Game Playing: 2-Person, 0-Sum

CSE 415: Introduction to Artificial Intelligence
University of Washington
Spring, 2017

© S. Tanimoto and University of Washington, 2017

Outline

• Two-person, zero-sum games.
• Static evaluation functions.
• Minimax search.
• Alpha-beta pruning.
• Iterative deepening with a time limit.
• Zobrist Hashing.
• Learning a scoring polynomial from experience.

Two-Person, Zero-Sum, Perfect Information Games

1. A two-person, zero-sum game is a game in which only one player wins and only one player loses. There may be ties ("draws"). There are no "win-win" or "lose-lose" instances.
2. Most 2PZS games involve turn taking. In each turn, a player makes a move. Turns alternate between the players.
3. Perfect information: no randomness as in Poker or bridge.
4. Examples of 2PZS games include Tic-Tac-Toe, Othello, Checkers, and Chess.

Why Study 2PZS Games in AI?

1. Games are idealizations of problems.
2. AI researchers can study the theory and (to some extent) practice of search algorithms in an easier information environment than, say, software for the design of the Space Shuttle.
   ("Pure Search")

Static Evaluation Functions

In most of the interesting 2PZS games, state spaces are too large to exhaustively search each alternative evolutionary path to its end.

To find good moves, let’s compute a real-valued function $h(s)$ of a state: $h(s)$ will be high if it is favorable to one player (the player we’ll call Max) and unfavorable to the other player (whom we will call Min).

This function $h(s)$ is called a static evaluation function.

Example in Checkers:

$$h(s) = 5x_1 + x_2$$

Where $x_1 = Max$’s king advantage;
$x_2 = Max$’s single man advantage.

Tic-Tac-Toe Static Eval. Fn.

$$h(s) = 100 A + 10 B + C – (100 D + 10 E + F)$$

A = number of lines of 3 Xs in a row.
B = number of lines of 2 Xs in a row (not blocked by an O).
C = number of lines containing one X and no Os.
D = number of lines of 3 Os in a row.
E = number of lines of 2 Os in a row (not blocked by an X).
F = number of lines containing one O and no Xs.
Minimax Search (Rationale)

If looking ahead one move, generate all successors of the current state, and apply the static evaluation function to each of them, and if we are Max, make the move that goes to the state with the maximum score.

If looking ahead two moves, we will be considering the positions that our opponent can get in two moves, from each of the positions that we can get to in one move.

Assuming that the opponent is playing rationally, the opponent, Min, will be trying to minimize the value of the resulting board.

Therefore, instead of using the static value at each successor of the current state, we examine the successors of each of those, computing their static values, and take the minimum of those as the value of our successor.

Minimax Search (Algorithm)

Procedure minimax(board, whoseMove, plyLeft):
  if plyLeft == 0: return staticValue(board)
  if whoseMove == 'Max': provisional = -100000
  else: provisional = 100000
  for s in successors(board, whoseMove):
    newVal = minimax(s, other(whoseMove), plyLeft-1)
    if (whoseMove == 'Max' and newVal > provisional
        or (whoseMove == 'Min' and newVal < provisional):
      provisional = newVal
  return provisional

Checkers Example

Black to move, White = "Min", Black = "Max"

Alpha-Beta Cutoffs

An alpha (beta) cutoff occurs at a Maximizing (minimizing) node when it is known that the maximizing (minimizing) player has a move that results in a value alpha (beta) and, subsequently, when an alternative to that move is explored, it is found that the alternative gives the opponent the option of moving to a lower (higher) valued position.

Any further exploration of the alternative can be canceled.
Strategy to Increase the Number of Cutoffs

At each non-leaf level, perform a static evaluation of all successors of a node and order them best-first before doing the recursive calls. If the best move was first, the tendency should be to get cutoffs when exploring the remaining ones.

Or, use Iterative Deepening, with ply limits increasing from, say 1 to 15. Use results of the last iteration to order moves in the next iteration.

Another Performance Technique

Avoid recomputing values for some states (especially those within 3 or 4 ply of the current state, which are relatively expensive to compute), by saving their values.

Use a hash table to save: [state, value, ply-used].

As a hashing function, use a Zobrist hashing function:

For each piece on the board, exclusive-or the current key with a pre-generated random number.

Hash values for similar boards are very different.

Hash values can be efficiently computed with an incremental approach (in some games, like checkers and chess, at least).

Zobrist Hashing in Python

```python
# Set up a 64x2 array of random ints.
S = 64
P = 2
zobristnum = [[0]*P for i in range(S)]
from random import randint

def myinit():
    global zobristnum
    for i in range(S):
        for j in range(P):
            zobristnum[i][j] = randint(0, 2**32)
myinit()

# Hash the board to an int.
def zhash(board):
    global zobristnum
    val = 0;
    for i in range(S):
        piece = None
        if(board[i] == 'B'): piece = 0
        if(board[i] == 'W'): piece = 1
        if(piece != None):
            val ^= zobristnum[i][piece]
    return val

# Testing:
b = [' ']*64 ; b[0] = 'B' ; b[1] = 'W'
print(zhash(b))
3473306553
```

Game-Playing Issues

Representing moves: a (Source, Destination) approach works for some games when the squares on the board have been numbered.

Source: The number of the square where a piece is being moved from.

Destination: The number of the square where the piece is being moved to.

(For Othello, only the destination is needed.)

Opening moves:

Some programs use an "opening book" Some competitions require that the first 3 moves be randomly selected from a set of OK opening moves, to make sure that players are "ready for anything"

Regular maximum ply are typically 15-20 for machines, with extra ply allowed in certain situations.

Static evaluation functions in checkers or chess may take 15 to 20 different features into consideration.

Learning a Scoring Polynomial From Experience


Scoring Polynomial

\[ f(s) = a_1 \text{ADV} + a_2 \text{APEX} + a_3 \text{BACK} + \ldots + a_{16} \text{THRET} \]

There are 16 terms at any one time. They are automatically selected from a set of 38 candidate terms.

26 of them are described in the following 3 slides.
Scoring Polynomial Terms

**ADV (Advantage)**
The parameter is credited with 1 for each piece that is in the line and one square beyond the opponent's line of pieces, and is debited with 1 for each piece out of the line and one square beyond the opponent's line of pieces.

**APES (Adv.)**
The parameter is credited with 1 if there are no gaps on the board, or if each gap is occupied by an opponent piece or by no move. The parameter is debited with 1 for each gap or for each opponent piece in a row or column.

**B4C1 (Bail Row Hijack)**
The parameter is credited with 1 if there is an active row on the board and if the row of opponent pieces is occupied by a row of opponent pieces.

**CDD4 (Corner Control)**
The parameter is credited with 1 if any of the following squares (I, 12, 15, 18, 21, 24 and 27) is occupied by a passive piece.

**CDD (Corner Control)**
The parameter is credited with 1 if any of the following squares (I, 12, 15, 18, 21, 24 and 27) is occupied by a passive piece.

**CO4 (Double-Center Control)**
The parameter is credited with 1 if the opponent has no active piece or passive piece on any square in the line of pieces.

**CDD (Double-Center Control)**
The parameter is credited with 1 if the opponent has no active piece or passive piece on any square in the line of pieces.

**EYD (Eyes)**
The parameter is credited with 1 if the opponent is without an active piece in the line of pieces.

**FIVE (Five)**
The parameter is credited with 1 for each piece in the opponent's line of pieces.

**CRAMP**
The parameter is credited with 2 if the passive piece occupies the corner square (12) for Black, and 1 for White,

**DENT (Delete Occupancy)**
The parameter is credited with 1 for each passive piece in the opponent's line of pieces or in the other line of pieces.

**DENT (Delete Occupancy)**
The parameter is credited with 1 for each passive piece in the opponent's line of pieces or in the other line of pieces.

**SCORING POLYNOMIAL COEFFICIENT ADJUSTMENT**

Coefficients are powers of 2.

For each term, the program keeps track of whether its value was correlated with an improvement in the game position over a series of moves.

If so, its value goes up; if not, it goes down.

**Checkers: Computer vs Human**

Samuel's program beat a human player in a widely publicized match in 1962.

Later a program called Chinook, developed by Jonathan Schaeffer at the Univ. of Alberta became the nominal "Man vs Machine Champion of the World" in 1994.*

Checkers playing was the vehicle under which much of the basic research in game playing was developed.

* http://www.math.wisc.edu/~propp/chinook.html