AN OPTIMAL MINIMAX ALGORITHM

E.N. GILBERT

AT & T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974, USA

Abstract

Computer game-playing programs repeatedly calculate minimax elements $\mu = \min_i \max_j M_{ij}$ of large payoff matrices M_{ij}. A straightforward *row-by-row calculation* of μ scans rows of M_{ij} one at a time, skipping to a new row whenever an element is encountered that exceeds a current minimax. An *optimal calculation*, derived here, scans the matrix more erratically but finds μ after testing the fewest possible matrix elements. Minimizing the number of elements tested is reasonable when elements must be computed as needed by evaluating future game positions. This paper obtains the expected number of tests required when the elements are independent, identically distributed, random variables. For matrices 50 by 50 or smaller, the expected number of tests required by the row-by-row calculation can be at most 42% greater than the number for the optimal calculation. When the numbers R, C of rows and columns are very large, both calculations require an expected number of tests near $RC/\ln R$.

Keywords and phrases

Minimax, games, strategy, matrix, chess, algorithm.

1. Introduction

The *minimax* of a matrix M_{ij} is the value

$$\mu = \min_i \max_j M_{ij}. \tag{1}$$

Analyses of games or other competitions frequently require a minimax. The matrix has R rows, representing options for player 1, and C columns, representing options for player 2. M_{ij} is the payment from player 1 to player 2 when the players choose

options i and j. Player 2 moves second, and chooses j knowing the option i that player 1 has already chosen. He can then obtain a payment $\max_j M_{ij}$ by optimal choice of j. Knowing this, player 1 minimizes his loss by optimal choice of i. Then the value of the game to player 2 is the minimax μ.

A straightforward calculation of μ might find all R row-maxima $\max_j M_{ij}$ $(i = 1, 2, \ldots, R)$. Then μ is the smallest of these R numbers. This calculation examines each of the RC elements M_{ij}. A somewhat shorter *row-by-row calculation* of μ tests rows one at a time, keeping a record of the smallest row-maximum currently found. When testing row r, the minimax for rows above row r is

$$t_r = \min_{i<r} \max_j M_{ij} \geqslant \mu. \tag{2}$$

Most rows need not be examines completely. For, when an element M_{rk} is found to exceed t_r, then the row-maximum for row r is at least $M_{rk} > t_r \geqslant \mu$ and the minimax μ can not lie in row r. The calculation can now skip to row $r + 1$.

Section 3 derives an *optimal calculation* of μ, which examines the fewest possible matrix elements. The optimal calculation requires memory for extra information beside the current minimax and it also uses some sorting operations. These complications make it unclear whether the optimal calculation is really better to use than the row-by-row calculation. To decide between these two calculations, sect. 4 finds the mean $E(T)$ of the number T of matrix elements that each calculation tests when the RC elements are independent, identically-distributed random variables. For very large matrices, the row-by-row calculation is almost as good as the optimal.

2. Games

A complicated calculation of μ, that minimizes the number of matrix elements tested, can be uneconomical if the tests themselves are simple. However, in some applications, the matrix elements are not available in advance; they are computed, as needed, by long calculations. This situation is faced by computers playing board games, such as chess, checkers, and go, with two players moving by turns and following a game tree.

In principal, these games are equivalent to matrix games (see McKinsey [1], Chap. 5) with numbers R and C of player strategies that are too enormous to make the equivalence helpful. Instead, a computer looks only a few moves ahead, trying to secure a best possible future position. To compare future positions approximately, it calculates a numerical *valuation* based on numbers of pieces of different kinds and on positional advantages and weaknesses (see Shannon [2]).

Of course, a computer looking only two moves ahead solves a matrix game, as described in sect. 1, with M_{ij} the valuations of final positions. There is a large literature