Game Playing: 2-Person, 0-Sum

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

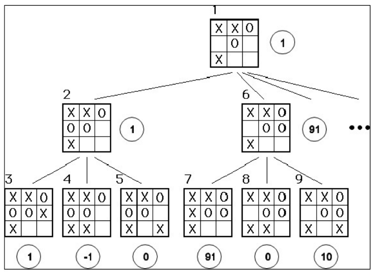
Tic-Tac-Toe Static Eval. Fn.

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 (Illustration)



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 two in one move, from each of the positions that we can get to in one move.

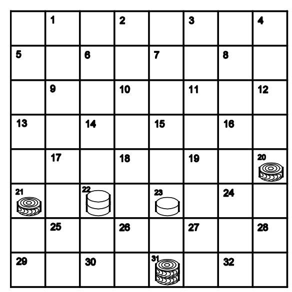
Assuming that the opponent is playing rationally, the opponent, Min, will be trying to minimize the value of the resulting board.

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

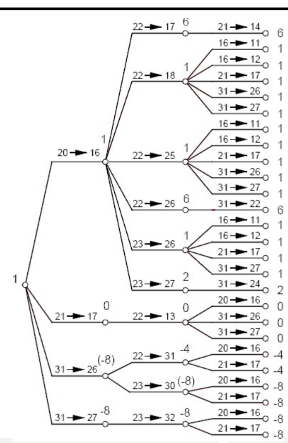


Black to move,

White = "Min",

Black = "Max"

Minimax Search Example



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

```
# 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, \
                4294967296)

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

Arthur Samuel: Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, Vol 3. pp.211-229, 1959.
http://www.research.ibm.com/journal/rd/033/ibmrd0303B.pdf

Arthur Samuel: Some Studies in Machine Learning Using the Game of Checkers. II --- Recent Progress. *IBM Journal*, Vol 116. pp.601-617, 1967.
http://www.research.ibm.com/journal/rd/116/ibmrd1106C.pdf

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.

Ss State-space Search

Scoring Polynomial Terms

ADV (Advancement)
The parameter is credited with 1 for each passive man in the 5th and 6th rows (counting in passive's direction) and debited with 1 for each passive man in the 3rd and 4th rows.

APEX (Apex)
The parameter is debited with 1 if there are no kings on the board, if either square 7 or 26 is occupied by an active man, and if neither of these squares is occupied by a passive man.

BACK (Back Row Bridge)
The parameter is credited with 1 if there are no active kings on the board and if the two bridge squares (1 and 3, or 30 and 32) in the back row are occupied by passive pieces.

CENT (Center Control I)
The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 which is occupied by a passive man.

CNTR (Center Control II)
The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 that is either currently occupied by an active piece or to which an active piece can move.

CORN (Double-Corner Credit)
The parameter is credited with 1 if the material credit value for the active side is 6 or less, if the passive side is ahead in material credit, and if the active side can move into one of the double-corner squares.

DYKE (Dyke)
The parameter is credited with 1 for each string of passive pieces that occupy three adjacent diagonal squares.

EXCH (Exchange)
The parameter is credited with 1 for each square to which the active side may advance a piece and, in so doing, force an exchange.

EXPOS (Exposure)
The parameter is credited with 1 for each passive piece that is flanked along one or the other diagonal by two empty squares.

FORK (Threat of Fork)
The parameter is credited with 1 for each situation in which passive pieces occupy two adjacent squares in one row and in which there are three empty squares so disposed that the active side could, by occupying one of them, threaten a sure capture of one or the other of the two pieces.

GAP (Gap)
The parameter is credited with 1 for each single empty square that separates two passive pieces along a diagonal, or that separates a passive piece from the edge of the board.

GUARD (Back Row Control)
The parameter is credited with 1 if there are no active kings and if either the Bridge or the Triangle of Oreo is occupied by passive pieces.

HOLE (Hole)
The parameter is credited with 1 for each empty square that is surrounded by three or more passive pieces.

Ss State-space Search

CRAMP (Cramp)
The parameter is credited with 2 if the passive side occupies the cramping square (13 for Black, and 20 for White) and at least one other nearby square (9 or 14 for Black, and 19 or 20 for White), while certain squares (17, 21, 22 and 25 for Black, and 8, 11, 12 and 16 for White) are all occupied by the active side.

DENY (Denial of Occupancy)
The parameter is credited with 1 for each square defined in MOB if on the next move a piece occupying this square could be captured without an exchange.

DIA (Double Diagonal File)
The parameter is credited with 1 for each passive piece located in the diagonal files terminating in the double-corner squares.

DIAV (Diagonal Moment Value)
The parameter is credited with 1/2 for each passive piece located on squares 2 removed from the double-corner diagonal files, with 1 for each passive piece located on squares 1 removed from the double-corner files and with 3/2 for each passive piece in the double-corner files.

KCENT (King Center Control)
The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 which is occupied by a passive king.

MOB (Total Mobility)
The parameter is credited with 1 for each square to which the active side could move one or more pieces in the normal fashion, disregarding the fact that jump moves may or may not be available.

MOBIL (Undenied Mobility)
The parameter is credited with the difference between MOB and DENY.

MOVE (Move)
The parameter is credited with 1 if pieces are even with a total piece count (2 for men, and 3 for kings) of less than 24, and if an odd number of pieces are in the move system, defined as those vertical files starting with squares 1, 2, 3 and 4.

NODE (Node)
The parameter is credited with 1 for each passive piece that is surrounded by at least three empty squares.

Ss State-space Search

OREO (Triangle of Oreo)
The parameter is credited with 1 if there are no passive kings and if the Triangle of Oreo (squares 2, 3 and 7 for Black, and squares 26, 30 and 31 for White) is occupied by passive pieces.

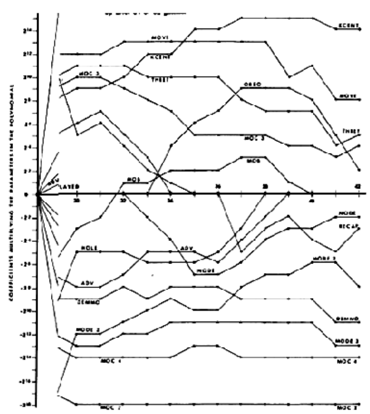
POLE (Pole)
The parameter is credited with 1 for each passive man that is completely surrounded by empty squares.

RECAP (Recapture)
This parameter is identical with Exchange, as defined above. (*It was introduced to test the effects* produced by the random times at which parameters are introduced and deleted from the evaluation polynomial.)

THRET (Threat)
The parameter is credited with 1 for each square to which an active piece may be moved and in so doing threaten the capture of a passive piece on a subsequent move.

Ss State-space Search

Scoring Polynomial Coefficient Adjustment



Coefficients are powers of 2.

They are ordered so that no two are equal at any time.

Ss State-space Search

Polynomial adjustment

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.

Ss State-space Search

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