Foundations of Artificial Intelligence

CS472/3 — Fall 1999
Lecture #9

Bart Selman

Slide CS472–1

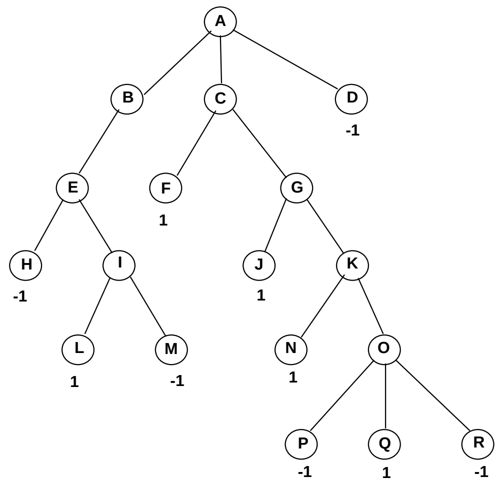
Today's Lecture

Game Playing, cont.

Readings: R&N, Chapter 5.

Slide CS472–2

Improving Minimax — $\alpha - \beta$ pruning

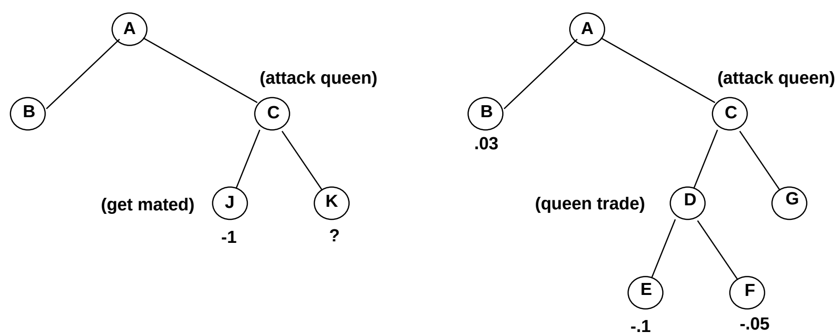


Slide CS472–3

1. Key: Once we realize that Max can win by moving to c, we don't have to analyze any other options from a. The values assigned to all of the nodes under b are guaranteed not to affect the overall value assigned to a.

Slide CS472–4

Two More Example

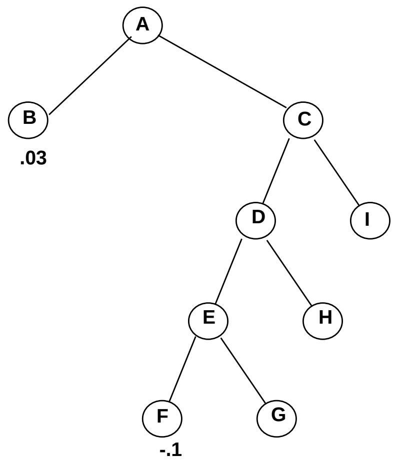


Slide CS472–5

1. First figure. We're considering two moves, b and c. If we make move c (i.e., *attack Min's queen*), then we lose (i.e., *get mated*). Does it matter what happens along any other branches from c? No. We've seen enough.
2. Second figure. Assume we're using DFS to analyze the tree (our Minimax algorithm implicitly uses DFS) and we've determined the value to be assigned to b. It's slightly favorable to us.
3. Examine c — *attacking Min's queen.* Check d and g. Apply static evaluator to e, we find -.1 and -.05. Choose -.05. (d is an exchange of queens.) Now, does is matter what happens at g and below? No.

Slide CS472–6

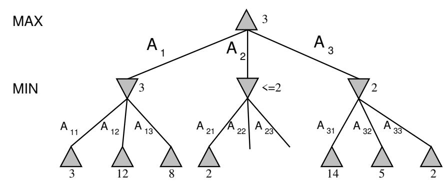
A deep $\alpha - \beta$ cutoff



Slide CS472–7

1. What can we say about g? Do we have to explore g's children? If g is part of a good solution for Max, then so is e. But from e, Min can choose f, guaranteeing a value of -.1. This is worse than Max's choice at the root, so it doesn't matter what happens at g.
2. What about h? Still have to explore h. Same for i.
3. Note: Reason for pruning is more than one ply above the pruned node.

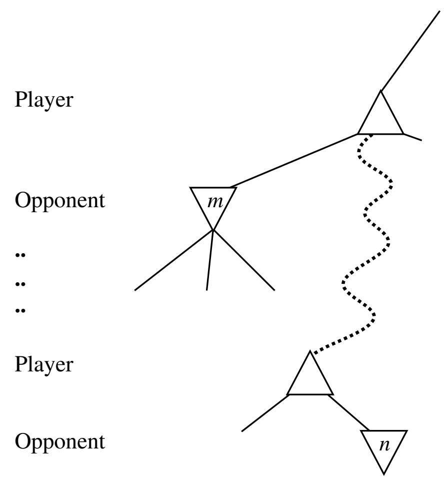
Slide CS472–8



Slide CS472–9

- Move A_2 is worth at most 2; we have already A_1 with utility 3, so we can prune the rest of the search at A_2.

Slide CS472–10



If m is better than n for Player, never get to n in play.

Slide CS472–11

- Add automatic pruning to minimax. Called alpha-beta pruning.
- Idea: Note minimax expands in depth-first manner. Let α be best choice we've found so far at any choice point along the path for MAX, and β best for MIN (i.e., lowest value). Alpha-beta search updates α and β during search, and prunes subtree as soon as it's worse than the current α or β.

Slide CS472–12

$\alpha - \beta$ Search

c = search cutoff
α = lower bound on Max's outcome; initially set to $-\infty$
β = upper bound on Min's outcome ; initially set to $+\infty$

We'll call $\alpha - \beta$ procedure recursively with a narrowing range between α and β.

Maximizing levels may reset α to a higher value; Minimizing levels may reset β to a lower value.

Slide CS472–13

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           α, the best score for MAX along the path to state
           β, the best score for MIN along the path to state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
      α ← MAX(α, MIN-VALUE(s, game, α, β))
      if α ≥ β then return β
   end
   return α
---------------------------------------------------------------
function MIN-VALUE(state, game, α, β) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
      β ← MIN(β, MAX-VALUE(s, game, α, β))
      if β ≤ α then return α
   end
   return β
```

Slide CS472–14

Search Space Size Reductions

1. Reduction in search space depends on ordering nodes!
2. Optimistic analysis: banching only $\sqrt{b}$ — for chess, 6 instead of 35.
3. Can go twice as deep! E.g., eight ply instead of four. Tremendous difference (also in practice).

Including chance

- Section 5.5, R&N.
- E.g. Backgammon.
- **expectiminimax.**

- State of the Art in Adversary Search
- Role in AI

Slide CS472–17

State of the Art in Checkers

- 1952: Samuel developed a checkers program that learned its own evaluation function through self play.
- 1992: *Chinook* (J. Schaeffer) wins the U.S. Open. At the world championship, Marion Tinsley beat *Chinook*

Slide CS472–18

State of the Art in Backgammon

- 1980: *BKG* using one-ply search and lots of luck defeated the human world champion.
- 1992: Tesauro combines Samuel's learning method with neural networks to develop a new evaluation function, resulting in a program ranked among the top 3 players in the world.

State of the Art in Go

$2,000,000 prize available for first computer program to defeat a top level player.

History of Chess in AI

500	legal chess
1200	occasional player
2000	world-ranked
2900	Gary Kasparov

Early 1950's Shannon and Turing both had programs that (barely) played legal chess (500 rank).

1950's Alex Bernstein's system, (500+ϵ).

1957 Herb Simon claims that a computer chess program would be world chess champion in 10 years...yeah, right.

Slide CS472–21

1966 McCarthy arranges chess match, Stanford vs. Russia. Long, drawn-out match. Russia wins.

1967 Richard Greenblatt, MIT. First of the modern chess programs, *MacHack* (1100 ranking).

1968 McCarthy, Michie, Papert bet Levy (rated 2325) that a computer program would beat him within 10 years.

1970 ACM started running chess tournaments. Chess 3.0-6 (rated 1400).

1973 By 1973...Slate: "It had become too painful even to look at Chess 3.6 any more, let alone work on it."

1973 Chess 4.0: smart plausible-move generator rather than

Slide CS472–22

speeding up the search. Improved rapidly when put on faster machines.

1976 Chess 4.5: ranking of 2070.

1977 Chess 4.5 vs. Levy. Levy wins.

1980's Programs depend on search speed rather than knowledge (2300 range).

1993 DEEP THOUGHT: Sophisticated special-purpose computer; $\alpha - \beta$ search; searches 10 ply; singular extensions; rated about 2600.

1995 DEEP BLUE: searches 14-ply; considers 100–200 billion positions per move.