CS-F407: Artificial Intelligence

12 Heuristic Function

Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Sept 06, 2024 M/W/F - 3:00 pm 6101 @ BITS-Pilani [Aug-Dec 2024]

http://ktiwari.in/ai

Hill Climbing Search

- Steepest ascent version
- Look around and choose the best
- Greedy-local search
- Makes quite rapid progress

Algorithm 1: Hill-Climbing(problem)

```
1 Returned a state of problem that is local maximum.
2 current ← Make-Node (problem.Initial-State)
3 while not terminate do
4     neighbor ← a highest valued successor of current
5     if neighbor.Value ≤ current.Value then
6         return current.State
7     current ← neighbor
```


Hill Climbing Variants

Stochastic hill climbing

Instead of the best one, choose a path with probability of its steepness (converge slowly).

First-choice hill climbing

Do not explore the state-space when you see the first improvement.

Random-restart hill climbing

if probability of success is p then run $1/p$ number of times.

- For our previous case [a] it needs 7 iterations.
- and 22 steps

[a] With random start 8-queens state, hill climbing gets stuck 86% of time.

Simulated Annealing (Metallurgy approach)

- Simulated annealing is hill climbing combined with **random walk**
- Step size is gradually reduced
- First Applied around 1980, for VLSI layout problem

Algorithm 2: Simulated-Annealing(problem, schedule)

```
1 current ← Make-Node (problem.Initial-State)
2 for t = 1 to ∞ do
3     T ← schedule(t)
4     if T = 0 then return current
5     next ← a randomly selected successor of current
6     ΔE ← next.Value - current.Value
7     if ΔE > 0 then current ← next
8     else current ← next with probability e^{ΔE/T}
```


Local Beam Search

- Modification of Best First Search to reduce memory requirements
- **Greedy:** only a predetermined number of promising node are kept as candidates.
- Do not put all unexplored nodes in frontier - instead use Beam Width number of nodes only.
- Improves space - also time.
- With Beam Width = 1, it is hill climbing.

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Example: Induct three new airports in Romania

- Let at (x_1, y_1), (x_2, y_2) and (x_3, y_3) on the map
- Minimize sum of distances of all the cities from its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2)$$

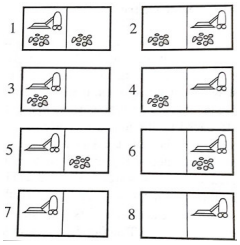
- If you discretize the neighborhood then there are only 12 next state (move only $\pm\delta$ in one step). One can apply hill climbing.
- If you attempt to use gradient $\bigtriangledown f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3})$ it cannot be solved as globally finding $\bigtriangledown f$ is not possible.
- Given locally correct values of $\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_i - x_c)$ one can perform steepest-ascent using $x \leftarrow x + \alpha \bigtriangledown f$

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[1]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt



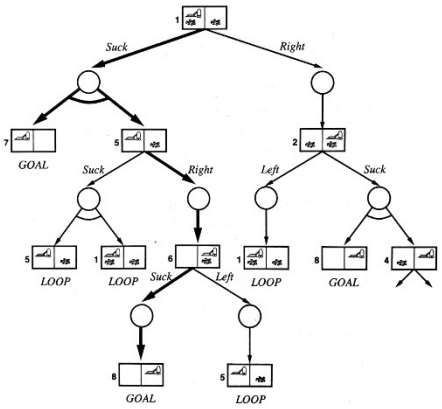
- Transition would lead use to more than one state
- *suck* in 1, would lead {5,7}
- Solution would have nested if-else

[*suck*, **if** state=5 **then** [*right*,*suck* **else** []]

- Search tree would contain some OR nodes and some AND nodes

[1] Percepts would tell where have we reached.

AND-OR Search Tree



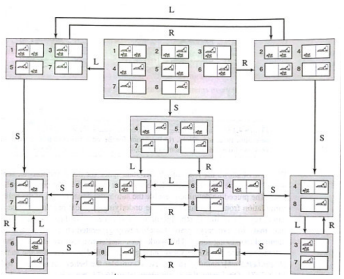
Solution

1. has goal node at every leaf
2. takes one action at each OR node
3. includes every outcome branch at each AND node

Searching with Partial Observations

When percepts do not suffice to pin down the exact state

- **Sensor less**. consider [*right*,*suck*,*left*,*suck*] guarantees to reach in state 8 that is a goal state (traverses through belief states)
- All possible belief states may not be reachable (only 12 out of 2^8)

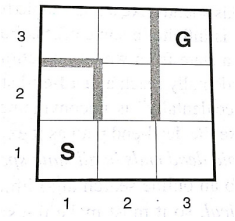


Online Search and Unknown Environment

Agent interleaves computation and action

Take action → observe environment → compute next action

- Online Search is necessary for unknown environment



1. Consider following maze problem
2. A robot need to go from S to G
3. Knows nothing about the environment

Random-walk?

No algorithm can avoid dead-end in all state space

Adversarial Search (game)

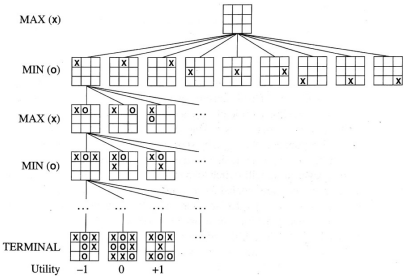
Agents having conflicting goals in competitive multiagent environment

- Deterministic, fully-observable, turn-taking, two-player, zero-sum
- Chess has roughly branching factor 35, moves 50 so tree search space is $35^{100} = 10^{154}$ however, graph has 10^{40} nodes
- Finding optimal move is infeasible but, needs an ability to decide

Game is between MAX and MIN (MAX moves first)

- S_0: the initial state
- PLAYER(s): defines which player has move to start
- ACTIONS(s): returns set of legal moves in a state
- RESULT(s, a):termination model defining result of a move
- TERMINAL_TEST(s): is true when game is over
- UTILITY(s, p): utility function defining reward (for chess +1,0,1/2)

Game Tree for tic-tac-toe

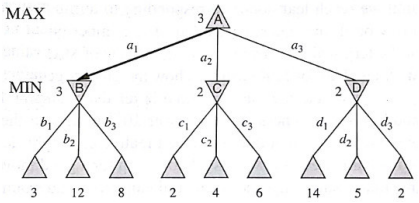


The search tree of the game has less than 9! = 362880 nodes.

MAX must find a contingent **strategy**.

Analogous to AND-OR search (MAX plays OR and MIN plays AND)

Two half moves is one **ply**



Given the game tree, optimal strategy can be determined from **minimax value** of each node.

$$MINIMAX(s) = \begin{cases} UTILITY(s) & \text{if } TERMINAL_TEST(s) \\ \max_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MAX \\ \min_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MIN \end{cases}$$

Action a_1 is the optimal choice [2]
(essentially optimizing worst-case outcome for MAX)

[2] utility value for MAX of being in corresponding state (assuming then onwards both player play optimally)

MINIMAX Algorithm

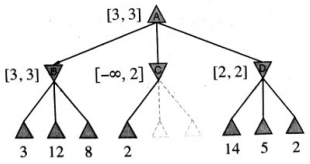
Returns the action corresponding to best move

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Recursion proceeds all the way down to the leaves. Time complexity $O(b^m)$ that is impractical but provides a basis of solution.

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

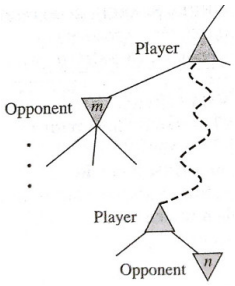


Consider two unevaluated successor of node C have value x and y

MINIMAX(root)
= max(min(3,12,8), min(2,x,y), min(14,5,2))
= max(3, min(2,x,y), 2)
= max(3, z, 2) where z=min(2,x,y)$\leq$ 2
= 3

ALPHA-BETA Pruning

- Alpha-beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtree rather than just leaves.



If m is better than n for player then we would never go to n in play

α	=	value of best choice (highest) found so far for MAX
β	=	value of best choice (lowest) found so far for MIN

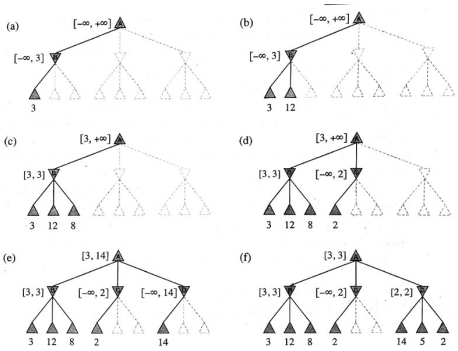
ALPHA-BETA Pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Order matters. So, examine likely to be best successor first.

Is it possible?
No

In-action: ALPHA-BETA Pruning



Move Ordering

- With perfect ordering we need to examine only $O(b^{m/2})$ nodes
- When successors are examined in random order it needs to examine $O(b^{3m/4})$ nodes
- In chess, a strategy that capture → threat → forward → backward, gets you to within about a factor of 2 of the best case $O(b^{m/2})$
- Try first the move that were found useful in past (killer move)
- Iterative deepening could help (adds constant fraction time)
- Hash table of previously seen positions (**transposition table**) can restrict re-computation of states

Imperfect Real-time Decision

- Generating entire search space is overkill
- Covering till large depth is not possible
- Idea is to cutoff the search earlier

$$H\text{-}MINIMAX(s,d) = \begin{cases} Eval(s) & \text{if } CUTOFF_TEST(s,d) \\ \text{argmax}_{a \in Actions(s)} H\text{-}MINIMAX(RESULT(s,a), d+1) & \text{if } PLAYER(s) = MAX \\ \text{argmin}_{a \in Actions(s)} H\text{-}MINIMAX(RESULT(s,a), d+1) & \text{if } PLAYER(s) = MIN \end{cases}$$

Evaluation Function

- Returns the estimate of expected utility
- Performance would deeply depend on it
- Eval(win) $\geq$ Eval(draw) $\geq$ Eval(lose)
- Computation should be quick
- Value should relate to actual chance of winning
- Can use features (Number of pawns, Number of queens, ...)
- Can have various categories (all pawn vs one pawn, etc) Suppose experience suggests 72% of states encountered in the two-pawn vs. one pawn category lead to win, 20% lose, and 8% in draw. then eval could be

$$0.72 \times 1 + 0.20 \times 0 + 0.08 \times 0.5 = 0.76$$

- **Weighted linear function** are also possible

$$Eval(s) = w_1 \times f_1(s) + w_2 \times f_2(s) + w_3 \times f_3(s) + ...$$

Important Considerations

- **Cutting off search**: having fixed depth may not be a good idea. **Quiescent** states are unlikely to have wild swing so expand till it. **Horizon effect** is difficult to eliminate so **singular extension**[3] could be used
- **Forward Pruning**: cutting off the some search without further consideration. (we do not evaluate all possibilities **beam search**, we only use some in our mind) This may be fatal.
- **Search Vs Lookup** We have some policy on how to start and finish well. Use them as lookup

[3] can I take a previous good move

Thank You!

Thank you very much for your attention!

Queries ?

(Reference[4])

[4] 1) Book - *AIMA*, ch-03+04, Russell and Norvig. 2) Book - Machine Learning, ch-09, *Tom Mitchell*