From Search to Games
Heads-up limit Texas hold 'em poker solved by University of Alberta scientists

Poker algorithm another step toward artificial intelligence

By Aleksandra Sagan, CBC News

Scientists at the University of Alberta have essentially solved heads-up limit hold 'em poker with an algorithm they hope will lead to advances in artificial intelligence. (Shutterstock)
Recap of last class

• problem formulation
  • Importance of good state description and successor function to solution efficiency

• search methods
  • explore state space for solution to problem
  • can be uninformed (blind) or use some reasonable knowledge (heuristics) to guide search
Recap of last class

- Office hours: Thursday 10-11:30 in MC 424
- Course website: www.cim.mcgill.ca/~jer/courses/ai
Today’s Agenda

• informed and informed search
• game-theory
• minimax algorithm
• alpha-beta pruning
Uninformed Search

- **breadth-first**
  - expand shallowest nodes first (FIFO)
- **depth-first**
  - expand deepest nodes first (LIFO)
- **depth-limited search**
  - depth-first with cutoff
Breadth-first search

- Expand *shallowest* unexpanded node
- Put successors at end of FIFO queue
Exercise:
cost to locate ‘K’ and ‘U’ using BF search
**Breadth-first search**

<table>
<thead>
<tr>
<th><strong>Complete?</strong></th>
<th>Yes (if b is finite)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time complexity</strong></td>
<td>$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$</td>
</tr>
<tr>
<td><strong>Space complexity</strong></td>
<td>$O(b^d)$ (every node kept in memory)</td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td>Yes (if cost = 1 per step)</td>
</tr>
</tbody>
</table>

Exponential time/memory requirements make breadth-first search unsuitable for large problems.
Depth-first search

- Expand **deepest** unexpanded node
- Put successors at front of LIFO queue
Exercise:
cost to locate ‘K’ and ‘U’ using DF search
# Depth-first search

<table>
<thead>
<tr>
<th>Complete?</th>
<th>No (fails in infinite-depth spaces or spaces with loops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time complexity</td>
<td>$O(b^m)$ (bad if $m &gt;&gt; d$)</td>
</tr>
<tr>
<td>Space complexity</td>
<td>$O(bm)$ (linear in space)</td>
</tr>
<tr>
<td>Optimal?</td>
<td>No</td>
</tr>
</tbody>
</table>
How to get best of both worlds?

• i.e., how to combine completeness of breadth first & space complexity of depth-first search?

• start with depth-limited search to solve the infinite depth problem
Depth-limited search

• depth-first search with depth limit $l$

Complete? \hspace{1cm} only if $l > d$

Time complexity \hspace{1cm} $O(b^l)$

Space complexity \hspace{1cm} $O(bl)$

Optimal? \hspace{1cm} only if $l = d$
Iterative-deepening search

- use depth-limited search as subroutine with increasing \( \ell \)
- is this efficient?

Complete? Yes

Time complexity \( d + (d-1)b + (d-2)b^2 + \ldots + b^d = O(b^d) \)

Space complexity \( O(bd) \)

Optimal? Yes (if cost = 1 per step)
8-squares problem
What’s a good state description and successor function?

initial state

goal state
Informed Search: Greedy Search

- minimize estimated cost to goal, $h(n)$
- start by expanding minimal cost node
**Informed Search: A* search**

- **minimize estimated cost to goal:** \( f(n) = g(n) + h(n) \)
  - \( g(n) \) = cost of solution from start to \( n \) and \( h(n) \) is estimated cost of cheapest solution from \( n \) to goal
  - \( h(n) \) is **admissible** if it never over-estimates true cost to reach goal
- **A* uses a best-first search:** chooses least-cost path from initial state to goal state
Optimality of A* search

- If $h(n)$ is admissible (valid), A* is optimal:
  - no optimal algorithm employing the same heuristic will expand fewer nodes than A*
- Exercise: why?
Exercise

• for the 8-squares problem, which of the following are valid (admissible) heuristics?
  h(n) = number of displaced tiles or
  h(n) = sum of Manhattan distances of displaced tiles

• which heuristic is better?
Exercise

- for the 8-squares problem, which of the following are valid (admissible) heuristics?
  - \( h(n) = \text{number of displaced tiles} \)
  - \( h(n) = \text{sum of Manhattan distances of the displaced tiles} \)

  **ANSWER: BOTH**
  - \( h(n) \) is valid if it never overestimates actual cost to reach goal

- which heuristic is better?

  **ANSWER: Second one**
  - It provides a higher lower bound on the estimate
Homework: the 8-squares problem

Generate the first two levels of the state space for this problem by drawing a labelled state tree, using the Manhattan distance heuristic to assign an A* value to each node. What are the first three moves you would make?
Homework

Generate the first two levels of the state space for this problem by drawing a labelled state tree, using the Manhattan distance heuristic to assign an A* value to each node. What are the first three moves you would make?
Let’s work it through...

- That’s no good!
- What to do?

h(n) = 20
h(n) = 19
h(n) = 18
h(n) = 19
From problem formulation to game defn

- **states**: description of “world of interest”
- **initial state**
- **successor function**: generates set of legal next states from available actions
- **goal test** → **terminal test**: when is game over?
- **path cost** → **utility function**: value of terminal states
Why search won’t work

• search for sequence of moves that leads to terminal state with positive utility (winning state)
• opponent might not be so cooperative!
Optimal Decisions in 2-player games

- solution: find strategy that leads to winning state regardless of what opponent does
Minimax strategy for 2-player games

• generate whole game tree down to terminal nodes
• find value of each terminal state using utility function
• repeat
  • determine utility of parent nodes from children
    • MIN chooses move that minimizes utility
    • MAX chooses move that maximizes utility
• until we reach root
• choose move that leads to “best” value
MiniMax algorithm

minmax(u) {    // u is node to evaluate
    if u is a leaf return score of u;
    else if u is a min node
        for all children of u: v1, .. vn
            return min{ minmax(v1),..., minmax(vn)}
    else    // u is a max node
        for all children of u: v1, .. vn
            return max{ minmax(v1),...,minmax(vn)}
}
NIM - player to take last stick loses

Exercise: build tree for NIM 122…
Expanded tree

MAX
You: : 

MIN
Opponent: :

MAX
You: :

MIN
Opponent: :

MIN
Opponent: :

MAX
You: :

MIN
Opponent: :
Improving Search:
Alpha-Beta Pruning – Exercise #1

MAX

MIN
Improving Search:
Alpha-Beta Pruning – Exercise #2
Improving Search: Alpha-Beta Pruning – Exercise #3
Improving Search:
Alpha-Beta Pruning – Exercise #3
**Alpha-Beta pseudocode**

\( \alpha \): minimal score \( \text{MAX} \) is guaranteed  
\( \beta \): maximum score \( \text{MAX} \) can hope to obtain (against a sensible opponent)

```python
function MaxV(state, \( \alpha \), \( \beta \))
if CUTOFF-TEST(state) then
    return EVAL(state)
for each \( s \) in SUCC(state) do
    \( \alpha \leftarrow \text{MAX}(\alpha, \text{MinV}(s, \alpha, \beta)) \)
    if \( \alpha \geq \beta \) then return \( \beta \)
return \( \alpha \)

function MinV(state, \( \alpha \), \( \beta \))
if CUTOFF-TEST(state) then
    return EVAL(state)
for each \( s \) in SUCC(state) do
    \( \beta \leftarrow \text{MIN}(\beta, \text{MaxV}(s, \alpha, \beta)) \)
    if \( \alpha \geq \beta \) then return \( \alpha \)
return \( \beta \)
```
Problem

- usually impossible to explore entire state space (e.g., chess search tree has approx. $35^{100}$ nodes)
- infeasible to make optimal decision
- solution: use heuristic position evaluators – an estimate of utility of states based on insight
Position Evaluators

• Game specific: have to be creative
• What are the determining factors in the goodness of a game state (utility)?
  • e.g., chess:
    • Sum of point value of pieces
    • Control of centre of board
    • Pawn structure
    • Defence of king
    • Mobility of pieces
Recap

- game-theory – why normal search won’t work
- minimax algorithm – brute-force traversal of game tree for “best” move
- alpha-beta pruning – how to improve on minimax for a more efficient traversal
- position evaluator functions – how to determine utility of a non-terminal node
Next Class

• Intelligence and agency:
  • intro to AI from an agent perspective
  • AI agent architectures