Announcements

- HW5 now available!
  - May do in groups of two.
  - Review in recitation
  - No fancy data structures except trie!
  - Due Monday 11:59 pm

Backtracking

- An algorithm-design technique
- "Organized brute force"
  - Explore the space of possible answers
  - Do it in an organized way
  - Example: maze
    - Try S, E, N, W

Backtracking decisions

- Develop answers by identifying a set of successive decisions.
  - Maze
    - Where do I go now: N, S, E or W?
  - 8 Queens
    - Where do I put the next queen?
  - Knight’s tour
    - Where do I jump next?

Backtracking is useful ...

- when a problem is too hard to be solved directly
  - 8-queens problem
  - Knight’s tour
- when we have limited time and can accept a good but potentially not optimal solution
  - Game playing (second part of lecture)
  - Planning (not in this class)
Backtracking decisions

- Develop answers by identifying a set of successive decisions.

  - **Pure** Backtracking
    - Decisions can be binary: ok or impossible
      - Cannot put an "X" in two places
      - Can move the rook two spaces forward

- **Heuristic** Backtracking
  - Decisions can have goodness:
    - Good to sacrifice a pawn to take a bishop
    - Bad to sacrifice the queen to take a pawn
    - Bad not to block in tic-tac-toe

Backtracking technique

- When you reach a dead-end
  - Withdraw the most recent choice
  - Undo its consequences

- When you reach a dead-end
  - Withdraw the most recent choice
  - Is there a new choice?
  - If not, you are at another dead-end
Backtracking technique

- When you reach a dead-end
  - Withdraw the most recent choice
  - Undo its consequences
  - Is there a new choice?
    - If so, try that

Search trees

- The tree
  - Is rarely explicit as a data structure!
- How do we keep track of our search?
  - Use a stack ("control stack")
    - Represents a stack of tentative decisions
    - Tracks decisions to be undone when backtracking.
    - Can be implicit in recursive calls

No optimality guarantees

- If there are multiple solutions, simple backtracking will find one of them, but not necessarily the optimal.
- No guarantees that we’ll reach a solution quickly.
Backtracking Summary

- Organized brute force
- Formulate problem so that answer amounts to taking a set of successive decisions
- When stuck, go back, try the remaining alternatives
- Does not give optimal solutions

Why Make Computers Play Games?

- People have been fascinated by games since the dawn of civilization.
- Idealization of real-world problems containing adversary situations
  - Typically rules are very simple
  - State of the world is fully accessible
  - Example: auctions

What Kind of Games?

- No, not Quake (although interesting research there, too)
- Simpler Strategy/Board Game
  - Chess
  - Checkers
  - Othello
  - Go
- Let’s look at how we might go about playing Tic-Tac-Toe

Consider this position

- We are playing X, and it is now our turn.

Let’s write out all possibilities

- Each number represents a position after each legal move we have.
Now let’s look at their options

Here we are looking at all of the opponent responses to the first possible move we could make.

Now let’s look at their options

Opponent options after our second possibility. Not good again...

Now let’s look at their options

Struggling...

More interesting case

Now they don’t have a way to win on their next move. So now we have to consider our responses to their responses.

Our options

We have a win for any move they make. So the original position in purple is an X win.

Finishing it up...

They win again if we take our fifth move.
Summary of the Analysis

So which move should we make? ;-)

A Tic Tac Toe Game Tree

A path in a game tree

Properties of Tic Tac Toe

- Two players
- Deterministic
  - no randomness, e.g., dice
- Perfect information
  - No hidden cards or hidden Chess pieces...
- Finite
  - Game must end in finite number of moves
- Zero sum
  - Total winnings of all players

Games Not Considered

- Poker
  - Hidden cards
- Backgammon
  - not deterministic (e.g., dice)
- Prisoner’s dilemma, game of chicken
  - Not zero-sum

Have rich and beautiful theories and algorithms, though. (see CS 15-859)
**Two-player games**

- We can define the **value** (goodness) of a certain game state (board).

```
1 0 -1 ?
```

- What about the non-final board?
  - Look at board, assign value
  - Look at children in game tree, assign value

**How do we play?**

- Traverse the “game tree”.
  - Enumerate all possible moves at each node.
- Start by evaluating the terminal positions, where the game is over.
- Propogate the values up the tree, assuming we pick the best move for us and the opponent picks the best move for her.

**How to play?**

- What is the rule to compute the value of nonterminal node?
  - If it is our turn (X) then we pick the maximum outcome from the children.
  - If it is the opponent's turn (O) then we pick the minimum outcome from the children.

- This process is known as the **Minimax** algorithm.
- We play the move that gave us the maximum at the root.

**More generally**

- Player A (us) **maximize** goodness.
- Player B (opponent) **minimizes** goodness

**Games Trees are useful...**

- Provide “**lookahead**” to determine what move to make next
- Build whole tree = we know how to play perfectly
But there’s one problem...

- Games have **large search trees:**
  - **Tic-Tac-Toe**
    - There are 9 ways we can make the first move and opponent has 8 possible moves he can make.
    - Then when it is our turn again, we can make 7 possible moves for each of the 8 moves and so on.
    - $9! = 362,880$

Or chess...

- Suppose we look at 16 possible moves at each step
- And suppose we explore to a depth of 50 turns
- Then the tree will have $16^{50} = 10^{120}$ nodes!
- DeepThought (1990) searched to a depth of 10

So...

- Need techniques to avoid enumerating and evaluating the whole tree
- **Heuristic search**

Heuristic search

1. Organize search to **eliminate** large sets of possibilities
   - **Chess:** Consider major moves first
2. Explore decisions **in order** of likely success
   - **Chess:** Use a library of known strategies
3. Save time by **guessing** search outcomes
   - **Chess:** Estimate the quality of a situation
     - Count pieces
     - Count pieces, using a weighting scheme
     - Count pieces, using a weighting scheme, considering threats

Let’s see it in action

**Mini-Max Algorithm**
Minimax, in reality

- Rarely do we reach an actual leaf
  - Use **estimator functions** to statically guess goodness of missing subtrees

Minimax algorithm - pseudo code

```
Max( P, depth ) {
  if( depth == 0 ) return estimator(P)
  m = -\infty
  for each possible move P'
    m = max( m, Min(P', depth-1) )
  return m
}

Min( P, depth ) {
  if( depth == 0 ) return estimator(P)
  m = \infty
  for each possible move P'
    m = min( m, Max(P', depth-1) )
  return m
}
```

NegaMax Algorithm

- Rewrite from the point of view of moving player:

```
negaMax( P, depth ) {
  if (depth == 0) return estimator(P)
  m = -\infty
  for each possible move P'
    v = -negaMax( P', depth-1 )
    if m < v then m = v
  return m
}
```

Can you prove these two versions are the same?
How fast?

- Minimax is pretty slow even for a modest depth.
- It is basically a brute force search.
- What is the running time?
  - Each level of the tree has some average \( b \) moves per level. We have \( d \) levels. So the running time is \( O(b^d) \).

Can we speed this up?

- Let us observe the following game tree.

\[
\begin{array}{c}
\text{Max} \\
2 \\
\text{Min} \\
2 \\
\text{Max} \\
7 \\
1 \\
\end{array}
\]

What do we know about the root?
What do we know about the root’s right child?

Pruning

- Minimax sometimes evaluates nodes that have no impact on the search.
- E.g., Evaluating moves in Chess
  - My 1st move: I’m ahead a rook
  - My 2nd move: Opponent 1st move leaves me down a queen.
    - No need to look at Opponent’s 2nd move to find out it is checkmate. Already know my 1st move is better.

Alpha Beta Pruning

- Idea: Track “window” of expectations. Two variables:
  - \( \alpha \) – Best score so far at a \textbf{max} node: increases
    - At a child \textbf{min} node: Parent wants \textbf{max}. To affect the parent’s current \( \alpha \), our \( \beta \) cannot drop below \( \alpha \).
  - \( \beta \) – Best score so far at a \textbf{min} node: decreases
    - At a child \textbf{max} node.
      - Parent wants \textbf{min}. To affect the parent’s current \( \beta \), our \( \alpha \) cannot get above the parent’s \( \beta \).
  - Either case: If \( \alpha = \beta \):
    - Stop searching further subtrees of that child. They do not matter!
  - Start the process with an infinite window (\( \alpha = -\infty \), \( \beta = \infty \)).

Alpha-Beta algorithm - pseudo code

MaxAB( \( P \), depth, \( \alpha \), \( \beta \) ) {
    if( depth == 0 ) return estimator(\( P \))
    for each possible move \( P' \) \&\& \( \alpha < \beta \)
        \( \alpha = \max(\alpha, \text{MinAB}( P', \text{depth-1}, \alpha, \beta )) \)
    return \( \alpha \)
}

MinAB( \( P \), depth, \( \alpha \), \( \beta \) ) {
    if( depth == 0 ) return estimator(\( P \))
    for each possible move \( P' \) \&\& \( \alpha < \beta \)
        \( \beta = \min(\beta, \text{MaxAB}( P', \text{depth-1}, \alpha, \beta )) \)
    return \( \beta \)
}
Alpha-Beta algorithm - Alternative

\[ \text{AB}(P, \text{depth}, \alpha, \beta) \{
    \text{if} (\text{depth} == 0) \text{ return estimator}(P)
    \text{for each possible move } P' \text{ if } \alpha < \beta
    v = -\text{AB}(P', \text{depth}-1, -\beta, -\alpha)
    \text{if } \alpha < v \text{ then } \alpha = v
    \text{return } \alpha
\} \]

- In MinAB replace \( \alpha \) with \(-\beta\) and \( \beta \) with \(-\alpha\), and negate the return value. Get MaxAB.

---

Alpha Beta Example

- Does Alpha Beta ever return a different root value than Minimax?
  - No! As long as the root value is within the root (Alpha, Beta) range.
  - Alpha Beta does the same thing Minimax does, except it is able to detect parts of the tree that make no difference.

---

Alpha Beta Pruning

- Theorem: Let \( v(P) \) be the value of position \( P \). Let \( X \) be the value returned by \( \text{AB}(\alpha, \beta) \). Then one of the following holds:
  \[ v(p) \leq \alpha \ \text{and} \ X \leq \alpha \]
  \[ \alpha < v(P) < \beta \ \text{and} \ X = v(P) \]
  \[ \beta \leq v(P) \ \text{and} \ X \geq \beta \]

- The proof is by induction, and is an easy but tedious case analysis.
Alpha Beta speedup

- **Claim:** The optimal Alpha Beta search tree is $O(b^{d/2})$ nodes or the square root of the number of nodes in the regular Minimax tree.
  - Can enable twice the depth
- The speedup is *greatly* dependent on the order in which you consider moves at each node. Why?

Summary

- **Backtracking**
  - Organized brute force
  - Answer to problem = set of successive decisions
  - When stuck, go back, try the remaining alternatives
  - No optimality guarantees

- **Game playing**
  - Game trees
  - Mini-max algorithm
  - Optimization
    - Alpha-beta pruning
    - Heuristics search