Adversarial Search
(or Game Search)

Vincent A. Cicirello, Ph.D.
Professor of Computer Science
cicirelv@stockton.edu https://www.cicirello.org/

STOCKTON UNIVERSITY www.stockton.edu

1

Lesson 1

GAME SEARCH TERMINOLOGY

STOCKTON UNIVERSITY www.stockton.edu

2

Two-player zero-sum discrete finite deterministic games of perfect information

- **Two player:** well, there are two players…
- **Zero Sum:** In any outcome of any game Player A's gains equals Player B's losses.
- **Discrete:** All game states and decisions are discrete values.
- **Finite:** There are only a finite number of states and decisions.
- **Deterministic:** no chance… no dice rolls… etc
- **Games:** defined shortly….
- **Perfect information:** Both players can see the state, and each decision is made sequentially.

STOCKTON UNIVERSITY www.stockton.edu

3

A Game Defined

- A two-player zero-sum discrete finite deterministic game of perfect information is a quintuplet, (S, I, Succs, T, V) where:
  - S: Finite set of states (must include sufficient information to deduce whose turn it is to move next)
  - I: Initial state
  - Succs: Function that takes a state as input and returns a set of states (legal positions after a move).
    - Must be non-empty if its argument is not a terminal state
  - T: Set of terminal states (i.e., states when game ends and payoff occurs)
  - V: Maps terminal states to real values (player A's gain / player B's loss)

STOCKTON UNIVERSITY www.stockton.edu

4

Example: Nim

- You begin with some number of piles of matches.
- During a turn, the player may remove any number of matches from one pile
- The last person to remove a match loses
- In II-Nim, you begin with two piles each with two matches
- **States of Nim**
  - A(jj,jj); A(j,jj); A(,jj); A(jj,j); A(jj,); A(j,j); A(,j); A(j,); A(,)
  - B(jj,jj); B(j,jj); B(,jj); B(jj,j); B(jj,); B(j,j); B(,j); B(j,); B(,)

STOCKTON UNIVERSITY www.stockton.edu

5

Common Trick: Exploit Symmetry

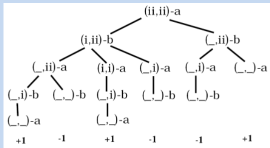
- **States of Nim**
  - A(jj,jj); A(j,jj); A(,jj); A(jj,j); A(jj,); A(j,j); A(,j); A(j,); A(,)
  - B(jj,jj); B(j,jj); B(,jj); B(jj,j); B(jj,); B(j,j); B(,j); B(j,); B(,)
- **Common Trick: Symmetry**
  - Some states are trivially equivalent (e.g., A(,jj); A(jj,))
  - Use some canonical description to make them one state
    - e.g., right pile always has at least as many matches as right
- **States of Nim Using Symmetry**
  - A(jj,jj); A(j,jj); A(,jj); A(j,j); A(,j); A(,)
  - B(jj,jj); B(j,jj); B(,jj); B(j,j); B(,j); B(,)

STOCKTON UNIVERSITY www.stockton.edu

6

Nim Formalized

- S = {A(jj,jj); A(j,jj); A(,jj); A(j,j); A(,j); A(,); B(jj,jj); B(j,jj); B(,jj); B(j,j); B(,j); B(,)}
- I = A(jj,jj)
- T = {A(,); B(,)}
- V(A(,)) = +1; V(B(,)) = -1
- Succs(A(jj,jj)) = { B(j,jj); B(,jj) }
- Succs(B(j,jj)) = { A(,jj); A(j,j); A(,j) }
- Succs(B(,jj)) = { A(,j); A(,) }
- etc.



7

Lesson 2

GAME THEORETIC VALUES AND SOLVING GAMES

8

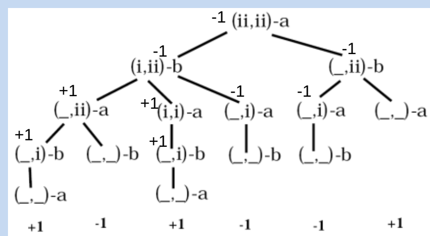
Game Theoretic Value

- We usually assume that all players are perfectly rational
  - Perfect rationality = always chooses action that maximizes expected value of outcome given what we know
  - Useful assumption even though not met in practice
- **Game theoretic value:** The game theoretic (or minimax) value of a game state is the value of the terminal state that will be reached if both players play optimally.
- Game theoretic value of a terminal state is simply the value of the state.
- How do we find the game theoretic value of non-terminal states?

9

Game Theoretic Value

- How do we find the game theoretic value of non-terminal states?
- Idea: fill in the tree bottom-up.



10

The Minimax Algorithm

- Generate the full Game tree, storing it in memory
- Run through all of the terminal states assigning them values
- Run through all predecessors assigning them values, etc, etc, etc…
- Question: Do we really need to store the whole game tree in memory?
  - NO… Use DFS

```
Minimax-Value(S)
    if (S is a terminal)
        return V(S)
    else
        Let S1,S2,…Sk = Succs(S)
        Let vi=Minimax-Value(Si)
        for each Si.
        If PlayerToMove(S) = A
            return Max(vi)
        else
            return Min(vi)
```

11

Dynamic Programming (DP)

- Dynamic Programming---Russell & Norvig's definition:
  - "solutions to subproblems are constructed incrementally from those of smaller subproblems and are cached to avoid recomputation"
- You may have encountered this in other classes (e.g., if you've taken Data Structures & Algorithms II).
  - E.g., Floyd-Warshall for All Pairs Shortest Paths is a dynamic programming algorithm

12

Solving Games

- Solving a Game means determining the game theoretic value for the initial state (i.e., determining who will win if both players play optimally)
- Running the minimax algorithm on the initial state will solve a game:
  - Minimax requires that $O(B^L)$ states are expanded (worst case and best case), where B is the average branching factor, and L is the usual length of the game.
  - DFS on simple search problems can get lucky and find a solution quickly but Minimax is a DFS of the entire game tree, so always $O(B^L)$
- What if the total number of game states, N, is much less than B^L ?
  - E.g., for chess, $N = 10^{40}$, while $B^L = 10^{120}$
- In such cases, DP is a better method, assuming you can afford the memory.
  - Runtime of DP for game solving: $O(NL)$. Memory: $O(N)$.

13

Example: DP for Chess Endgames

- Playing the endgame of a game of chess is easy.
  - Chess playing computers use an endgame database that stores the "solutions" for all game states with fewer than a certain number of pieces left on the board.
  - Playing the endgame simply involves database lookups.
- Dynamic programming for generating an endgame database
  - Consider that there are only 4 chess pieces left on the board.
  - With sufficient computational resources, you can compute, for all possible positions, whether it is a win for black, white, or a draw.
  - Details next slide….

14

DP for Chess Endgames

- Assume there are N positions with no more than 4 pieces left:
  1. Define a 1-to-1 mapping from the N board positions to the integers 0..N-1
  2. Create an of length N (2 bits per entry). Each element can take on one of three values:
     - W: White will eventually win.
     - B: Black will eventually win
     - ?: We don't know who wins from this state
  3. Mark all terminal states with their values, W or B.
  4. Look through all states still marked by "?"
     - If W is about to move, then
       - if all successors are marked with B, mark the state B
       - if any successor state is marked W, then mark the state W
       - else leave the state unchanged (marked "?")
     - if B is about to move, then
       - if all successors are marked with W, mark the state W
       - if any successor state is marked B, then mark the state B
       - else leave the state unchanged (marked "?")
  5. If 4 changed the label of at least one state, then repeat 4.
  6. Any state still marked with "?" is a state from which no one can force a win---thus a draw.

15

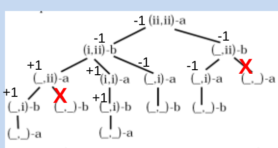
Lesson 3

ALPHA BETA PRUNING

16

Cutting Off Unnecessary Game States

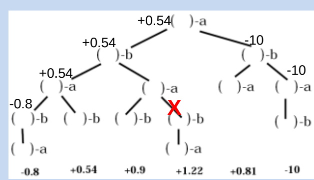
- We're now returning to minimax with DFS.
- Minimax searches the entire game tree with DFS.
- So we really need to search the entire game tree?
- If we knew that the only possible game values were +1 and -1, can we save computation?
  - Yes (a lot actually, although not much in this example)
  - If any successor is a forced win for the current player, then don't bother expanding any further successors.



17

What if possible terminal values are unknown?

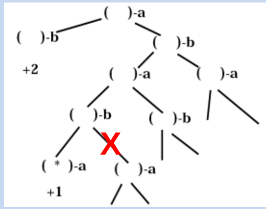
- Do DFS, but if something is discovered that implies your parent would not choose you, then don't bother expanding further successors.
- More generally, not just your parent, but any ancestors.



18

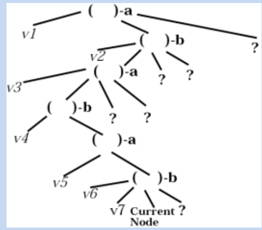
An ancestor causing cutoff….

- Suppose we've done a full DFS, expanding left-most successors first and that we are currently at the search state marked by the *
- What can we cut off in the rest of the search?
- If either player has a better alternative at an ancestor of a given search node, then it will not be visited.



19

A general cutoff rule



- In this example:
  - let α=max(v1,v3,v5)
  - let β=min(v6,v7)
  - if β <= α, then we can be certain that it is a waste of time searching the "current node" or its sibling to the right
- In general:
  - if at a B-move node,
    - let α=max of all A's choices on current path
    - let β=min of all B's choices including those at current node
    - Cut-off if β<= α
  - Converse rule if at an A-move node

20

Alpha Beta Pruning

```
function Max-Value(s,α,β)
inputs:
   s: current state in game, A about to play
   α: best score (highest) for A along path to s
   β: best score (lowest) for B along path to S
output: min(β , best-score (for A) available from s)

   if ( s is a terminal state )
   then return ( terminal value of s )
   else for each s' in Succs(s)
            α := max( α , Min-value(s',α,β) )
            if ( α >= β ) then return β
      return α
```

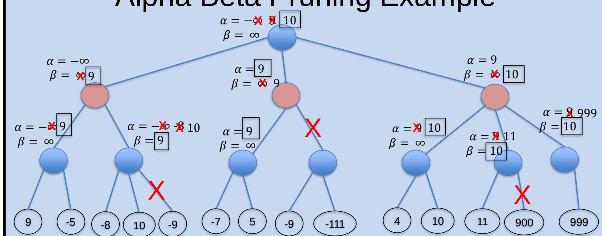
```
function Min-Value(s,α,β)
output: max(α , best-score (for B) available from s)

   if ( s is a terminal state )
   then return(terminal value of S)
   else for each s' in Succs(s)
            β:= min( β, Max-value(s',α,β) )
            if ( β <= α) then return α
      return β
```

- Alpha-Beta Pruning from Russell & Norvig.
- Assumes players alternate moves.
- Top level call: **Max-Value(I,-∞ ,+ ∞)**

21

Alpha Beta Pruning Example



22

How Useful is Alpha Beta Pruning?

- What is the best case performance of alpha-beta?
- How much of the tree would you examine if you were very lucky in the order you tried successors?
- Best case:
  - The number of nodes you need to search in the tree is $O(B^{L/2})$.
  - The square root of the recursive minimax cost.
  - Large real-sized games with a huge number of states are still problematic (e.g., chess).

23

Lesson 4

FROM GAME SOLVING TO GAME PLAYING

24

A Few Solved Games

- So far, we've entirely focused on game solving (determining what the outcome of a game will be if both players play optimally).
- **Solving a game** means proving the game-theoretic value of the start state.
- A few solved games:
  - By brute-force dynamic programming
    - Four-in-a-row
    - Chess endgames (certainly NOT chess itself though)
  - By brute-force DP to create endgame database, plus alpha-beta search
    - nine men's morris
  - By brute-force DP plus game specific analysis
    - Connect-Four

STOCKTON UNIVERSITY www.stockton.edu

25

Checkers has been Solved: Draw

- The largest game that has been solved is Checkers.
- There are approximately $5 * 10^{20}$ states in the game of checker.
- Checkers was solved by a system called Chinook.
  - Mostly brute-force Dynamic Programming
  - Dozens of computers during the years 1989-2007 (18 years, with a couple year break in the middle)
  - First computer program in any game that won the right to play in a human world championship (1990).
  - Lost in 1992, but became the world champion in 1994.
  - Retired from playing in human tournaments in 1996 (clear no human could win).
  - Game theoretic value of checkers is 0 (a draw).

STOCKTON UNIVERSITY www.stockton.edu

26

Game Playing vs Game Solving

- Two very different activities
- Game Solving: finding the true game-theoretic value of a state.
- What about game playing?
- Game solving often very different from playing a game well.
- Example, what do real chess playing programs do?
- Some features that the search algorithms covered so far don't have:
  - Cannot possibly find a guaranteed solution (not enough time).
  - Must make decisions quickly in real-time.
  - It is not possible to pre-compute a solution.

STOCKTON UNIVERSITY www.stockton.edu

27

Heuristic Evaluation Functions

- Popular solution: use heuristic evaluation functions
- An evaluation function maps a state to a real value.
  - The higher the evaluation, the higher the true game-theoretic value is estimated to be.
- Note: this is not the same as the heuristic in A*…
  - no notion of admissibility
  - not an estimate of path cost to reach a goal
- Search the game tree as deeply as time allows
- Leaves of tree you search are not leaves of game tree, but are intermediate nodes
- The values assigned to leaves are from the heuristic evaluation function

STOCKTON UNIVERSITY www.stockton.edu

28

Heuristic Evaluation Intuition

- **Visibility:**
  - Evaluation function will be more accurate nearer the end of the game.
  - So worth using heuristic estimates from there.
- **Filtering:**
  - If we used the evaluation function without searching, we'd be using a handful of inaccurate estimates (near the root).
  - By searching, we're combining thousands of these estimates, hopefully eliminating noise.
- Can give counter-examples… But often works very well in practice for real games…

STOCKTON UNIVERSITY www.stockton.edu

29

Heuristic Evaluation Example

- A simple heuristic for chess:
  - The typical introductory chess book heuristic: a bishop or knight worth the value of 3 pawns; a rook worth 5; a queen worth 9
  - This leads to a simple **weighted linear evaluation** function
- More sophisticated chess heuristics consider other state features:
  - good pawn structure might be worth value of a pawn
  - "king safety" might be worth a pawn
- Or **nonlinear evaluation functions** are possible:
  - two bishops might be worth slightly more than twice the value of a single bishop
  - a bishop near the end of the game may be worth more than earlier in the game (e.g., more powerful in open space)
- Machine learning also applicable here
  - E.g., to learn a heuristic (then use alpha-beta pruning search)

STOCKTON UNIVERSITY www.stockton.edu

30

Monte Carlo Tree Search

- Monte Carlo Tree Search, an alternative to using a heuristic
- Common for the game of Go, as well as games with chance.
- Pure Monte Carlo Tree Search:
  - Let's say player has N alternative moves they can make.
  - Simulate making move 1, and then both players playing randomly to the end of the game, record the outcome (1 for win, -1 for loss, 0 for draw), repeat this process k times, and average the outcomes.
  - Repeat for move 2, move 3, … move N.
  - Pick the move that has the highest average outcome.
- One simple improvement to Pure Monte Carlo Tree Search:
  - Run Alpha-Beta Pruning search to depth D. If the states at that depth are not terminal, then use Monte Carlo Tree Search to evaluate the states at depth D.

STOCKTON UNIVERSITY www.stockton.edu

31

Some Other Issues for Game Playing

- How to determine how far to search if you only have a fixed time to make a decision?
- **Quiescence:** What if you stop the search at a state where subsequent moves drastically change the evaluation?
  - e.g., you search to depth d in chess, but at depth d+1, a queen is taken…
- **Quiescence search:** an extra bit of search to attempt to reach a quiescent state
  - e.g., in chess, continue search only considering "capture" moves to resolve any uncertainties in position

STOCKTON UNIVERSITY www.stockton.edu

32

More Game Playing Issues

- **The horizon problem:**
  - Consider a state in which it is inevitable that your opponent will be able to do something bad to you.
    - e.g., an inevitable queening of a pawn
  - Now consider that you have some delaying tactics.
  - The search algorithm won't recognize the inevitable if the number of delaying steps exceeds the search depth limit…
  - Thus not recognizing the badness of the search state.
- **Endgames:** Are easy to play well. How?
  - An end game database
    - essentially a lookup table (e.g., generated by DP)
- **Openings:** Are easy to play well. How?
  - An opening book
  - e.g., for chess, based on hundreds of years of human chess playing knowledge

STOCKTON UNIVERSITY www.stockton.edu

33

Putting it all together….

```
MaxValue(s, α, β, d)
  if s is a terminal state
     return V(s)
  else if s is in endgame DB
     return Endgame(s)
  else if d = 0
     return Heuristic(s)
  for each s' in Succs(s)
     α=max(α, MinValue(s',α,β,d-1)
     if α >= β then return β
  return α
```

```
MinValue(s, α, β, d)
  if s is a terminal state
     return V(s)
  else if s is in endgame DB
     return Endgame(s)
  else if d = 0
     return Heuristic(s)
  for each s' in Succs(s)
     β=min(β, MaxValue(s',α,β,d-1)
     if α >= β then return α
  return β
```

STOCKTON UNIVERSITY www.stockton.edu

34

How to choose a move

```
ChooseMovePlayerA(s,d)
  bestMove = null
  bestValue = -∞
  if s is in opening book
     return opening(s)
  for s' in Succs(s)
     v = MinValue(s',-∞,∞,d-1)
     if v > bestValue
        bestValue = v
        bestMove = s'
  return bestMove
```

```
ChooseMovePlayerB(s,d)
  bestMove = null
  bestValue = ∞
  if s is in opening book
     return opening(s)
  for s' in Succs(s)
     v = MaxValue(s',-∞,∞,d-1)
     if v < bestValue
        bestValue = v
        bestMove = s'
  return bestMove
```

STOCKTON UNIVERSITY www.stockton.edu

35

Lesson 5

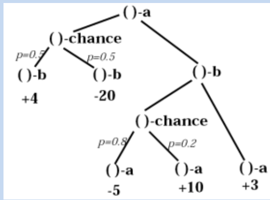
GAMES INVOLVING CHANCE

STOCKTON UNIVERSITY www.stockton.edu

36

2-player zero-sum finite nondeterministic games of perfect information

- The game tree now includes states in which neither player makes a choice.
- Instead, a random decision is made according to a known set of outcome probabilities.
- Game-theoretic value is the **expected** final outcome if both players are optimal.



37

Expectiminimax

- Obvious generalization of minimax:
  - Expectiminimax(s) =
    - Value(s) if s is a terminal state
    - max{s' in succs(s)} Expectiminimax(s') if s is a player A node
    - min{s' in succs(s)} Expectiminimax(s') if s is a player B node
    - Sum{s' in succs(s)} P(s') Expectiminimax(s') if s is a chance node
- Can we use alpha-beta pruning? yes
  - for Min and Max nodes it works unchanged
  - for chance nodes, if we have a bound on terminal values
    - compute an upper bound on the value of a chance node without looking at all of its children

38

Bad News for Expectiminimax

- Assume a game with dice rolls.
- Expectiminimax considers all possible dice roll sequences, then it is:
  - $O(b^m n^m)$ where n is the number of distinct dice rolls
- Example: Backgammon
  - n=21
  - b usually 20, but as high as 4000 for dice rolls that are doubles
  - can probably only manage about m=6 (3 moves each player)
- The equivalent of Alpha-Beta pruning helps the situation a bit but not much
- State-of-the-art Backgammon programs rely heavily on sophisticated evaluation heuristics utilizing machine learning techniques

39

TD-Gammon (Gerald Tesauro, 1992)

- Machine learning to learn a heuristic board evaluation function
  - A neural network trained by TD-lambda
    - TD-lambda: A form of "temporal difference learning", a type of reinforcement learning.
- TD-Gammon stopped improving performance after 1.5 million games against itself.
- Only 2-ply look ahead used.
- Reached level equivalent to the top human players at the time.
- Made human expert players rethink how to play
  - Discovered strategies that human players never considered or mistakenly thought bad
  - Tournament players changed how they play
  - Began using TD-gammon to study and learn new strategies
- Strong positional player, sometimes plays poorly in end game

40