Game Playing – Summary So Far

- Game tree
  - describes the possible sequences of play
  - is a graph if we merge together identical states
- Minimax:
  - utility values assigned to the leaves
  - Values “backed up” the tree by
  - MAX node takes max value of children
  - MIN node takes min value of children
  - Can read off best lines of play
- Depth Bounded Minimax
  - utility of terminal states estimated using an “evaluation function”

Game Playing – Beyond Minimax

- Efficiency of the search
  - Game trees are very big
  - Evaluation of positions is time-consuming
- How can we reduce the number of nodes to be evaluated?
  - “alpha-beta search”
- Bounding the depth of minimax has deficiencies
  - Why?
  - How can we mitigate these deficiencies?

Game Playing – Improving Efficiency

- Suppose that we are doing depth-bounded minimax
- We have a game tree to create and then insert the minimax values in order to find the values for our possible moves from the current position

Game Playing – Minimax using DFS

- The presentation of minimax was done by “backing up from the leaves” – a “bottom-up” breadth-first search.
  - This has the disadvantage of taking a lot of space
  - Compare this with the space usage issues for DFS vs. BFS in earlier lectures
  - If we can do minimax using DFS then it is likely to take a lot less space
  - Minimax can be implemented using DFS
  - But reduced space is not the only advantage:

Pruning

- What is pruning?
  - The process of eliminating a branch of the search tree from consideration without examining it.
- Why prune?
  - To eliminate searching nodes that are potentially unreachable.
  - To speedup the search process.
Alpha-beta pruning

1. In chess, can only search full-width tree to about 4 levels
2. The trick is to “prune” subtrees
3. Fortunately, the best move is provably insensitive to certain subtrees
4. If there exists a winning move at a node, then its sibling nodes (and their subtrees) need not be examined.

Alpha-Beta Pruning

1. A particular technique to find the optimal solution according to a limited depth search using evaluation functions.
2. Returns the same choice as minimax cutoff decisions, but examines fewer nodes.
3. Gets its name from the two variables that are passed along during the search which restrict the set of possible solutions.

Definitions

1. Use bounds on minimax values to prune subtrees
2. Bounds on MAX nodes are called **alpha values** and are initially -inf. The alpha value for a MAX node is the value of its best successor and can never decrease.
3. Bounds on MIN nodes are called **beta values** and are initially inf. The beta value for a MIN node is the value of its worst successor and can never increase.
4. Alpha cutoff: A MIN node can be pruned if its beta value is £ the alpha value of its MAX parent
5. Beta cutoff: A MAX node can be pruned if its alpha value £ the beta value of its MIN parent.

Game Playing – Pruning nodes

1. If we are scanning the tree using DFS then there was no point in evaluating node K
2. Whatever the value of K there cannot be any rational sequence of play that would go through it
3. Node K can be pruned from the search: i.e. just not selected for further expansion
4. “At node B then MIN will never select E, because J is better than D for MAX and so MIN must not allow MAX to have that opportunity”
5. Q. So what! It’s just one node?
6. A. Suppose that the depth limit were such that K was far from the depth bound. Then evaluating K corresponds to a large sub-tree. Such prunings can save an exponential amount of work.

Game Playing – Improving Efficiency

1. Suppose that we were doing Breath-First Search, would you still be able to prune nodes in this fashion?
2. NO! Because the pruning relied on the fact that we had already evaluated node D by evaluating the tree underneath D
3. This form of pruning is an example of “alpha-beta pruning” and relies on doing a DEPTH-FIRST search of the depth bounded tree
Suppose that nodes K and J were evaluated in the opposite order can we expect that we would be able to do a similar pruning? The answer depends on the value of K. Suppose that K had a value of 2 and is expanded first:

On discovering util(D) = 6 we know that util(B) <= 6.
On discovering util(K) = 2 we know that util(E) >= 2.
Can NOT stop expansion of E as best play might still go via E.
Value of J is relevant – no pruning.

STOP! What else can you deduce now? Can NOT stop expansion of E as best play might still go via E.
Value of J is relevant – no pruning.

When K had a value of 2 and was expanded first then we did not get to prune a child of E.
To maximise pruning we want to first expand those children that are best for the parent.
Cannot know which ones are really best.
Use heuristics for the “best-first” ordering.
If this is done well then alpha-beta search can effectively double the depth of search tree that is searchable in a given time.
Effectively reduces the branching factor in chess from about 30 to about 8.
This is an enormous improvement!

The games are symmetric so is natural that we can also do a similar pruning with the MIN and MAX roles reversed.
The reasoning is identical other than for the reversal of roles.
Can deduce that some other nodes can not be involved in the line of best play.

The diagrams illustrate the node ordering and pruning process in a game tree.
The pruning was based on using the results of the “DFS so far” to deduce upper and lower bounds on the values of nodes. Conventionally these bounds are stored in terms of two parameters:

- **α** values are stored with each MAX node and each MAX node is given a value of alpha that is the current best lower-bound on its final value.
  - Initially is $-\infty$ to represent that nothing is known.
  - As we do the search then α at a node can increase, but it can never decrease — it always gets better for MAX.

- **β** values are stored with each MIN node and each MIN node is given a value of beta that is the current best upper-bound on its final value.
  - Initially is $+\infty$ to represent that nothing is known.
  - As we do the search then β at a node can decrease, but it can never increase — it always gets better for MIN.

### Implementation

- Set root node alpha to negative infinity and beta to positive infinity.
- Search depth first, propagating alpha and beta values down to all nodes visited until reaching desired depth.
- Apply evaluation function to get the utility of this node.
- If parent of this node is a MAX node, and the utility calculated is greater than parents current alpha value, replace this alpha value with this utility.
- If parent of this node is a MIN node, and the utility calculated is less than parents current beta value, replace this beta value with this utility.

**Alpha-beta Pruning**

- If parent of this node is a MIN node, and the utility calculated is less than parents current beta value, replace this beta value with this utility.
- Based on these updated values, it compares the alpha and beta values of this parent node to determine whether to look at any more children or to backtrack up the tree.
- Continue the depth first search in this way until all potentially better paths have been evaluated.
Effectiveness of alpha-beta

- Alpha-beta pruning does not affect minimax value. It just improves efficiency of search.
- How much it improves efficiency of search depends on ordering of successors.
- With perfect ordering, can search twice as deep in given amount of time (i.e., effective branching factor is $\sqrt{b}$).
- Perfect ordering cannot be achieved, but simple ordering heuristics are very effective.

Effectiveness (cont.)

- Full-width minimax search in chess allows about 4-ply lookahead. With 4-ply lookahead, a chess program performs poorly and at the level of a human novice.
- For Deep Blue, alpha-beta pruning reduced the effective branching factor from about 35 to 6. This allows 8-ply lookahead, a chess program performs at the level of a human master.
- But Deep Blue can actually perform 12-ply lookahead.

Transposition tables

- Basic idea is caching: once position is evaluated, save in hash table to avoid re-evaluating.
- Called “transposition” tables because different orderings (transpositions) of same set of moves can lead to same position.
- Converts search tree to search graph (Chess game tree has approximately 35100 nodes while chess game graph has approximately 1040.)
- Deep Blue: huge transposition tables (100,000,000+) must be carefully managed.

Chance Nodes

- Many games that unpredictable outcomes caused by such actions as throwing a dice or randomizing a condition.
- Such games must include chance nodes in addition to MIN and MAX nodes.
- For each node, instead of a definite utility or evaluation, we can only calculate an expected value.

Games that Include an Element of Chance

Example: Depth = 4

Effectiveness of alpha-beta

- Alpha-beta pruning does not affect minimax value. It just improves efficiency of search.
- How much it improves efficiency of search depends on ordering of successors.
- With perfect ordering, can search twice as deep in given amount of time (i.e., effective branching factor is $\sqrt{b}$).
- Perfect ordering cannot be achieved, but simple ordering heuristics are very effective.
Inclusion of Chance Nodes

Calculating Expected Value

- For the terminal nodes, we apply the utility function.
- We can calculate the expected value of a MAX move by applying an expectimax value to each chance node at the same ply.
- After calculating the expected value of a chance node, we can apply the normal minimax-value formula.

Expectimax Function

- Provided we are at a chance node preceding MAX’s turn, we can calculate the expected utility for MAX as follows:
  - Let \( d_i \) be a possible dice roll or random event, where \( P(d_i) \) represents the probability of that event occurring.
  - If we let \( S \) denote the set of legal positions generated by each dice roll, we have the expectimax function defined as follows:
    \[
    \text{expectimax}(C) = \sum_i P(d_i) \max_{s \in S} (\text{utility}(s))
    \]
- Where the function \( \max_{s \in S} \) will return the move MAX will pick out of all the choices available.
- Alternately, you can generate an expectimin function for chance nodes preceding MIN’s move.
- Together they are called the expectiminimax function.

Application to an Example

- MAX
- Chance
- MIN

Complexity of Expectiminimax

- Where minimax does \( O(b^m) \), expectiminimax will take \( O(b^m n^m) \), where \( n \) is the number of distinct rolls.
- The extra cost makes it unrealistic to look too far ahead.
- How much this effects our ability to look ahead depends on how many random events that can occur (or possible dice rolls).
Games of chance

- Backgammon is a two-player game with uncertainty.
- Players roll dice to determine what moves to make.
- White has just rolled 5 and 6 and has four legal moves:
  - 5-10, 5-11
  - 5-11, 19-24
  - 5-10, 10-16
  - 5-11, 11-16
- Such games are good for exploring decision making in adversarial problems involving skill and luck.

Game Trees with Chance Nodes

- Chance nodes (shown as circles) represent random events.
- For a random event with N outcomes, each chance node has N distinct children, and probability is associated with each.
- (For 2 dice, there are 21 distinct outcomes)
- Use minimax to compute values for MAX and MIN nodes.
- Use expected values for chance nodes.
- For chance nodes over a max node, as in C:
  \[ \text{expectimax}(C) = \sum P(d_i) \times \max(\text{value}(i)) \]
- For chance nodes over a min node:
  \[ \text{expectimin}(C) = \sum P(d_i) \times \min(\text{value}(i)) \]

Meaning of the evaluation function

- Dealing with probabilities and expected values means we have to be careful about the “meaning” of values returned by the static evaluator.
- Note that a “relative-order preserving” change of the values would not change the decision of minimax, but could change the decision with chance nodes.
- Linear transformations are OK.

Wrapping Things Up

Game Playing – Deficiencies of Minimax

- The bound on the depth of search is artificial and can lead to many anomalies.
- We only consider two:
  1. Non-quiescence: “quiescent” = inactive, quiet, calm, …
  2. Horizon Effect
- (These deficiencies also apply to alpha-beta as it is just a more efficient way to do the same calculation as minimax)

Game Playing – Non-Quiescence

- Suppose that change depth bound from k to k+1 – i.e. expand one more move
- The values given to a node might change wildly
Quiescent and Secondary search

If a node represents a state in the middle of an exchange of pieces, the evaluation function may not give a reliable estimate of board quality. Example: after you capture a knight the evaluation may be good, but this is misleading if opponent is about to capture your queen.

Solution: if node evaluation is not “quiescent,” continue alphabeta search below that node but limit moves to those that significantly change evaluation function (e.g., capture moves, promotions). The branching factor for such moves is small.

This partially (but not completely) solves horizon problem
Deep Blue uses this technique to search some paths to depth 25, even though it ordinarily only searches to depth 14.

Game Playing – Quiescence Search

Suppose that change depth bound from k to k+1 – i.e. expand one more move
The values given to a node might change wildly

Keep on increasing the depth bound in that region of the game tree until the values become “quiescent” (“quiet”, i.e. stop being “noisy”)

Game Playing – Quiescence Search

In quiescence search the depth bound is not applied uniformly but adapted to the local situation
in this case so that the values are not wildly changing
Many other improvements to minimax also work by adapting to depth-bound to get better results and/or do less work

Game Playing – Horizon Effect

Sometimes there is a bad thing, “X”, such that
1. X cannot be avoided
2. X can be delayed by some pointless moves
3. X is not detected by the evaluation function
In this case depth-limited minimax can be fooled
It will use the pointless moves to push X beyond the depth limit, “horizon”, in which case it will “not see X”, and ignore it.
This can lead the search to take bad moves because it ignores the inevitability of X

Horizon effect

There may be disaster or success just beyond the search horizon
A state seen by the evaluation function may be evaluated as better or worse than it really is because a desirable or undesirable potential play just beyond the horizon is not considered
Instead of making a move that eliminates a threat, limited-depth search may choose a move that simply postpones a threat by pushing it past the horizon

Game Playing – Beyond alphabeta

We looked briefly at two problems
“non-quiescence”, and the “horizon effect”
and one solution “quiescence search”

To seriously implement a game
Deep-blue, chinook, etc
it is necessary to solve many such problems!
Good programs implement many techniques and get them to work together effectively
Game Playing – Game Classification

- So far have only considered games such as chess, checkers, and nim.
- These games are:
  1. Fully observable
     - Both players have full and perfect information about the current state of the game
  2. Deterministic
     - There is no element of chance
     - The outcome of making a sequence of moves is entirely determined by the sequence itself

Things to Consider

- Calculating optimal decisions are intractable in most cases, thus all algorithms must make some assumptions and approximations.
- The standard approach based on minimax, evaluation functions, and alpha-beta pruning is just one way of doing things.
- These search techniques do not reflect how humans actually play games.

Demonstrating A Problem

- Given this two-ply tree, the minimax algorithm will select the right-most branch, since it forces a minimum value of no less than 100.
- This relies on the assumption that 100, 101, and 102 are in fact actually better than 99.

Summary

- We defined the game in terms of a search.
- Discussion of two-player games given perfect information (minimax).
- Using cut-off to meet time constraints.
- Optimizations using alpha-beta pruning to arrive at the same conclusion as minimax would have.
- Complexity of adding chance to the decision tree.

Summary

- Few people would risk a sure gain of $1,000,000 for an even chance of gaining $10,000,000, for example. In fact, many decisions people make, such as buying insurance policies, playing lottery games, and gambling in a casino, indicate that they are not maximizing their average profits. Game theory does not attempt to indicate what a player's goal should be; instead, it shows the player how to attain his goal, whatever it may be.
- As a conclusion Game theory is the study of competitive interaction; it analyzes possible outcomes in situations where people are trying to score points off each other, whether in bridge, politics of war. You do this by trying to anticipate the reaction of your competitor to your next move and then factoring that reaction into your actual decision. It teaches people to think several moves ahead. Whoever it was who said it doesn’t matter if you win or lose but how you play the game, missed the point. It matters very much. According to game theory, it’s how you play the game that usually determines whether you win or lose.

Why Game theory

Game theory is both easy and excruciatingly difficult. People use it all the time, average people, in their daily lives. It comes into play in mundane deals like buying a car, where a certain skill in haggling is required. The buyer’s offer is usually formulated on the basis of what he or she presumes the seller will take. The seller is guided by a presumption about how high the buyer will go.
- It is used to describe any relationship and interaction, economic, social or political. And it’s useful in creating strategies for negotiators. It can help you win, and that is why companies and governments hire game theorists to write strategies against other players in whatever game they’re in. Mathematics and statistics are the tools they use. For example, during the Cold War the Pentagon became interested in game theory to help develop its nuclear strategy, and with some success.