22c:145 Artificial Intelligence

Adversarial Search

Textbook: Chapter 5

Outline

- Adversarial Search
- Optimal decisions
- Minimax
- α-β pruning
- ExpectMinimax
- Case study: Deep Blue

Adversarial Reasoning: Games

Mathematical Game Theory

Branch of economics that views any multi-agent environment as a game, provided that the impact of each agent on the others is "significant", regardless of whether the agents are cooperative or competitive.

First step to AI Games:

- **Deterministic**
- **Turn taking**
- **2-player**
- **Zero-sum game of perfect information (fully observable) "my win is your loss" and vice versa; utility of final states opposite for each player. My +10 is your -10.**

Game Playing vs. Search

Multi-agent game vs. single-agent search problem

"Unpredictable" opponent need a strategy: specifies a move for each possible opponent reply.
E.g with "huge" lookup table.

Time limits → unlikely to find optimal response, must approximate

Rich history of game playing in AI, in particular in the area of chess.

Turing viewed chess as an important challenge for machine intelligence because playing chess appears to require some level of intelligence.

Why is Game-Playing a Challenge for AI?

- **Competent game playing is a mark of some aspects of "intelligence"**
  - **Requires planning, reasoning and learning**
- **Proxy for real-world decision making problems**
  - **Easy to represent states & define rules**
  - **Obtaining good performance is hard**
- **"Adversary" can be nature**
- **PSPACE-complete (or worse)**
  - **Computationally equivalent to hardware debugging, formal verification, logistics planning**
  - **PSPACE is harder than NP.**

Traditional Board Games

- Finite
- Two-player
- Zero-sum
- Deterministic
- Perfect Information
- Sequential

Formal definition of a game:

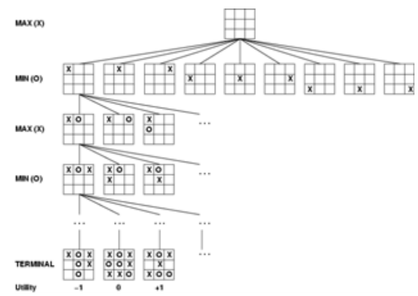
- Initial state
- Successor function: returns list of *(move, state)* pairs
- Terminal test: determines when game over
  Terminal states: states where game ends
- Utility function (objective function or payoff function): gives numeric value for terminal states

We will consider games with 2 players (**Max and Min**)

Max moves first.

Game Tree Example: Tic-Tac-Toe

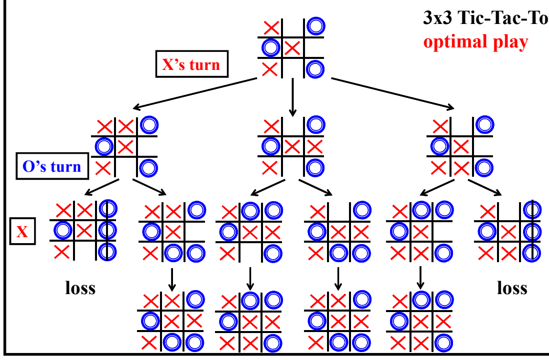
Tree from Max's perspective



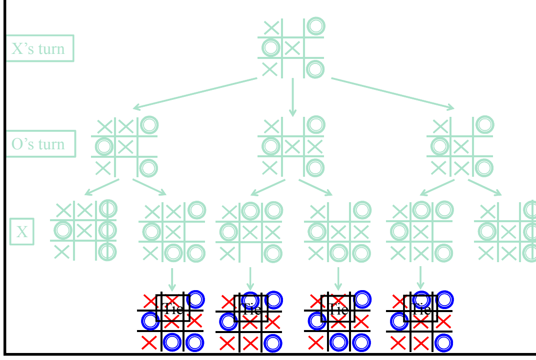
Key Idea: Look Ahead

After 3 moves per player we are in:

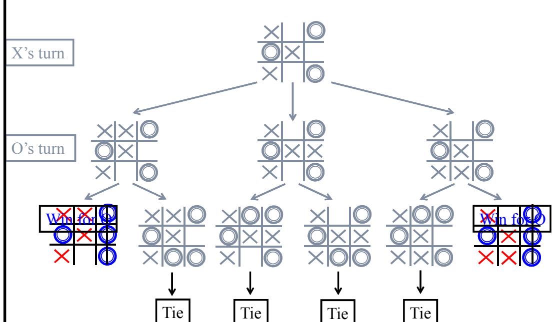
3x3 Tic-Tac-Toe
optimal play



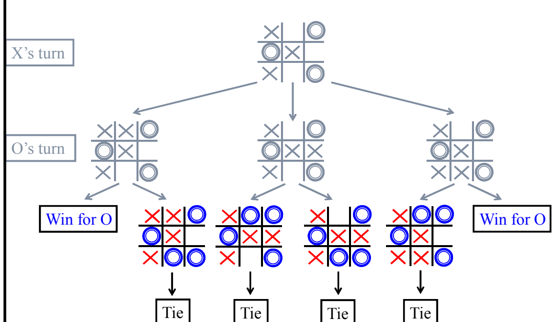
Look-ahead based Tic-Tac-Toe



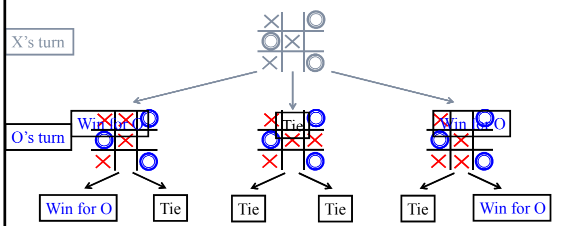
Look-ahead based Tic-Tac-Toe



Look-ahead based Tic-Tac-Toe



Look-ahead based Tic-Tac-Toe

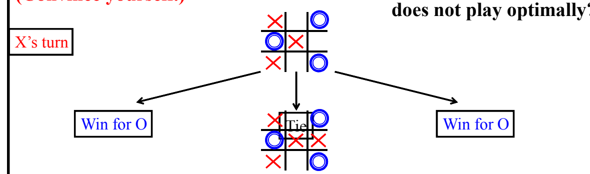


13

Each board in game tree gets unique game tree value (utility; -1/0/+1) under optimal rational play. (Convince yourself.)

E.g. 0 for top board.

What if our opponent does not play optimally?



Approach: Look first at bottom tree. Label bottom-most boards.
Then label boards one level up, according result of best possible move.
… and so on. Moving up layer by layer.

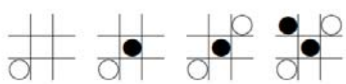
Termed the Minimax Algorithm
- **Implemented as a depth-first search**

14

Well… Why not use a strategy / knowledge, as humans do?

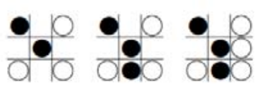
Consider for Tic-Tac-Toe:

1. If there is a winning move, make it.
2. If the opponent can win at a square by his next move, play that move.
3. Taking the central square is more important than taking other squares.
4. Taking corner squares is more important than taking squares on the edges.



Oops!!

Consider Black uses the strategy…



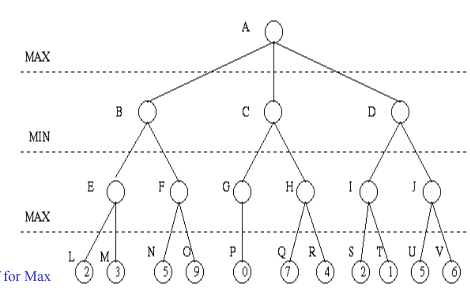
15

So, although one can capture strategic knowledge of many games in high-level rules (at least to some extent), in practice any interesting game will revolve *precisely around the exceptions to those rules!*

Issue has been studied for decades but research keeps coming back to game tree search (or most recently, game tree sampling).

16

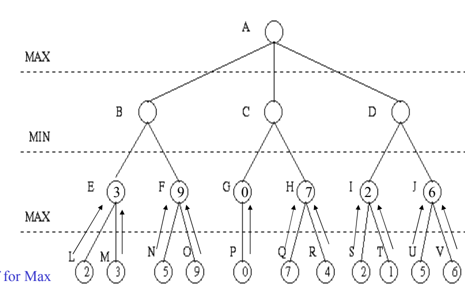
Minimax Algorithm



Payoff for Max

17

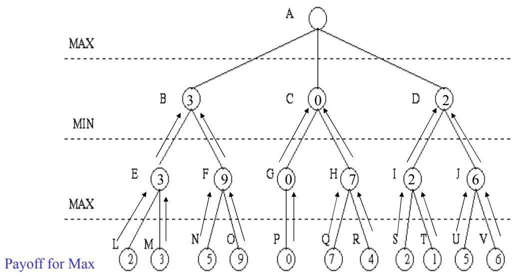
Minimax Algorithm (cont' d)



Payoff for Max

18

Minimax Algorithm (cont’d)

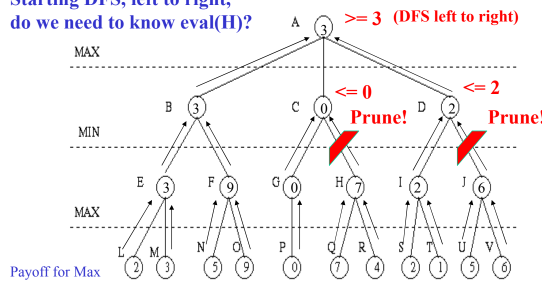


Minimax Algorithm

What if
payoff(Q) = 100
payoff(R) = 200

Starting DFS, left to right,
do we need to know eval(H)?

Do DFS. Real games:
use iterative deepening.
(gives “anytime” approach.)



Minimax Algorithm

Minimax algorithm

- Perfect play for deterministic, 2-player game
- Max tries to maximize its score
- Min tries to minimize Max’s score (Min)
- Goal: Max to move to position with highest minimax value
  → Identify best achievable payoff against best play

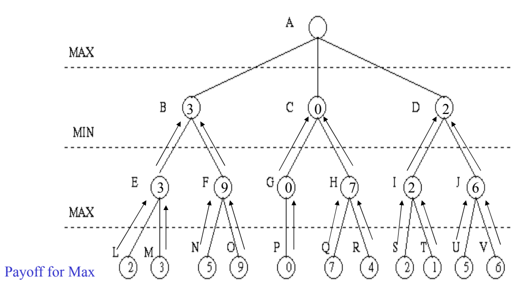
Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
   v ← MAX-VALUE(state)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s))
   return v

function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s))
   return v
```


Minimax Algorithm



Properties of minimax algorithm:

<u>Complete?</u> Yes (if tree is finite)

<u>Optimal?</u> Yes (against an optimal opponent)

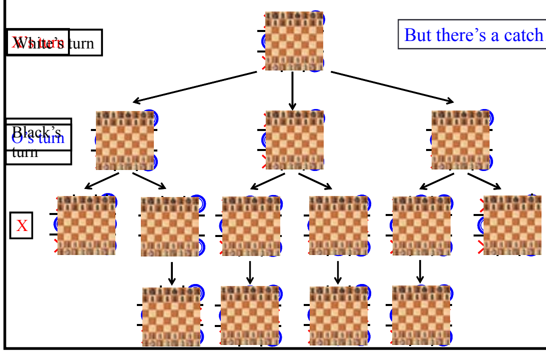
<u>Time complexity?</u> $O(b^m)$

<u>Space complexity?</u> O(bm) (depth-first exploration, if it generates all successors at once)

For chess, $b \approx 35$, $m \approx 80$ for "reasonable" games
→ exact solution completely infeasible

m – maximum depth of the tree; b – legal moves

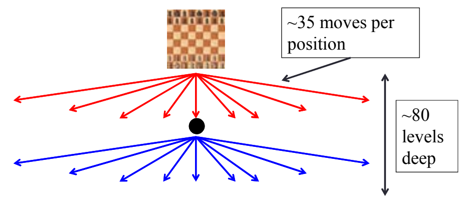
Look-ahead based Chess



But there's a catch…

25

How big is this tree?



Approx. 10^120 > Number of atoms in the observable universe (10^80)

We can really only search a **tiny, miniscule faction** of this tree!

Around 60 x 10^9 nodes for 5 minute move. **Approx. 1 / 10^70 fraction.**

26

Cutoff Search

Suppose we have 100 secs, explore 10^4 nodes/sec
→ 10^6 nodes per move

Does it work in practice?

$b^m = 10^6$, $b=35$ → $m=4$

4-ply lookahead is a hopeless chess player!
- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov

Other improvements…

27

Resource limits

Can't go to all the way to the "bottom:"

evaluation function
= estimated desirability of position

cutoff test:
e.g., depth limit
(Use Iterative Deepening)

What is the problem with that?

Horizon effect.

"Unstable positions:"
Search deeper.
Selective extensions.
E.g. exchange of several pieces in a row.

→ add quiescence search:
→ quiescent position: position where next move unlikely to cause large change in players' positions

28

Evaluation Function

- **Performed at search cutoff point**
- **Must have same terminal/goal states as utility function**
- **Tradeoff between accuracy and time → reasonable complexity**
- **Accurate**
  - **Performance of game-playing system dependent on accuracy/goodness of evaluation**
  - **Evaluation of nonterminal states strongly correlated with actual chances of winning**

29

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
$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Key challenge – find a good evaluation features:
Not just material! (as used by novice)
Isolated pawns are bad.
How well protected is your king?
How much maneuverability to you have?
Do you control the center of the board?
Strategies change as the game proceeds

Features are a form of chess knowledge. Hand-coded in eval function.
Knowledge tightly integrated in search.
Feature weights: can be automatically tuned ("learned").
Standard issue in machine learning:
Features, generally hand-coded; weights tuned automatically.

30

Minimax Algorithm

Limitations

- **Generally not feasible to traverse entire tree**
- **Time limitations**

Key Improvements

- **Use evaluation function instead of utility (discussed earlier)**
  - **Evaluation function provides estimate of utility at given position**
- **Alpha/beta pruning**

31

α-β Pruning

Can we improve search by reducing the size of the game tree to be examined?

→ **Yes! Using alpha-beta pruning**

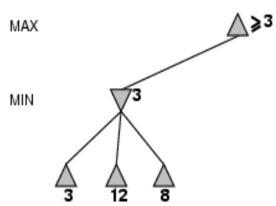
Principle

- **If a move is determined worse than another move already examined, then there is no need for further examination of the node.**

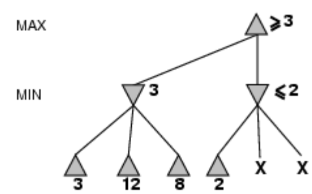
Analysis shows that will be able to search almost twice as deep.
This really is what makes game tree search practically feasible.
E.g. Deep Blue 14 plies using alpha-beta pruning.
Otherwise only 7 or 8 (weak chess player). (plie = half move / one player)

32

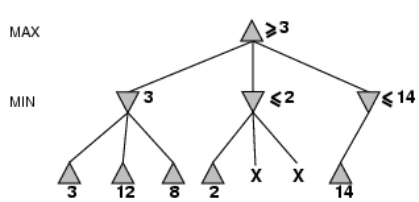
α-β Pruning Example



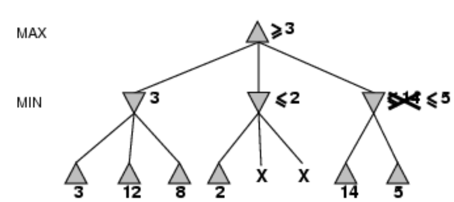
33



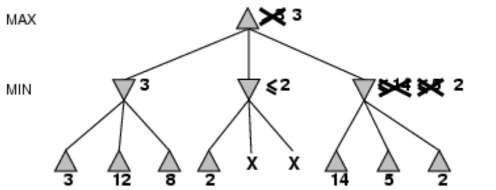
34



35



36



Note: order children matters!

What gives best pruning?

Visit most promising (from min/max perspective) first.

Alpha-Beta Pruning

Initially for the root, α is –infinity, and β is +infinity.

Rules:

- **α is the best (highest) found so far along the path for Max**
- **β is the best (lowest) found so far along the path for Min**
- **Search below a MIN node may be alpha-pruned if its β ≤ α of some MAX ancestor**
- **Search below a MAX node may be beta-pruned if its α ≥ β of some MIN ancestor.**

The α-β algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game
   v ← MAX-VALUE(state, −∞, +∞)
   return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s, α, β))
      if v ≥ β then return v
      α ← MAX(α, v)
   return v
```

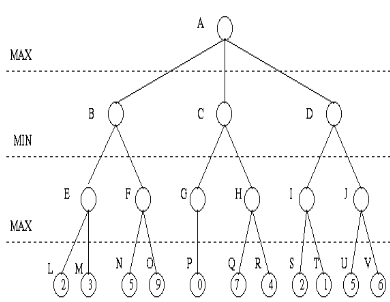

The α-β algorithm

```
function MIN-VALUE(state, α, β) returns a utility value
   inputs: state, current state in game
           α, the value of the best alternative for MAX along the path to state
           β, the value of the best alternative for MIN along the path to state
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← +∞
   for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s, α, β))
      if v ≤ α then return v
      β ← MIN(β, v)
   return v
```


Alpha-Beta Pruning Example

1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.

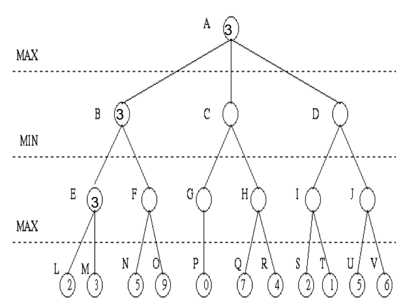
2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.



Alpha-Beta Pruning Example

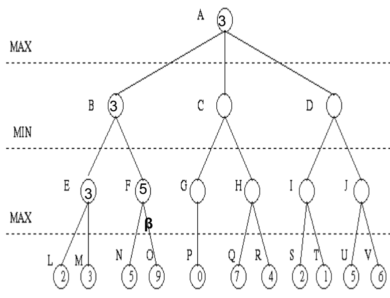
1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.



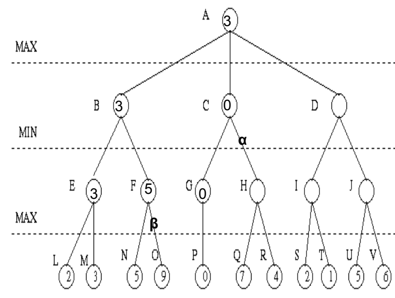
Alpha-Beta Pruning Example

1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.
2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.



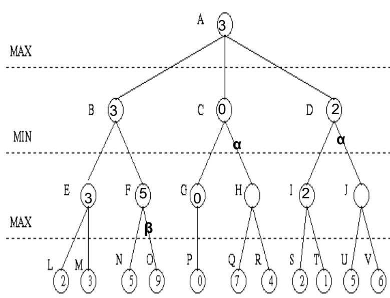
Alpha-Beta Pruning Example

1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.
2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.

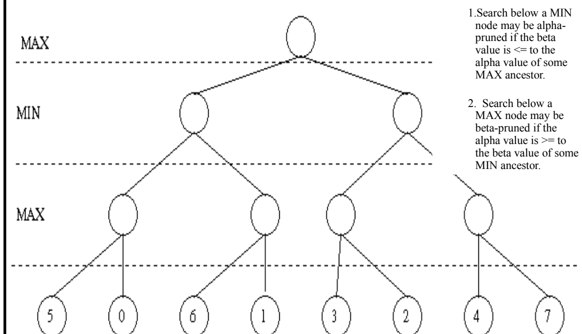


Alpha-Beta Pruning Example

1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.
2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.

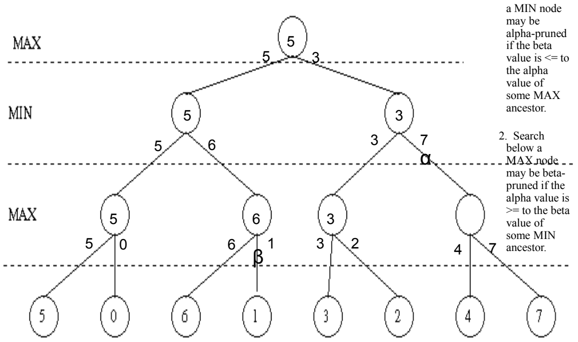


Another Example



1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.
2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.

Example



1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.
2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.

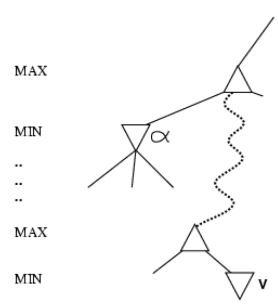
More abstractly

α* is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max

If *v* is worse than α, *max* will avoid it

→ prune that branch

Define β similarly for *min*



48

Properties of α-β Prune

Pruning does not **affect final result**

Good move ordering improves effectiveness of pruning b(e.g., chess, try captures first, then threats, forward moves, then backward moves…)

With "perfect ordering," time complexity = O($b^{m/2}$)
→ doubles depth of search that alpha-beta pruning can explore

Example of the value of reasoning about which computations are relevant (a form of metareasoning)

A few quick approx. numbers for Chess:

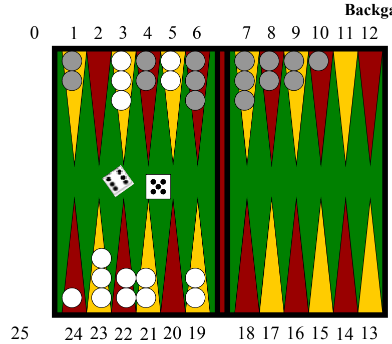
b = 35
200M nodes / second ===> 5 mins = 60 B nodes in search tree
(2 M nodes / sec. software only, fast PC ===> 600 M nodes in tree)

35^7 = 64 B
35^5 = 52 M

So, basic minimax: around 7 plies deep. (5 plies)
With, alpha-beta 35^(14/2) = 64 B. Therefore, 14 plies deep. (10 plies)

> **Aside:**
> **4-ply ≈ human novice**
> **8-ply / 10-ply ≈ typical PC, human master**
> **14-ply ≈ Deep Blue, Kasparov (+ depth 25 for "selective extensions") / 7 moves by each player.**

When Chance is involved: Backgammon Board



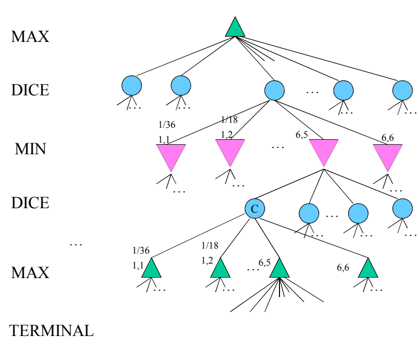
Expectiminimax

Generalization of minimax for games with chance nodes

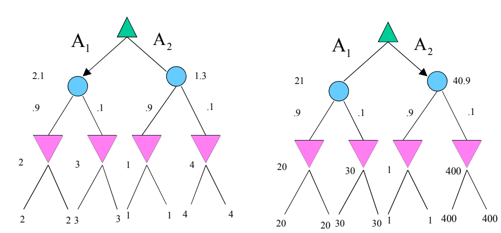
Examples: Backgammon, bridge

Calculates expected value where probability is taken over all possible dice rolls/chance events
- Max and Min nodes determined as before
- Chance nodes evaluated as weighted average

Game Tree for Backgammon



Expectiminimax



.9 * 2 + .1 * 3 = 2.1

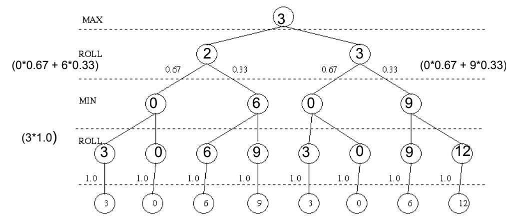
Small chance at high payoff wins. But, not necessarily the best thing to do!

Expectiminimax

Expectiminimax(n) =

Utility(n)	for n, a terminal state
$max_{s \in Succ(n)}$ expectiminimax(s)	for n, a Max node
$min_{s \in Succ(n)}$ expectiminimax(s)	for n, a Min node
$\Sigma_{s \in Succ(n)} P(s) *$ expectiminimax(s)	for n, a chance node

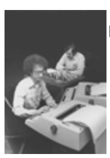
Expectiminimax Example



A Brief History of Computer Chess


1912


1950s


1970s

1997


Today

Human-computer hybrid most exciting new level of play. Computers as smart assistants are becoming accepted.
Area referred to as "Assisted Cognition."

Chess: Computer vs Human

Deep Blue is a chess-playing computer developed by IBM.

- On February 10, 1996, Deep Blue became the first machine to win a chess game against a reigning world champion (Garry Kasparov) under regular time controls.
- On May 11, 1997, the machine won a six-game match by two wins to one with three draws against world champion Garry Kasparov.

Chess: Computer vs Human

Deep Fritz is a German chess program developed by Frans Morsch and Mathias Feist and published by ChessBase.

- In 2002, Deep Fritz drew the Brains in Bahrain match against the classical World Chess Champion Vladimir Kramnik 4-4.
- On June 23, 2005, in the ABC Times Square Studios, Fritz 9 drew against the then FIDE World Champion Rustam Kasimdzhanov.
- From 25 November-5 December 2006 Deep Fritz played a six game match against Kramnik in Bonn. Fritz was able to win 4-2.

Combinatorics of Chess

- Opening book
- Endgame
  - database of all 5 piece endgames exists; database of all 6 piece games being built
- Middle game
  - Positions evaluated (estimation)
    - 1 move by each player = 1,000
    - 2 moves by each player = 1,000,000
    - 3 moves by each player = 1,000,000,000

61

Positions with Smart Pruning

Search Depth (ply)		Positions
2		60
4		2,000
6		60,000
8		2,000,000
10	(<1 second DB)	60,000,000
12		2,000,000,000
14	(5 minutes DB)	**60,000,000,000**
16		2,000,000,000,000

How many lines of play does a grand master consider?

Around 5 to 7 ☺

62

History of Search Innovations

Shannon, Turing	**Minimax search**	**1950**
Kotok/McCarthy	**Alpha-beta pruning**	**1966**
MacHack	**Transposition tables**	**1967**
Chess 3.0+	**Iterative-deepening**	**1975**
Belle	**Special hardware**	**1978**
Cray Blitz	**Parallel search**	**1983**
Hitech	**Parallel evaluation**	**1985**
Deep Blue	**ALL OF THE ABOVE**	**1997**

63

Time vs Space

- **Iterative Deepening**
  - **a good idea in chess, as well as almost everywhere else!**
  - **Chess 4.x, first to play at Master's level**
  - **trades a little time for a huge reduction in space**
    - **lets you do breadth-first search with (more space efficient) depth-first search**
  - **anytime: good for response-time critical applications**

64

Special-Purpose and Parallel Hardware

- **Belle (Thompson 1978)**
- **Cray Blitz (1993)**
- **Hitech (1985)**
- **Deep Blue (1987-1996)**
  - **Parallel evaluation: allows more complicated evaluation functions**
  - **Hardest part: coordinating parallel search**
  - **Interesting factoid: Deep Blue never quite played the same game, because of "noise" in its hardware!**

65

Deep Blue

- **Hardware**
  - **32 general processors**
  - **220 VSLI chess chips**
- **Overall: 200,000,000 positions per second**
  - **5 minutes = depth 14**
- **Selective extensions - search deeper at unstable positions**
  - **down to depth 25 !**

Aside:
4-ply ≈ human novice
8-ply to 10-ply ≈ typical PC, human master
14-ply ≈ Deep Blue, Kasparov (+ depth 25 for "selective extensions")

66

Evolution of Deep Blue

From 1987 to 1996
- faster chess processors
- port to IBM base machine from Sun
  - Deep Blue's non-Chess hardware is actually quite slow, in integer performance!
- bigger opening and endgame books
- 1996 differed little from 1997 - fixed bugs and tuned evaluation function!
  - After its loss in 1996, people underestimated its strength!

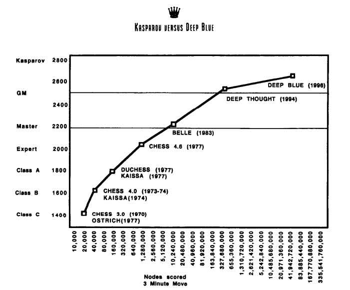


Figure 6.23. Relationship between the level of play by chess programs

Rybka: Computer Chess

Rybka is a computer chess engine designed by International Master Vasik Rajlich.
- As of February 2011, Rybka is one of the top-rated engines on chessengine rating lists and has won many computer chesstournaments.
- Rybka won four consecutive World Computer Chess Championships from 2007 to 2010, but it was stripped of these titles in June 2011 because that Rybka plagiarized code from both the Crafty and the Fruit chess engines. Others dispute this.
- Rybka 2.2n2 is available as a free download and Deep Rybka 3 is ranked first among all computer chess programs in 2010
- Rybka uses a bitboard representation, and is an alpha-beta searcher with a relatively large aspiration window. It uses very aggressive pruning, leading to imbalanced search trees.

Bitboard Representation for Chess

- There are 64 positions on a chessboard.
- There are six types of chess pieces: Pawn, Bishop, Knight, Rook, Queen, and King.
- One 64-bit vector for each type and each side: 12 64-bit vectors are needed.
- Legal moves of each type can also be represented by 64-bit vectors.

Zappa: Computer Chess

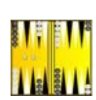
- Zappa is a chess engine written by Anthony Cozzie, a graduate student at the University of Illinois atUrbana-Champaign.
- The program emphasizes sound search and a good use of multiple processors. Zappa scored an upset victory at the WorldComputer Chess Championship in August, 2005, in Reykjavmk, Iceland. Zappa won with a score of 10 out of 11, and beat both Juniorand Shredder, programs that had won the championship many times.
- Zappa's other tournament successes include winning CCT7 and defeating Grandmaster Jaan Ehlvest 3-1. In Mexico in September 2007 Zappa won a matchagainst Rybka by a score of 5.5 to 4.5.
- In March 2008 Anthony Cozzie announced that ``the Zappa project is100% finished", which includes both tournaments and future releases.
- In June 2010, Zach Wegner announced that he had acquired the rights to maintain and improve the Zappa engine. The improved engine competed in the 2010 WCCC under the name Rondo, achieving second place behind Rybka.

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a pre-computed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

2007: proved to be a draw! Schaeffer et al. solved checkers for "White Doctor" opening (draw) (about 50 other openings).

Othello: human champions refuse to compete against computers, who are too strong.



Backgammon: TD-Gammon is competitive with World Champion (ranked among the top 3 players in the world). Tesauro's approach (1992) used learning to come up with a good evaluation function. Exciting application of reinforcement learning.

Summary

Game systems rely heavily on

- Search techniques
- Heuristic functions
- Bounding and pruning techniques
- Knowledge database on game

For AI, the abstract nature of games makes them an appealing subject for study:

state of the game is easy to represent;
agents are usually restricted to a small number of actions whose outcomes are defined by precise rules

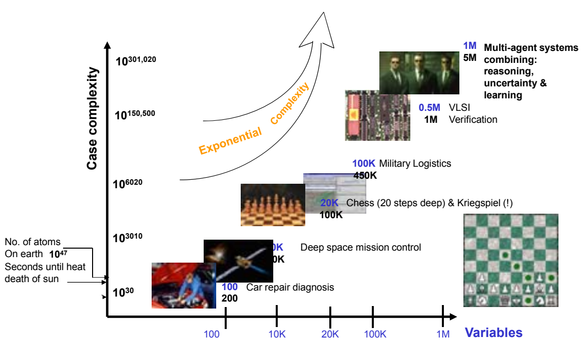
73

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.

74

Automated reasoning --- the path



75