Games vs. search problems

• "Unpredictable" opponent → specifying a move for every possible opponent’s reply.

• Time limits → unlikely to find goal, one must approximate
Game tree (2-player, deterministic, turns)

How do we search this tree to find the optimal move?
Idea: choose a move to a position with the highest minimax value = best achievable payoff against a rational opponent.

Example: deterministic 2-ply game:

Minimax value is computed bottom up:
- Leaf values are given.
- 3 is the best outcome for MIN in this branch.
- 3 is the best outcome for MAX in this game.
- We explore this tree in depth-first manner.
Properties of minimax

- **Complete?** Yes (if tree is finite)
- **Optimal?** Yes (against an rational opponent)
- **Time complexity?** $O(b^m)$
- **Space complexity?** $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$\rightarrow$ exact solution completely infeasible
1. Do we need to expand all nodes?

2. No: We can do better by pruning branches that will not lead to success.
α-β pruning example

MAX knows that it can at least get “3” by playing this branch.

MIN will choose “3”, because it minimizes the utility (which is good for MIN).
**α-β pruning example**

MAX knows that the new branch will never be better than 2 for him. He can *ignore* it.

MIN can certainly do as good as 2, but maybe better (= smaller)
α-β pruning example

MIN will do at least as good as 14 in this branch (which is very good for MAX!) so MAX will want to explore this branch more.
α-β pruning example

MIN will do at least as good as 5 in this branch (which is still good for MAX) so MAX will want to explore this branch more.
α-β pruning example

Bummer (for MAX): MIN will be able to play this last branch and get 2. This is worse than 3, so MAX will play 3.
Properties of $\alpha$-$\beta$

- Pruning does not affect final result (it is exact).
- Good move ordering improves effectiveness of pruning (see last branch in example).
- With "perfect ordering," time complexity = $O(b^{m/2})$ → doubles depth of search.
The Algorithm

- Visit the nodes in a depth-first manner
- Maintain bounds on nodes.
- A bound may change if one of its children obtains a unique value.
- A bound becomes a unique value when all its children have been checked or pruned.
- When a bound changes into a tighter bound or a unique value, it may become inconsistent with its parent.
- When an inconsistency occurs, prune the sub-tree by cutting the edge between the inconsistent bounds/values.

→ This is like propagating changes bottom-up in the tree.
Try it yourself

-which nodes can be pruned?
-always try going right before going left.
-maintain bounds!
Practical Implementation

How do we make this practical?

Standard approach:

• **cutoff test:** (where do we stop descending the tree)
  - depth limit
  - better: iterative deepening
  - cutoff only when no big changes are expected to occur next (quiescence search).

• **evaluation function**
  - When the search is cut off, we evaluate the current state by estimating its utility. This estimate is captured by the evaluation function.
  - Run $\alpha$-$\beta$ pruning minimax with these estimated values at the leaves instead.
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
  \[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.} \]
Deterministic games in practice

• Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.


• Othello: human champions refuse to compete against computers: they are too good.

• Go: human champions refuse to compete against computers: they are too bad.

• Poker: Machine was better than best human poker players in 2008.
Chance Games.

Backgammon

your element of chance
Expected Minimax

\[ v = \sum_{\text{chance nodes}} P(n) \times \text{Minimax}(n) \]

\[ 3 = 0.5 \times 4 + 0.5 \times 2 \]

Again, the tree is constructed bottom-up.

Now we have even more nodes to search!
4. (6pts) **Games** Consider the following 3-ply game: First MAX moves and has 2 choices (L or R), then MIN moves and has also 2 choices (L or R), finally MAX moves again and has two choices (L or R). So the total number of possible games add up to 8. The pay-offs for each possible game for MAX are as follows: RRR=6, RRL=5, RLR=8, RLL=7, LRR=2, LRL=1, LLL=4, LLL=3. For example: if MAX moves R, MIN moves R and MAX moves R again will pay 6 to MAX. MAX tries to maximize it’s pay-off and MIN tries to minimize MAX’s pay-off.

a. (2pts) Draw the 3-ply game tree for this game including the usual leaf-nodes which contain the pay-off values for MAX. Assign to every node in the tree the best pay-off for MAX in that branch. For instance, the root node should contain the value that is paid to MAX after actually playing the game rationally.

b. (3pts) Search the game tree using the MINIMAX algorithm and alpha-beta pruning. R moves should be explored before L moves, when the MINIMAX algorithm has no preference. Which nodes need NOT be visited if we apply alpha-beta pruning? Each time you decide a branch can be pruned, redraw the part of the game tree that has been visited already and explain why the branch can be pruned. It is recommended that you maintain bounds on the possible values of each node while searching the tree, and explain each pruning step using those bounds (similarly to what was done in class).

c. (1pt) What are the worst case time and space complexity of the MINIMAX search algorithm?
4. (6pts) **Games** Consider the following 3-ply game: First MAX moves and has 2 choices (L or R), then MIN moves and has also 2 choices (L or R), finally MAX moves again and has two choices (L or R). So the total number of possible games add up to 8. The pay-offs for each possible game for MAX are as follows: RRR=6, RRL=5, RLR=8, RLL=7, LRR=2, LRL=1, LLR=4, LLL=3. For example: if MAX moves R, MIN moves R and MAX moves R again will pay 6 to MAX. MAX tries to maximize its pay-off and MIN tries to minimize MAX’s pay-off.

a. (2pts) Draw the 3-ply game tree for this game including the usual leaf-nodes which contain the pay-off values for MAX. Assign to every node in the tree the best pay-off for MAX in that branch. For instance, the root node should contain the value that is paid to MAX after actually playing the game rationally.

a) answer: Root=6,L=2,R=6,LL=4,LR=2,RR=6,RL=8.

b. (3pts) Search the game tree using the MINIMAX algorithm and alpha-beta pruning. R moves should be explored before L moves, when the MINIMAX algorithm has no preference.
Which nodes need NOT be visited if we apply alpha-beta pruning? Each time you decide a branch can be pruned, redraw the part of the game tree that has been visited already and explain why the branch can be pruned. It is recommended that you maintain bounds on the possible values of each node while searching the tree, and explain each pruning step using those bounds (similarly to what was done in class).

c. (1pt) What are the worst case time and space complexity of the MINIMAX search algorithm?
4.(6pts) **Games** Consider the following 3-ply game: First MAX moves and has 2 choices (L or R), then MIN moves and has also 2 choices (L or R), finally MAX moves again and has two choices (L or R). So the total number of possible games add up to 8. The pay-offs for each possible game for MAX are as follows: RRR=6, RRL=5, RLR=8, RLL=7, LRR=2, LRL=1, LLR=4, LLL=3. For example: if MAX moves R, MIN moves R and MAX moves R again will pay 6 to MAX. MAX tries to maximize it’s pay-off and MIN tries to minimize MAX’s pay-off.

   a.(2pts) Draw the 3-ply game tree for this game including the usual leaf-nodes which contain the pay-off values for MAX. Assign to every node in the tree the best pay-off for MAX in that branch. For instance, the root node should contain the value that is paid to MAX after actually playing the game rationally.

   a) answer: Root=6, L=2, R=6, LL=4, LR=2, RR=6, RL=8.

   b.(3pts) Search the game tree using the MINIMAX algorithm and alpha-beta pruning. R moves should be explored before L moves, when the MINIMAX algorithm has no preference. Which nodes need NOT be visited if we apply alpha-beta pruning? Each time you decide a branch can be pruned, redraw the part of the game tree that has been visited already and explain why the branch can be pruned. It is recommended that you maintain bounds on the possible values of each node while searching the tree, and explain each pruning step using those bounds (similarly to what was done in class).

   b) answer: Branches: RLL, LLL and LLR can be pruned.

   c.(1pt) What are the worst case time and space complexity of the MINIMAX search algorithm?
4.(6pts) **Games** Consider the following 3-ply game: First MAX moves and has 2 choices (L or R), then MIN moves and has also 2 choices (L or R), finally MAX moves again and has two choices (L or R). So the total number of possible games add up to 8. The pay-offs for each possible game for MAX are as follows: RRR=6, RRL=5, RLR=8, RLL=7, LRR=2, LRL=1, LLR=4, LLL=3. For example: if MAX moves R, MIN moves R and MAX moves R again will pay 6 to MAX. MAX tries to maximize it’s pay-off and MIN tries to minimize MAX’s pay-off.

a.(2pts) Draw the 3-ply game tree for this game including the usual leaf-nodes which contain the pay-off values for MAX. Assign to every node in the tree the best pay-off for MAX in that branch. For instance, the root node should contain the value that is paid to MAX after actually playing the game rationally.

a) answer: Root=6,L=2,R=6,LL=4,LR=2,RR=6,RL=8.

b.(3pts) Search the game tree using the MINIMAX algorithm and alpha-beta pruning. R moves should be explored before L moves, when the MINIMAX algorithm has no preference.
Which nodes need NOT be visited if we apply alpha-beta pruning? Each time you decide a branch can be pruned, redraw the part of the game tree that has been visited already and explain why the branch can be pruned. It is recommended that you maintain bounds on the possible values of each node while searching the tree, and explain each pruning step using those bounds (similarly to what was done in class).

b) answer: Branches: RLL, LLL and LLR can be pruned.

c.(1pt) What are the worst case time and space complexity of the MINIMAX search algorithm?

c.) answer: space: $O(bm)$ or $O(m)$, time $O(b^m)$ with $b$ the branching factor and $m$ the depth of the solution.
Summary

• Games are fun to work on!

• We search to find optimal strategy

• perfection is unattainable ➞ approximate

• Chance makes games even harder