CS2212
PROGRAMMING
CHALLENGE II

EVALUATION FUNCTIONS

N. H. N. D. DE SILVA
Game playing was one of the first tasks undertaken in AI as soon as computers became programmable. (e.g., Turing, Shannon, and Wiener tackled chess)
Game playing research has spawned a number of interesting research ideas on search, data structures, databases, heuristics, evaluations functions and other areas of computer science.
Recap

Two-person, zero-sum, perfect info games.
Static evaluation functions.
Minimax search.
Alpha-beta pruning.
EVALUATION FUNCTIONS - CONTENTS

• Introduction
• Examples
• Further Improvements
EVALUATION FUNCTIONS - CONTENTS

- Introduction
- Examples
- Further Improvements
EVALUATION FUNCTIONS

INTRODUCTION
In most of the interesting 2PZS games, there are too many possibilities for game positions to exhaustively search each alternative evolutionary path to its end.

In order to determine good moves, one can compute some real-valued function of the board that results from a sequence of moves, and this value 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 is called a static evaluation function.
(STATIC) HEURISTIC EVALUATION FUNCTIONS

- An Evaluation Function:
  - Estimates how good the current board configuration is for a player.
  - Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent’s score from the player’s.
  - Often called “static” because it is called on a static board position.
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces
UTILITY EVALUATION FUNCTION

• If the board evaluation is X for a player, it’s -X for the opponent
  • “Zero-sum game”

• Very game-specific

• Take into account knowledge about game

• “Stupid” utility
  • 1 if player 1 wins (Can use +infinity as well)
  • -1 if player 0 wins (Can use -infinity as well)
  • 0 if tie (or unknown)
  • Only works if we can evaluate complete tree
  • But, should form a basis for other evaluations
UTILITY EVALUATION

• Need to assign a numerical value to the state
  • Could assign a more complex utility value, but then the min/max
determination becomes trickier

• Typically assign numerical values to lots of individual
  factors
  • $a = \#\text{ player 1’s pieces} - \#\text{ player 2’s pieces}$
  • $b = 1$ if player 1 has queen and player 2 does not, -1 if the
    opposite, or 0 if the same
  • $c = 2$ if player 1 has 2-rook advantage, 1 if a 1-rook
    advantage, etc.
STATIC EVALUATION

• A static evaluation function should estimate the true value of a node
  • true value = value of node if we performed exhaustive search
  • need not just be $\infty/0/-\infty$ even if those are only final scores
  • can indicate degree of position
    • e.g. nodes might evaluate to +1, 0, -10

• Children learn a simple evaluation function for chess
  • $P = 1, N = B = 3, R = 5, Q = 9, K = 1000$
  • Static evaluation is difference in sum of scores
  • chess programs have much more complicated functions
UTILITY EVALUATION

• The individual factors are combined by some function

• Usually a linear weighted combination is used
  • $u = \alpha a + \beta b + \gamma c$
  • Different ways to combine are also possible

• Notice: quality of utility function is based on:
  • What features are evaluated
  • How those features are scored
  • How the scores are weighted/combined

• Absolute utility value doesn’t matter – relative value does.
EVALUATION FUNCTIONS - CONTENTS

- Introduction
- Examples;
  - Tic-Tac-Toe
  - Chess
  - Checkers
- Further Improvements
EVALUATION FUNCTIONS

EXAMPLES: TIC-TAC-TOE
O’S AND X’S

• A simple evaluation function for O’s and X’s is:
  • Count lines still open for maX,
  • Subtract number of lines still open for min
  • evaluation at start of game is 0
  • after X moves in center, score is +4

• Evaluation functions are only heuristics
  • e.g. might have score -2 but maX can win at next move
    • O - X
    • - O X
    • - - -

• Use combination of evaluation function and search
TIC-TAC-TOE STATIC EVAL. FN.

\[ f(\text{board}) = 100 \ A + 10 \ B + C - (100 \ D + 10 \ E + F) \]

- **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.
EVALUATION FUNCTIONS

EXAMPLES: CHESS
Evaluation functions

For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with

$$f_1(s) = \text{(number of white queens)} - \text{(number of black queens)}, \text{ etc.}$$
REPRESENTATIONS REQUIRED

• Board Positions:

• Board-Centric:
  - Chess board has 64 squares, each may be empty or occupied by one of twelve kinds of chessmen.
  - Therefore one may use an 2-d 8x8 array (or 1-d 64-vector) of 4-bit values, (or several bitboards to be described in a future lecture).

• Piece-Centric:
  - A player has 16 men, each is either captured or on one of 64 squares.
  - Therefore one may use a 1-d 16-vector of 7-bit values.
REPRESENTATIONS REQUIRED

• **Moves:**
  • A chessman belonging to one player moves from one square to another.
  • Therefore one may use a pair \((\text{FromSquare}, \text{ToSquare})\) of 6-bit numbers.
REQUIREMENTS OF REPRESENTATIONS

• The rules of chess have two oddities that defy this simple representation of moves (and piece-centric board representations too)

• **Castling:**
  • Subject to certain provisos, a king K and a rook R may move simultaneously
    • Each K just once per game, neither it nor R has yet moved, K is not moving out of check or through check, all spaces empty between K & R
  • K may normally only move one square; in castling it moves two squares.
  • Special logic could handle the implied R movement of a 2-square K move.
REQUIREMENTS OF REPRESENTATIONS

• Pawn Promotion:
  • A pawn that reaches the 8th rank is replaced with another piece: Queen Rook Bishop or Knight
  • Piece-Centric board representation now needs to say what type of piece
  • (FromSquare, ToSquare) pair needs extra WhatPieceNow element
CHESS ENGINE ELEMENT: THE RULES

• The rules of movement and capture need to be represented:

• Different pieces may move in different ways:
  • Pawn: forward two empty squares from its starting position;
  • Pawn: one empty square forward from any position;
  • Bishop: diagonally any direction any number of empty squares;
  • Rook: forward backward or sideways any number of empty squares;
CHESS ENGINE ELEMENT: THE RULES

• Different pieces may move in different ways:
  • Queen: diagonally like bishop or to-or-fro-or-sideways like rook;
  • Knight: one square forward or back plus two sideways, or vice versa
  • King: to any of eight adjacent squares, or special castling move.
CHESS ENGINE ELEMENT: THE RULES

• All pieces except pawns may capture an opposing piece by finishing their movement where it is. The opposing piece is removed.
  • Pawns capture with a one-square forward-diagonal;
  • There is an en-pasquant rule too. Read all about it!
CHESS ENGINE ELEMENT: MAKE MOVE AND RETRACT MOVE

• User-input moves, and Computer-Generated responses, must have the correct effect, and be reflected in the GUI.

• Interactive users should be given the opportunity to Undo a pair of moves.
  • Therefore, Move Representation must include information on captures too, so that a captured piece can be reinstated by Undo.
CHESS ENGINE ELEMENT: MAKE MOVE AND RETRACT MOVE

• The Make-Move and Retract-Move functionality is needed also, intensively, while the computer player is searching the game tree
  • Must therefore be fast
  • Will not involve GUI
CHESS ENGINE ELEMENT: EVALUATION FUNCTION

• An evaluation function provides a number which indicates how good a position is for one player.
  • This is vague, but should not be treated as probability of a win.
  • Evaluation function will be heavily used in search, so should be fast.

• Evaluation functions for chess are typically dominated by material balance.

• Typical values: Pawn 1; Bishop 3; Knight 3; Rook 5; Queen 9; King infinite.
CHESS ENGINE ELEMENT: EVALUATION FUNCTION

• Other features taken into account too:
  • Control of the centre four squares
  • Passed pawns
  • Mobility, especially of the queens.
    • The sum of possible value of all other features combined is typically regarded as no more than 1.5 pawns
CHESS ENGINE ELEMENT: OPENING BOOK

• It quickly proved too hard to select good opening moves using limited search and an evaluation function.

• Centuries of human experience are codified in opening books which serious players study.

• Chess programs use the knowledge in these publications,
  • perhaps augmented by team members expert or better in chess,
  • coded by programmers into a form their program can use.
CHESS ENGINE ELEMENT: OPENING BOOK

- A common strategy of human players confronting computers is to make moves out of the book - i.e. not found in the book - in the expectation that the computer will not be able to find the responses which make the move sub-optimal.
CHESS ENGINE ELEMENT: ENDCASE DATABASES

• An Endgame database is a tabulation of the possible positions in which only a very small number of chessmen remain on the board. For each position, it records the best move.

• Examples are:
  • King and Pawn versus King (KPK)
  • King and Rook versus King (KRK)
  • King Rook and Pawn versus King and Rook (KRPKR)
CHESS ENGINE ELEMENT: ENDGAME DATABASES

• Some endgame databases did exist, as books, before computer chess.

• But Computer Chess has contributed enormously to the chess world’s knowledge of several endgames, through exhaustive analysis of positions too numerous for humans to tabulate. Championship contenders have been known to consult computer-generated databases overnight during an adjourned game.
• If you had a *perfect* utility evaluation function, what would it mean about the minimax tree?
EVALUATION FUNCTIONS

• If you had a perfect utility evaluation function, what would it mean about the minimax tree?

You would never have to evaluate more than one level deep!

• Typically, you can’t create such perfect utility evaluations, though.
EVALUATION FUNCTIONS FOR ORDERING

• As mentioned earlier, order of branch evaluation can make a big difference in how well you can prune

• A good evaluation function might help you order your available moves
  • Perform one move only
  • Evaluate board at that level
  • Recursively evaluate branches in order from best first move to worst first move (or vice-versa if at a MIN node)
EVALUATION FUNCTIONS

- Must execute quickly - constant time
  - parallel evaluation: allows more complex functions
    - tactics: patterns to recognize weak positions
    - arbitrarily complicated domain knowledge
LEARNING BETTER EVALUATION FUNCTIONS

- Deep Blue learns by tuning weights in its board evaluation function

\[ f(p) = w_1f_1(p) + w_2f_2(p) + \ldots + w_nf_n(p) \]

- Tune weights to find best least-squares fit with respect to moves actually chosen by grandmasters in 1000+ games. Weights tweaked multiple digits of precision.
LEARNING A SCORING POLYNOMIAL BY 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.
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.
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.

KCEN (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.
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.
Coefficients are powers of 2. They are ordered so that no two are equal at any time.
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.
EVALUATION FUNCTIONS

EXAMPLES: CHECKERS
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
SEARCH IN CHECKERS

• What is the search tree like in Checkers?
• It is possible to get Zugzwang -- every move loses
• Fine line between win and draw
  • e.g. textbook example with 40 critical moves required
SEARCH IN CHECKERS

• Captures are forced, reducing branching rate
  • about 8 when no captures, about 1.25 when captures

• Endgame databases heavily used
  • Chinook can use the endgame database at the root
STATIC EVALUATION FUNCTION

- Game divided into 5 phases
  - opening
  - middlegame
  - early endgame
  - late endgame
  - endgame database
STATIC EVALUATION FUNCTION

• Different evaluation function at each stage

• Search involves all phases early in the game
  • deal with ‘blemish effect’ by rescaling each phase
    • i.e. problem that score jumps as move from one phase to next
STATIC EVALUATION FUNCTION

• Game divided into 5 phases
  • Opening 22 parameters
  • Middlegame 22 parameters
  • early endgame 22 parameters
  • late endgame 22 parameters
  • endgame database perfect info

• Different evaluation function ...
STATIC EVALUATION FUNCTION

• Game divided into 5 phases
  • Opening  22 parameters
  • Middlegame  22 parameters
  • early endgame  22 parameters
  • late endgame  22 parameters
  • endgame database  perfect info

• Search involves all phases …
  • deal with ‘blemish effect’ by rescaling …
STATIC EVALUATION FUNCTION

• Game divided into 5 phases
  • Opening 22 parameters
  • Middlegame 22 parameters
  • early endgame 22 parameters
  • late endgame 22 parameters
  • endgame database perfect info

• Total is 88 parameters plus scaling factors!
SEARCH IN CHECKERS

• Search early in the game involves all of …
  • opening
  • middlegame
  • endgame
  • endgame database

• All of these factors make search different from chess
ENDGAME DATABASES

• Endgames can be 100 moves long in checkers
  • branching rate of 2 gives $2^{100} = 10^{30}$ positions.

• But there is a trick when few pieces left
  • first used by Ken Thompson in Chess
    • (Ken Thompson wrote first Unix, won Turing award)
ENDGAME DATABASES

• Calculate the true optimal move for every position
  • e.g. $406 \times 10^9$ eight piece positions, $4 \times 10^{11} << 10^{30}$

• Best version of Chinook used 8 piece databases
  • web version just uses 6 piece
HOW TO CALCULATE ENDOGAM DB’S

• 0. Duplicate every position, for W/B to move
• 1. Try every possible position of the pieces
  • Mark as Win/0 if W has won in this position
  • Mark as Loss/0 if W has lost in this position
  • Leave other positions unmarked
• 2. Iterate until no new positions are marked
  • 2a) Try every unmarked position of the pieces
    • If W has move to Win/n, mark as Win/n+1
    • If every W move is to Loss/n, mark as Loss/n+1
CALCULATING ENDGAME DATABASES

• When this process has finished
  • We know the number of moves to win for every position
    • where either B or W can force win
  • Therefore every other position must be drawn

• so the following is valid

• 3. Mark all unmarked positions as Drawn

• We can calculate optimal winning moves
  • In Win/n position, best move for W is to any Win/n-1
  • in Drawn, best move for W is to any Drawn position,
  • In Loss/n position, best move is to Loss/n-1
PUTTING IT ALL TOGETHER

• Standard techniques
  • $\alpha$–$\beta$ search
  • endgame databases

• variable search depth
  • e.g. pursue positions with material deficit less deeply
PUTTING IT ALL TOGETHER

• An AntiBook

  • not a standard opening book known to opponents
  • a book of disallowed moves, encouraging novelty
EVALUATION FUNCTIONS - CONTENTS

• Introduction

• Examples

• Further Improvements;
  • Quiescent search
  • Transposition Tables
  • Iterative deepening
  • Aspiration Windows
EVALUATION FUNCTIONS

FURTHER IMPROVEMENTS: QUIESCENT SEARCH
FURTHER IMPROVEMENTS: QUIESCENT SEARCH

• ‘Don’t leave a mess strategy’

• For evaluating the leaves at depth 0, instead of the evaluation function a special function is called that evaluates special moves (e.g. captures) only down to infinite depth

• Guarantees e.g. that the Queen will not be captured at move in depth 0
EVALUATION FUNCTIONS

FURTHER IMPROVEMENTS: TRANSPPOSITION TABLES
FURTHER IMPROVEMENTS: TRANSPOSITION TABLES

• Introduced by Greenblat's Mac Hack (1966)

• Basic idea: caching
  • once a board is evaluated, save in a hash table, avoid re-evaluating.
  • called “transposition” tables, because different orderings (transpositions) of the same set of moves can lead to the same board.
FURTHER IMPROVEMENTS: TRANSPOSITION TABLES AS LEARNING

• Is a form of root learning (memorization).
  • positions generalize sequences of moves
  • learning on-the-fly

• Deep Blue --- huge transposition tables (100,000,000+), must be carefully managed.
EVALUATION FUNCTIONS

FURTHER IMPROVEMENTS: ITERATIVE DEEPENING
FURTHER IMPROVEMENTS: ITERATIVE DEEPENING

• First try depth $n=1$
• If time left, try depth $n+1$
• Order moves of depth $n$ when trying depth $n+1$!
• Since alpha beta is order sensitive, this can speed up the process
• Fills time and doesn’t need predefined depth parameter
• Drawback: creates same positions over and over, but…
FURTHER IMPROVEMENTS: ITERATIVE DEEPENING

• Example for multiply generated moves:

• Assumption: worst case: no alpha beta pruning.

• Branching factor 10
## FURTHER IMPROVEMENTS: ITERATIVE DEEPENING

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Steps</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10 + 100</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>10 + 100 + 1000</td>
<td>1110</td>
</tr>
<tr>
<td>4</td>
<td>10 + 100 + 1000 + 10000</td>
<td>11110</td>
</tr>
<tr>
<td>5</td>
<td>10 + ... + 100000</td>
<td>111110</td>
</tr>
</tbody>
</table>

\[
111110 \text{ position} \quad 123,450 \text{ positions}
\]

\[
123,450 \div 111,110 = 1.11 \quad \Rightarrow \quad \text{only 11% additional pos. (worst case)}
\]
EVALUATION FUNCTIONS

FURTHER IMPROVEMENTS: ASPIRATION WINDOWS
FURTHER IMPROVEMENTS: ASPIRATION WINDOWS

• Extension of iterative deepening

• Basic Idea: feed alpha beta values of previous search into current search

• Assumption: new values won’t differ too much

• Extend alpha beta by +/- window value
EVALUATION FUNCTIONS – SUMMARY

• Introduction

• Examples:
  • Tic-Tac-Toe
  • Chess
  • Checkers

• Further Improvements:
  • Quiescent search
  • Transposition Tables
  • Iterative deepening
  • Aspiration Windows
REFERENCES

• **CIS 350 – I ; Game Programming** by Rolf Lakaemper

• **Minimax Trees:** Utility Evaluation, Tree Evaluation, Pruning initially by Yoonsuck Choe

• **Foundations of Artificial Intelligence - Adversarial Search R&N: Chapter 5 Part II** - Bart Selman

• **Game-Playing & Adversarial Search** – Richard H. Lathrop

• **Artificial Intelligence - Games 1: Game Tree Search** - Ian Gent

• **Game Playing: Checkers** - Ian Gent

• **Computer Chess** – A. Cater

• **Alpha-Beta Search** - S. Tanimoto