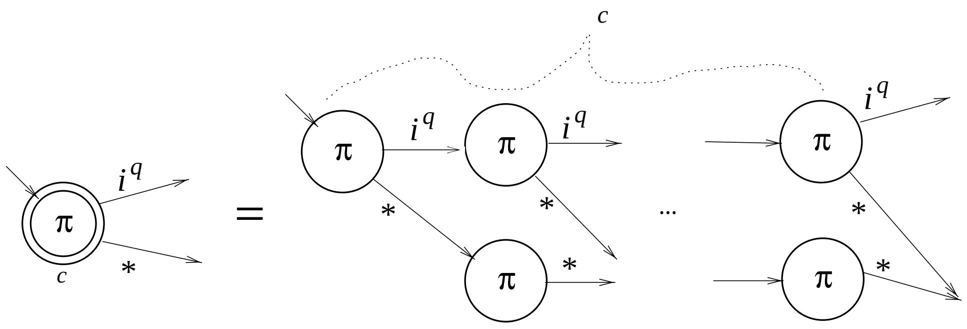

Figure 2: A generic counting node and an equivalent finite-state machine.

(vectors) over these action choices. We denote the probability of i^p in π^p as $\pi^p_{i^p}$. Define

$$\alpha^p = \operatorname*{argmin}_{\pi^p} \max_{i^q} \sum_{i^p} P^q_{i^q i^p} \pi^p_{i^p}, \tag{1}$$

and

$$\delta^p = \operatorname*{argmax}_{\pi^p} \min_{i^q} \sum_{i^p} P^p_{i^p i^q} \pi^p_{i^p}. \tag{2}$$

Here, α^p is the equilibrium strategy for p in the zero-sum game defined by p "attacking" q—minimizing q's expected payoff[2]. Similarly, δ^p is the equilibrium strategy for p in the zero-sum game defined by p "defending" against q—maximizing its expected payoff in the worst case. As we mentioned, α^p and δ^p can be computed in polynomial time using linear programming and the results can be expressed as polynomial-size numbers. They are also interchangeable in zero-sum games.

For convenience, define

$$g^p = \sum_{i^p} \sum_{i^q} \delta^p_{i^p} \alpha^q_{i^q} P^p_{i^p i^q}, \tag{3}$$

the expected payoff that p can guarantee itself by playing its defensive strategy. There is no Nash equilibrium in which p receives average payoff less than g^p, since p could always switch to δ^p and do better. In addition, we can define $A^p = P^p - g^p$ as the matrix of *advantages*—for each pair of action choices, how much more will p's payoff be than the value of its defensive strategy? Note that A^p and A^q can be interpreted as a bimatrix game that has several convenient properties:

- The advantage game defined by A^p and A^q has the same set of Nash equilibria as the original game defined by P^p and P^q, since average payoffs in the advantage game are simply shifted down by a constant without changing the relative ordering of payoffs.

- The attack and defensive strategies defined for the original game serve the same purposes in the advantage game for the reason listed in the previous point.

- The defensive strategies in the advantage game achieve expected payoffs of zero.

- The payoffs in the advantage game are polynomial-size rational numbers, so the overall complexity of working with the advantage game instead of the original game does not increase more than polynomially.

In what follows, we will focus on the advantage game, since it has the same equilibria and roughly the same representation size, and lets us rule out average payoffs that are negative when searching for equilibria.

3.2 The Mutual Advantage Case

To find a Nash equilibrium, we consider two cases. In this section, we show how to recognize the case in which it is possible for both players to achieve positive advantage. In the next section, we show how a Nash equilibrium can be constructed in this case. In the section following, the case in which mutual advantage cannot be achieved is treated.

When Player 1 chooses i^1 and Player 2 chooses i^2, the payoffs for the two players can be visualized as a point $x = (A^1_{i^1 i^2}, A^2_{i^2 i^1}) = (x^1, x^2)$ in a two-dimensional space. Following Nash (1950), we consider the set of all pairs of actions for the two players, $X = \{(A^1_{i^1 i^2}, A^2_{i^2 i^1}) | 1 \leq i^1 \leq n^1, 1 \leq i^2 \leq n^2\}$. All the points $x \in X$ can be achieved as average payoffs for the two players in the repeated game, simply by repeatedly playing the corresponding action pair.

The *convex hull* of a set of points is the set of points that can be formed by a linear combination of the points in the set, where the linear coefficients are positive and sum to one (a convex combination). For the purposes of this paper, we limit our attention to polynomial-size coefficients. Any point in the convex hull of the points in X can be achieved as average payoffs for the two players in the repeated game. Specifically, for any point u in the convex hull of X, there is a set of weights w such that $u = \sum_{x \in X} w_x x$. Consider integer weights w' derived by multiplying w by the least common denominator of its components. A pair of strategies that repeats any sequence that includes, for each $x \in X$, each action pair corresponding to x a total of w'_x times, achieves the payoff u on average.

The folk theorem tells us that any u simultaneously in the convex hull of X *and* in the first quadrant (positive advantage for both players) is the payoff of a Nash equilibrium in the repeated game. Typically there are many such points, any of which would be sufficient for our algorithm. However, the work of Nash (1950) justifies a particular selection for a point in the first quadrant via an axiomatic analysis of bargaining. It shows that a very sensible payoff point for

[2] Note that the optimization over q's action choice is deterministic; once π^p is selected, q's optimal choice can be made deterministically.

the two players to accept is one that maximizes the product of their advantages. We next show how this point can be identified efficiently.

Note that the point that maximizes the product of the advantages will be found on the outer boundary of the convex hull—the product of the advantages of any internal point can be increased by moving to a point above it and to the right (Nash 1950). This implies that that the point can be expressed by a weight vector w that has non-zero weight on only one or two $x \in X$, since the convex hull is a two-dimensional polygonal region bounded by line segments.

Next, let $x, y \in X$ be two points formed by two different action pairs, each with at least one component non-negative. We want to find a point z on the edge between x and y, $z = w_x x + (1 - w_x)y$ for $0 \leq w_x \leq 1$, such that the product of the advantages, $z^1 z^2$, is maximized. Setting the derivative of this product to zero and solving for w_x, we find that the product is maximized when

$$w_x = \frac{-y^2(x^1 - y^1) - y^1(x^2 - y^2)}{2(x^2 - y^2)(x^1 - y^1)}. \qquad (4)$$

If $w_x < 0$ or $w_x > 1$, then the maximum product is achieved at an endpoint.

We can now make the following observations:

- We are trying to identify a payoff in the convex hull of X in the first quadrant (both components positive) that maximizes the product of the components. We argued that such a point is either in X or on a line segment between two points in X. Therefore, our search can be limited to examining pairs of points in X.

- If a point $x \in X$ is in the third quadrant (both components negative), it need not be considered. To produce a point in the first quadrant, x would have to be paired with a point $y \in X$ in the first quadrant, and the point y would dominate any combination it could make with x: $w_x = 0$.

- Points in the second or fourth quadrant (one positive, one negative component) need to be considered, since they could possibly be combined with another point and result in a point with a larger product of advantages than either endpoint. For example, for $x = (3, -1)$, $y = (1, 5)$, Equation 4 results in $w_x = 1/6$, or $z = (4/3, 4)$ with a product of advantages of $16/3$.

- Evaluating Equation 4 results in a weight that is a polynomial-size rational number, since it involves a constant number of basic arithmetic operations.

- If the weight computed for points x and y is between 0 and 1, let the weight be represented by the rational number $\frac{r}{s}$ for integers $0 \leq r \leq s$. A pair of strategies that repeats any sequence that includes the action pair associated with y a total of $s - r$ times and x a total of r times achieves an average payoff whose product of advantages is maximized over all convex combinations of x and y.

- By looping through all points and pairs of points in X (a polynomial number $n^1 n^2 + (n^1)^2 (n^2)^2$ of combinations), ignoring any points with mutually negative advantages, computing weights, and checking which has the largest product of advantages, we can identify a pair of action pairs i^1, i^2 and j^1, j^2 and (polynomial-size) integers r_i and r_j. A pair of strategies that repeats any sequence that includes the action pair (i^1, i^2) a total of r_i times and (j^1, j^2) a total of r_j times maximizes the product of advantages over all achievable payoffs. One such pair of strategies is illustrated in Figure 3, in which the action choices are made consecutively. Note that if the product of advantages is maximized for a point $x \in X$, one-state strategies that repeat the actions associated with x suffice.

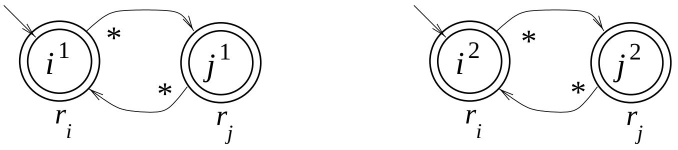

Figure 3: A pair of strategies that maximizes the product of advantages.

- The procedure executes in polynomial time and produces a polynomial-size output.

The next section shows how to combine the strategies in Figure 3 with attack strategies from Section 3.1 to produce a Nash equilibrium.

3.3 Punishment

The strategies in Figure 3 result in average advantages of

$$z = (r_i A^1_{i^1 i^2} + r_j A^1_{j^1 j^2}, r_i A^2_{i^2 i^1} + r_j A^2_{j^2 j^1})/(r_i + r_j). \qquad (5)$$

These strategies maximize the product of advantages over all strategies, so we say they represent *mutual cooperation*. However, they are not necessarily in equilibrium, as either player might have an incentive to select a different action that results in higher payoff at the expense of the opponent; we call any such strategy *defection*. We now show how to modify the cooperative strategies so that each player threatens the other with worse payoff for failing to cooperate, thus eliminating the incentive for defection.

Figure 4 illustrates strategies that build on the strategies in Figure 3 to include the threat of punishment if a player doesn't cooperate. Here, α^1 and α^2 are the attack strategies defined in Section 3.1. In words, the player p executes its cooperative actions for $r_i + r_j$ steps. If, at any point, the opponent defects by deviating from its assigned actions, the player enters a punishment phase for a^p steps at the end of the cooperative phase. When the punishment is complete, a new cooperative phase begins[3]. Once again, the structure of these strategies can be simplified in games in which the product of advantages is maximized for a point $x \in X$.

The only, as yet, undefined quantity in the strategies just described is the number of punishment steps for the two players. We next show how to select values for a^1 and a^2 so that mutual cooperation becomes a best response.

Let $(p, q) \in \{(1, 2), (2, 1)\}$. When faced with a strategy for p of the form given in Figure 4, the average payoff of

[3] Note that delaying punishment until the end of the cooperative phase is not necessary and is probably wasteful. We use this approach simply because it simplifies our presentation and is sufficient to prove our result.

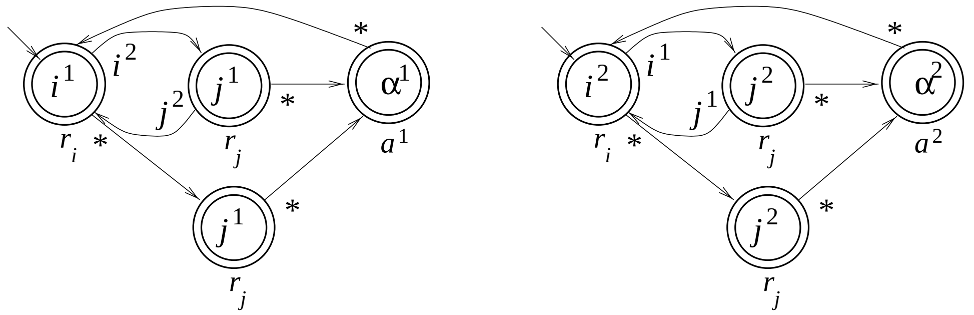

Figure 4: A pair of strategies that is a Nash equilibrium.

Player q for cooperating is

$$z^q = (r_i A^q_{i^q i^p} + r_j A^q_{j^q j^p})/(r_i + r_j), \tag{6}$$

much like in Equation 5. Let

$$d^q = \max_{x \in X} x^q, \tag{7}$$

which is an upper bound on the largest possible value that q can get in a single round by defecting. The average payoff to q in the advantage game for using a strategy that defects cannot be larger than $((r_i + r_j)d^q)/(r_i + r_j + a^p)$, since q can do no better than d^q on each cooperation round and 0 on each punishment round. In fact, this bound is exceedingly loose, as d^q is probably not achievable in most games. Nevertheless, we use this expression since it simplifies presentation.

We want to know how to set a^p so that cooperation is a best response: $z^q > ((r_i + r_j)d^q)/(r_i + r_j + a^p)$, or equivalently, $a^p z^q > (r_i + r_j)(d^q - z^q)$. Note that $z^q > 0$ since we specifically chose a point that has positive advantage. Similarly, $d^q > z^q$ because q's payoff for defection must exceed the payoff for cooperating or defection will not be considered. Solving for a^p and rounding up, we get

$$a^p = \lceil (r_i + r_j)(d^q - z^q)/z^q \rceil. \tag{8}$$

Punishing for a^p rounds provides a sufficient deterrent to discourage q's defection. Once again, since this expression uses only basic arithmetic operations on polynomial-size numbers, the result is a polynomial-size number. Therefore, in the case in which both players can achieve positive advantage, the counting-node machines illustrated in Figure 4 constitute a Nash equilibrium and have polynomial size.

Recall that the preceding rested on the assumption that there are strategies for which both players have positive advantage. But, mutual advantage is not always possible, for example, if none of the convex hull is strictly in the first quadrant. The next section completes the presentation of our algorithm by providing Nash equilibrium strategies in the case where positive advantages cannot be achieved.

3.4 Complete Strategy

The previous two subsections dealt with the case in which it is possible for both players to receive average payoffs better than what they can guarantee themselves using a defensive strategy. If no such strategies exist, then we must use a different tack to find an equilibrium. Let v^p be the largest payoff that p can achieve against the defensive strategy of its opponent in the advantage game:

$$v^p = \max_{i^p} \sum_{i^q} A^p_{i^p i^q} \delta^q \tag{9}$$

Note that it can't be the case that both $v^1 > 0$ and $v^2 > 0$; it would imply we are in the mutual advantage case. In addition, neither $v^1 < 0$ nor $v^2 < 0$ (Player p can guarantee itself at least 0 with a best response). This leaves two subcases: $v^1 = 0$ and $v^2 = 0$, or $v^p > 0$ and $v^q = 0$ for some $(p, q) \in \{(1, 2), (2, 1)\}$. In the first subcase, indefinitely repeating δ^1 and δ^2 is a Nash equilibrium (the game is essentially zero sum). In the second subcase, let h^p be p's best response against δ^q,

$$h^p = \operatorname{argmax}_{i^p} \sum_{i^q} A^p_{i^p i^q} \delta^q. \tag{10}$$

Note that δ^q must also be a best response to h^p. Why? Any better response to h^p would imply a payoff for q greater than 0 (perhaps at the expense of p), which would violate the assumption that there is no strategy pair with mutual advantage[4]. Therefore, playing h_p against δ^q must be a Nash equilibrium.

This completes our algorithm, which we summarize below.

1. Begin with a game specified by matrices P^1 and P^2.

2. Compute attack strategies α^1 and α^2 (Equation 1) and the values g^1 and g^2 the players can guarantee themselves (Equation 3).

3. Compute advantage matrices $A^1 = P^1 - g^1$ and $A^2 = P^2 - g^2$.

4. Loop through all pairs of actions and all pairs of pairs of actions to find a cooperation point that maximizes the product of advantages (Section 3.2).

[4] Playing h^p against δ^q results in the payoff point $(v^p, 0)$ with $v^p > 0$. Let h'^q be a best response to h^p. Playing h^p against h'^q results in the payoff point (v'^p, v'^q). Assume $v'^q > 0$. It must be that $v'^p < 0$, otherwise we have a strategy pair that is mutually advantageous. Now, consider $\epsilon = v^p/(v^p - v'^p)$. By the inequalities already stated, $0 < \epsilon < 1$ (v^p is positive, v'^p negative). Moving a fraction $\epsilon/2$ from the payoffs of h^p against δ^q to h^p against h'^q results in payoffs of $v^p/2 > 0$ for p and $v'^q \epsilon/2 > 0$ for q. This is impossible, since we could then construct a pair of strategies that is mutually advantageous. Therefore, the assumption that $v'^q > 0$ is invalid, implying that δ^q is a best response to h^p.

5. If a cooperation point is found with positive advantages, we have a pair of action pairs (i^1, i^2) and (j^1, j^2) and repeat counts r_i and r_j. Compute the average payoffs for cooperating z^1 and z^2 (Equation 6), and a bound on the maximum defection payoff d^1 and d^2 (Equation 7), and use these to compute a^1 and a^2, the number of punishments needed to stabilize cooperation (Equation 8). Output strategies as shown in Figure 4.

6. If no cooperation point is found with positive advantages, compute defensive strategies δ^1 and δ^2 (Equation 2). Compute the best responses h^1 and h^2 to the defensive strategies (Equation 10) and the players' expected payoffs v^1 and v^2 for adopting them (Equation 9). If $v^1 = 0$ and $v^2 = 0$, output δ^1 and δ^2 as a Nash equilibrium. If $v^1 > 0$ and $v^2 = 0$, output h^1 and δ^2 as a Nash equilibrium. If $v^1 = 0$ and $v^2 > 0$, output δ^1 and h^2 as a Nash equilibrium.

We have argued that each of these operations can be implemented to execute in polynomial time and produces polynomial-size output, and that the cases considered are exhaustive and correct.

3.5 Symmetric Games

A symmetric game is one in which $P^1 = P^2$, so both players face the same strategic situation. In human-constructed games, this situation is very common as it embodies the notion of a fair game.

The argument in Section 3.2 becomes much simpler in the symmetric case. First, note that the convex hull of the points in X will be symmetric about the line through the origin with slope one. The product of advantages will be maximized by a point on this line, which implies that it is only necessary to check pairs of points such that $x^1 = y^2$ and $x^2 = y^1$. In this case, Equation 4 simplifies to $w_x = \frac{1}{2}$, so the action pair with the largest average payoff for the two players determines the point to be considered.

For symmetric games, our algorithm can be simplified as follows.

1. Begin with a game specified by matrix P^1 (P^2 is the same).

2. Compute an attack strategy α^1 (Equation 1) and the value g^1 players can guarantee themselves (Equation 3).

3. Compute advantage matrix $A^1 = P^1 - g^1$.

4. Loop through all pairs of actions i, j to find the pair that maximizes $z^1 = (A^1_{ij} + A^1_{ji})/2$.

5. If $z^1 > 0$, and $i \neq j$, use $i^1 = j^2 = i$, $i^2 = j^1 = j$, $r_i = r_j = 1$, compute a bound on the maximum defection payoff d^1 (Equation 7), and use it to compute $a^1 = \lceil 2(d^1 - z^1)/z^1 \rceil$, the number of punishments needed to stabilize cooperation. Output strategies as shown in Figure 4 (the cooperation nodes no longer need to be counting nodes). On the other hand, if $i = j$, a simpler equilibrium strategy is possible with a single node that repeats action i as long as the opponent chooses the same action, then punishes for $a^1 = \lceil (d^1 - z^1)/z^1 \rceil$ times on a defection.

6. If $z^1 = 0$, compute defensive strategy δ^1 (Equation 2) and output the pair of strategies (δ^1, δ^1) as a Nash equilibrium.

Note that the cooperative phase in the symmetric case is never more than two states.

4. EXAMPLES

This section provides several concrete examples of games to help illustrate the ideas from the previous sections.

The *prisoner's dilemma* game is defined by $P^1 = P^2 = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$. This game is symmetric, so the simplified analysis in Section 3.5 applies. The attack strategy is Action 2 (defect) making the guaranteed value $g = 1$. The pair of actions $(i, j) = (1, 1)$ maximizes the average payoff of $z = 3$. Maximum defection payoff is bounded by $d = 5$. Since $i = j$, the formula for the number of punishment rounds computes $a = 1$. Thus, the equilibrium strategy computed by our algorithm is precisely tit-for-tat, the winning strategy in the prisoner's dilemma tournament described by Axelrod (1984).

The *battle of the sexes* game is defined by $P^1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $P^2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$. This game is not symmetric (although it is fair). The attack strategy for Player 1 is $(1/3, 2/3)$ and for Player 2 is $(2/3, 1/3)$. Players can guarantee themselves $g = (2/3, 2/3)$ by playing defensive strategies, which are obtained by reversing the probabilities in the attack strategies. Based on these calculations, the advantage game is $A^1 = \begin{bmatrix} 1/3 & -2/3 \\ -2/3 & 4/3 \end{bmatrix}$ and $A^2 = \begin{bmatrix} 4/3 & -2/3 \\ -2/3 & 1/3 \end{bmatrix}$. The maximum product of advantages comes from $i = (1, 2)$ and $j = (2, 1)$, for which we get $w_x = 1/2$. This results in $r_i = r_j = 1$ and $z = (5/6, 5/6)$. The defection bound is $d = (2, 2)$, resulting in punishment repeats of $a = (14, 14)$. Thus, the equilibrium strategy is to alternate between both players choosing Action 1 and both players choosing Action 2, and using the attack strategy 14 consecutive times as punishment if a player doesn't go along with the plan.

In the definition of this *unbalanced game*, the payoffs have the form $P^1 = \begin{bmatrix} 0 & 1/2 \\ -1 & 1 \end{bmatrix}$ and $P^2 = \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix}$. Any choice for Player 1 works equally well as an attack strategy, and the attack strategy for Player 2 is Action 1. Players can guarantee themselves $g = (0, 0)$, so the advantage game is identical to the original game. No pair of action pairs leads to a positive product of advantages, so the defensive strategies $\delta^1 =$ Action 1 and $\delta^2 =$ Action 2 are computed. The best response for Player 2 to δ^1 is to stay the same, whereas the best response for Player 1 to δ^2 is to switch to Action 2, leading to an improvement of $v^2 = 1$. The Nash equilibrium found by our algorithm is for both players to choose Action 2. Note how, in this case, there are no payoffs that achieve positive advantages *and* the defensive strategies are not in equilibrium. Nevertheless, our algorithm correctly identifies a Nash equilibrium.

As a final example, we define the *exponential game* as a family of games parameterized by the integer b. It is defined by $P^1 = \begin{bmatrix} -2^b & 0 \\ 0 & 2 \end{bmatrix}$ and $P^2 = \begin{bmatrix} 2^b + 1 & 0 \\ 0 & -1 \end{bmatrix}$. The attack strategy for Player 1 is Action 2 and for Player 2 it is Action 1. Players can guarantee themselves $g = (0, 0)$, so the advantage game is identical to the original game. The maximum product of advantages comes from $i = (1, 1)$ and $j = (2, 2)$, for which we get $w_x = 3/(2^{b+1} + 4)$. This results in $r_i = 3$ and $r_j = 2^{b+1}+1$ and $z = ((2^{b-1}+1)/(2^b+2), 1/2)$. The defection bound is $d = (2, 2^b + 1)$, resulting in punishment repeats of $a = (2^{2b+2} + 102^b + 4, 3 \cdot 2^{b+1} + 20)$. Note that these quantities are large, but are still polynomial-size numbers. Therefore, the strategies can be expressed by a polynomial-size counting-node machine. Note also that this example *requires* an exponential-size finite-state machine to achieve advantages greater than zero—by construction, i and j must be combined by a ratio of between $(2^b+1)/(2^b+2)$ and $(2^b)/(2^b + 2)$ and this requires at least 2^b states.

5. CONCLUSIONS

Our interest in repeated games and in the use of threats to spur mutually beneficial behavior came about during a study of automated bidders in simultaneous auctions (Csirik et al. 2001). When human bidders participate in auctions, they have been known to use threats to influence the behavior of other bidders (Weber 1997). We were able to show experimentally that threats can be a valuable tool for automated agents in auctions (Reitsma et al. 2002), and undertook the more theoretical treatment presented here.

Our use of threats in this context echoes their use in folk theorems (Osborne and Rubinstein 1994). Folk theorems can be considered algorithms, since they constructively prove the existence Nash equilibria. However, they generally don't include a computational analysis or include arguments for general games, in particular those with no mutual advantage point. Our results show that the idea behind folks theorems can be used to create an efficient algorithm for computing equilibria in repeated games.

The Nash equilibria found by our algorithm have a number of beneficial properties. The equilibrium payoffs are pareto-efficient; no set of payoffs exists in which both players improve. Equilibria can be found efficiently—nearly as efficiently as in the much easier zero-sum case. The resulting strategies are easy to execute, requiring only a simple counter and the ability to keep track of which nodes follow which other nodes. The payoffs described in this paper satisfy Nash's bargaining axioms, although other criteria such as maximizing the average combined payoff of the two players present no additional difficulty; in one-shot games, identifying such an equilibrium is NP-hard (Conitzer and Sandholm 2002). The fact that punishment is of finite duration makes the equilibrium subgame perfect (Osborne and Rubinstein 1994). For symmetric games, the resulting payoffs are socially optimal and symmetric.

On the negative side, our algorithm depends heavily on the use of the average-payoff criterion. It does not apply to finite-horizon games or even infinite-horizon discounted payoff games, unless the discount factor is set in a game-specific way to a value extremely close to 1.

Nonetheless, there are a number of other contexts in which we are applying the ideas in this paper, such as computing equilibria in repeated Markov games (Littman 2001) and n-player games expressed in a graphical notation (Kearns et al. 2001), and using reinforcement learning to issue and recognize threats (Littman and Stone 2001). The relative simplicity of the threat-based approach makes it a promising direction for future work in computational game theory and electronic commerce more generally.

Acknowledgments

We thank Amy Greenwald, Avi Pfeffer, Luis Ortiz, Michael Kearns, Satinder Singh, Roberto Serrano, and Tuomas Sandholm for discussions and suggestions. This work was supported in part while the authors were in AT&T's Artificial Intelligence Principles Research group and more recently by DARPA IPTO.

References

Robert Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

Vincent Conitzer and Tuomas Sandholm. Complexity results about Nash equilibria. Technical Report CMU-CS-02-135, CMU School for Computer Science, 2002.

János A. Csirik, Michael L. Littman, Satinder Singh, and Peter Stone. FAucS: An FCC spectrum auction simulator for autonomous bidding agents. In *Second International Workshop on Electronic Commerce (WELCOM-2001)*, pages 193–151, 2001.

Michael Kearns, Michael L. Littman, and Satinder Singh. Graphical models for game theory. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 253–260, 2001.

Michael L. Littman. Friend-or-foe Q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328. Morgan Kaufmann, 2001.

Michael L. Littman and Peter Stone. Implicit negotiation in repeated games. In *Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 393–404, 2001.

J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

John F. Nash. The bargaining problem. *Econometrica*, 28: 155–162, 1950.

Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory.* The MIT Press, 1994.

Christos H. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 749–753, 2001.

Paul S. A. Reitsma, Peter Stone, János A. Csirik, and Michael L. Littman. Self-enforcing strategic demand reduction. In *Agent Mediated Electronic Commerce IV: Designing Mechanisms and Systems*, volume 2531 of *Lecture Notes in Artificial Intelligence.* Springer Verlag, 2002.

Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience, New York, NY, 1986.

J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ, 1947.

Robert J. Weber. Making more from less: Strategic demand reduction in the FCC spectrum auctions. *Journal of Economics and Management Strategy*, 6(3):529–548, 1997.