Examples of heuristics in A*-search

Games and Adversarial Search
Recap: Search algorithms

- State-space diagram vs Search Tree

- Uninformed Search algorithms
  - BFS / DFS
  - Depth Limited Search
  - Iterative Deepening Search.
  - Uniform cost search.

- Informed Search (with an heuristic function $h$):
  - Greedy Best-First-Search. (not complete / optimal)
  - A* Search (complete / optimal if $h$ is admissible)
Recap: Summary table of uninformed search

<table>
<thead>
<tr>
<th>Criteria</th>
<th>BFS</th>
<th>Uniform-cost</th>
<th>DFS</th>
<th>Depth-limited</th>
<th>IDS</th>
<th>Bidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes#</td>
<td>Yes#$&amp;</td>
<td>No</td>
<td>No</td>
<td>Yes#</td>
<td>Yes#$+</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+[C/e]})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+[C/e]})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes$$</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes$$</td>
<td>Yes$$+</td>
</tr>
</tbody>
</table>

$b$: Branching factor
$d$: Depth of the shallowest goal
$l$: Depth limit
$m$: Maximum depth of search tree
$e$: The lower bound of the step cost

#: Complete if $b$ is finite
$\&$: Complete if step cost $\geq e$
$\$$: Optimal if all step costs are identical
$\$$+: If both direction use BFS

(Section 3.4.7 in the AIMA book.)
Recap: A* Search (Pronounced “A-Star”)

• Uniform-cost search minimizes $g(n)$ ("past" cost)

• Greedy search minimizes $h(n)$ ("expected" or "future" cost)

• “A* Search” combines the two:
  – Minimize $f(n) = g(n) + h(n)$
  – Accounts for the “past” and the “future”
  – Estimates the cheapest solution (complete path) through node $n$

function A*-SEARCH(problem, $h$) returns a solution or failure
return BEST-FIRST-SEARCH(problem, $f$)
Recap: Avoiding Repeated States using A* Search

• Is GRAPH-SEARCH optimal with A*?

Graph Search
Step 1: Among B, C, E, Choose C
Step 2: Among B, E, D, Choose B
Step 3: Among D, E, Choose E. (you are not going to select C again)
Recap: Consistency (Monotonicity) of heuristic $h$

- A heuristic is consistent (or monotonic) provided
  - for any node $n$, for any successor $n'$ generated by action $a$ with cost $c(n,a,n')$
    - $h(n) \leq c(n,a,n') + h(n')$
      - akin to triangle inequality.
      - guarantees admissibility (proof?).
      - values of $f(n)$ along any path are non-decreasing (proof?).
        - Contours of constant $f$ in the state space
  - GRAPH-SEARCH using consistent $f(n)$ is optimal.
  - Note that $h(n) = 0$ is consistent and admissible.
This lecture

• Example of heuristics / A* search
  – Effective branching factor

• Games

• Adversarial Search
Heuristics

• What’s a heuristic for
  – Driving distance (or time) from city A to city B?
  – 8-puzzle problem?
  – M&C?
  – Robot navigation?

• **Admissible** heuristic
  – Does not overestimate the cost to reach the goal
  – “Optimistic”

• **Consistent** heuristic:
  – Satisfy a triangular inequality: \( h(n) \leq c(n,a,n’) + h(n’) \)

• Are the above heuristics admissible? Consistent?
Example: 8-Puzzle

Start State

Goal State
Comparing and combining heuristics

- Heuristics generated by considering relaxed versions of a problem.
- Heuristic $h_1$ for 8-puzzle
  - Number of out-of-order tiles
- Heuristic $h_2$ for 8-puzzle
  - Sum of Manhattan distances of each tile
- $h_2$ dominates $h_1$ provided $h_2(n) \geq h_1(n)$.
  - $h_2$ will likely prune more than $h_1$.
- $\max(h_1, h_2, \ldots, h_n)$ is
  - admissible if each $h_i$ is
  - consistent if each $h_i$ is
- Cost of sub-problems and pattern databases
  - Cost for 4 specific tiles
  - Can these be added for disjoint sets of tiles?
Effective Branching Factor

• Though informed search methods may have poor worst-case performance, they often do quite well if the heuristic is good
  – Even if there is a huge branching factor

• One way to quantify the effectiveness of the heuristic: the effective branching factor, $b^*$
  – N: total number of nodes expanded
  – d: solution depth
  – $N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

• For a good heuristic, $b^*$ is close to 1
### Example: 8-puzzle problem

Averaged over 100 trials each at different solution lengths

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2.87</td>
<td>1.48</td>
<td>1.45</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
<td>2.80</td>
<td>1.33</td>
<td>1.24</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
<td>2.79</td>
<td>1.38</td>
<td>1.22</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
<td>1.24</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
<td>2.83</td>
<td>1.44</td>
<td>1.23</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>

**Solution length**

Ave. # of nodes expanded
Memory Bounded Search

• Memory, not computation, is usually the limiting factor in search problems
  – Certainly true for A* search
• Why? What takes up memory in A* search?

• Solution: Memory-bounded A* search
  – Iterative Deepening A* (IDA*)
  – Simplified Memory-bounded A* (SMA*)
  – (Read the textbook for more details.)
Summary of informed search

• How to use a heuristic function to improve search
  – Greedy Best-first search + Uniform-cost search = A* Search

• When is A* search optimal?
  – h is Admissible (optimistic) for Tree Search
  – h is Consistent for Graph Search

• Choosing heuristic functions
  – A good heuristic function can reduce time/space cost of search by orders of magnitude.
  – Good heuristic function may take longer to evaluate.
Games and Adversarial Search

- Games: problem setup
- Minimax search
- Alpha-beta pruning
Illustrative example of a simple game (1 min discussion)

Example: game 1

You choose one of the three bins. I choose a number from that bin. Your goal is to maximize the chosen number.

A
-50  50

B
1   3

C
-5   15

(Example taken from Liang and Sadigh)
Game as a search problem

- $S_0$ The initial state
- PLAYER(s): Returns which player has the move
- ACTIONS(s): Returns the legal moves.
- RESULT(s, a): Output the state we transition to.
- TERMINAL-TEST(s): Returns True if the game is over.
- UTILITY(s,p): The payoff of player p at terminal state s.
Two-player, Turn-based, Perfect information, Deterministic, Zero-Sum Game

• Two-player: Tic-Tac-Toe, Chess, Go!

• Turn-based: The players take turns in round-robin fashion.

• Perfect information: The State is known to everyone

• Deterministic: Nothing is random

• Zero-sum: The total payoff for all players is a constant.

  • The 8-puzzle is a one-player, perfect info, deterministic, zero-sum game.
  • How about Rock-Paper-Scissors?
  • How about Monopoly?
  • How about Starcraft?
Tic-Tac-Toe

- The first player is X and the second is O
- Object of game: get three of your symbol in a horizontal, vertical or diagonal row on a 3x3 game board
- X always goes first
- Players alternate placing Xs and Os on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled (a draw)

What’s the state, action, transition, payoff for Tic-Tac-Toe?
Partial game tree for Tic-Tac-Toe

X’s turn

O’s turn

X’s turn

O’s turn

X’s turn

O’s turn

X’s wins
Game trees

• A game tree is like a search tree in many ways ...
  – nodes are search states, with full details about a position
    • characterize the arrangement of game pieces on the game board
  – edges between nodes correspond to moves
  – leaf nodes correspond to a set of goals
    • \{ \text{win, lose, draw} \}
    • usually determined by a score for or against player
      – at each node it is one or other player’s turn to move

• A game tree is not like a search tree because you have an opponent!
Two players: MIN and MAX

• In a zero-sum game:
  – payoff to Player 1 = - payoff to Player 2

• The goal of Player 1 is to maximizing her payoff.

• The goal of Player 2 is to maximizing her payoff as well
  – Equivalent to minimizing Player 1’s payoff.
Minimax search

- Assume that both players play perfectly
  - do not assume player will miss good moves or make mistakes
- Score(s): The score that MAX will get towards the end if both player play perfectly from s onwards.
- Consider MIN’s strategy
  - MIN’s best strategy:
    - choose the move that minimizes the score that will result when MAX chooses the maximizing move
  - MAX does the opposite
Minimaxing

- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

Each move is called a “ply”. One round is K-plies for a K-player game.
Minimax example

Which move to choose?

The \textbf{minimax decision} is move \( A_1 \)
Another example

• In the game, it’s your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?

```
  1  7  2
A

  2  5  2  8
B

  4  9  4  6
C

  3  3  5
D
```

MAX

MIN
Minimax search

- The \textit{minimax decision} maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)

- Generate the tree of minimax values
  - Then choose best (maximum) move
  - Don’t need to keep all values around
    - Good memory property

- Depth-first search is used to implement minimax
  - Expand all the way down to leaf nodes
  - Recursive implementation
Minimax properties

- Optimal?
  Yes, against an optimal opponent, if the tree is finite

- Complete?
  Yes, if the tree is finite

- Time complexity?
  Exponential: $O(b^m)$

- Space complexity?
  Polynomial: $O(bm)$
But this could take forever…

- Exact search is intractable
  - Tic-Tac-Toe is $9! = 362,880$
  - For chess, $b \approx 35$ and $m \approx 100$ for “reasonable” games
  - Go is $361! \approx 10^{750}$

- Idea 1: Pruning

- Idea 2: Cut off early and use a heuristic function
Pruning

- What’s really needed is “smarter,” more efficient search
  - Don’t expand “dead-end” nodes!
- **Pruning** – eliminating a branch of the search tree from consideration
- **Alpha-beta pruning**, applied to a minimax tree, returns the same “best” move, while pruning away unnecessary branches
  - Many fewer nodes might be expanded
  - Hence, smaller effective branching factor
  - …and deeper search
  - …and better performance
    - Remember, minimax is *depth-first* search
Alpha pruning

![Alpha pruning diagram](image-url)
Beta pruning
Improvements via alpha/beta pruning

- Depends on the ordering of expansion

- Perfect ordering \( O\left(b^{m/2}\right) \)

- Random ordering \( O\left(b^{3m/4}\right) \)

- For specific games like Chess, you can get to almost perfect ordering.
Heuristic (Evaluation function)

• It is usually impossible to solve games completely

• Rather, cut the search off early and apply a heuristic evaluation function to the leaves
  – \( h(s) \) estimates the expected utility of the game from a given position (node/state) \( s \)
  – like depth bounded depth first, lose completeness
  – Explore game tree using combination of evaluation function and search

• The performance of a game-playing program depends on the quality (and speed!) of its evaluation function
Heuristics (Evaluation function)

• Typical evaluation function for game: weighted linear function
  \[ h(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_df_d(s) \]
  \[ \text{weights} \cdot \text{features \ [dot product]} \]

• For example, in chess
  \[ W = \{ 1, 3, 3, 5, 8 \} \]
  \[ F = \{ \# \text{pawns advantage,} \# \text{bishops advantage,} \# \text{knight}\]
  \[ \text{advantage,} \# \text{rooks advantage,} \# \text{queens advantage} \} \]
  \[ \text{Is this what Deep Blue used?} \]
  \[ \text{What are some problems with this?} \]

• More complex evaluation functions may involve learning
  \[ \text{Adjusting weights based on outcomes} \]
  \[ \text{Perhaps non-linear functions} \]
  \[ \text{How to choose the features?} \]
Tic-Tac-Toe revisited

a partial game tree for Tic-Tac-Toe
Evaluation function for Tic-Tac-Toe

• A simple evaluation function for Tic-Tac-Toe
  – count the number of rows where $X$ can win
  – subtract the number of rows where $O$ can win
• Value of evaluation function at start of game is zero
  – on an empty game board there are 8 possible winning rows for both $X$ and $O$
Evaluating Tic-Tac-Toe

\[ \text{evalX} = (\text{number of rows where X can win}) - (\text{number of rows where O can win}) \]

- After X moves in center, score for X is +4
- After O moves, score for X is +2
- After X’s next move, score for X is +4

\[
\begin{align*}
8-8 &= 0 \\
8-4 &= 4 \\
6-4 &= 2 \\
6-2 &= 4
\end{align*}
\]
Evaluating Tic-Tac-Toe

evalO = (number of rows where O can win) - (number of rows where X can win)

- After X moves in center, score for O is -4
- After O moves, score for O is +2
- After X’s next move, score for O is -4

8-8 = 0  4-8 = -4  4-6 = -2  2-6 = -4
Search depth cutoff

Tic-Tac-Toe with search depth 2

Evaluations shown for X
Expectimax: Playing against a benign opponent

- Sometimes your opponents are not clever.
  - They behave randomly.
  - You can take advantage of that by modeling your opponent.

- Example of game of chance:
  - Slot machines
  - Tetris
Expectimax example

- Your opponent behave randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability $[0.5,0.5]$
- If you move right, your opponent will select actions with $[0.6,0.4]$

Note: pruning becomes tricky in expectimax… think about why.
Summary of game playing

• Minimax search

• Game tree

• Alpha-beta pruning

• Early stop with an evaluation function

• Expectimax
More reading / resources about game playing

● Required reading: AIMA 5.1-5.3

● Stochastic game / Expectiminimax: AIMA 5.5
  – Backgammon. TD-Gammon
  – Blackjack, Poker

● Famous game AI: Read AIMA Ch. 5.7 (or in the “Historical notes” of the AIMA 4th Edition)
  – Deep blue
  – TD Gammon

● AlphaGo: https://www.nature.com/articles/nature16961