Available online at www.sciencedirect.com

SCIENCE DIRECT®

Decision Support Systems 39 (2005) 55–66

Decision Support Systems

www.elsevier.com/locate/dsw

A polynomial-time Nash equilibrium algorithm for repeated games

Michael L. Littman[a,*], Peter Stone[b]

[a]Department of Computer Science, Rutgers University, Piscataway, NJ 08854-8019, USA
[b]Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-0233, USA

Available online 2 October 2004

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 related but distinct 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 well-known "folk theorem" from game theory and shows how finite-state equilibrium strategies can be found efficiently and expressed succinctly.
© 2004 Elsevier B.V. All rights reserved.

PACS: F.2.m
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 two-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 [12]. The idea behind our algorithm echoes that of the well-known "folk theorem" [11], which shows how the notion of threats can stabilize a wide range of payoff profiles in repeated games. While the folk theorem provides a constructive method for identifying Nash equilibria in repeated games, the contribution of this paper is to show how the threat idea can be used to create a *computationally efficient* equilibrium-finding algorithm. While drawing heavily on the folk theorem, our result is not an immediate corollary. In fact, while there are folk theorems for n-player repeated

* Corresponding author.
E-mail addresses: mlittman@cs.rutgers.edu (M.L. Littman), pstone@cs.utexas.edu (P. Stone).
URLs: http://www.cs.rutgers.edu/~mlittman, http://www.cs.utexas.edu/~pstone.

games, our polynomial-time algorithm is only valid for $n=2$.

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 [10].

As a running example in this paper, we use the well-known Iterated Prisoner's Dilemma to illustrate and motivate our algorithm. In this repeated bimatrix game, on each round, each player can either cooperate (Action 1) or defect (Action 2). The two players use the same payoff matrix, $P^1 = P^2 = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}$.

One pair of equilibrium strategies in the Prisoner's Dilemma is for both players to defect in every round. The average payoff in this case is 1 for both players. These strategies are in equilibrium because a player facing an "always defect" opponent will receive a payoff of zero for every round in which it selects the cooperate action; the best respond to "always defect" is to always choose defect.

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. 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 [14].

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 mapping the interaction history to action choices, 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 joint actions for (p,q), with no two edges from a single node sharing the same label. One outgoing edge for each node is labeled "*" to designate a default edge, taken if the joint action of players p and q does not match any of the other labels.

[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.