Search

• One of the most basic techniques in AI
  • Underlying sub-module in most AI systems

• Can solve many problems that humans are not good at (achieving super-human performance)

• Very useful as a general algorithmic technique for solving many non-AI problems.
Heuristic Search*
(Informed Search)

*Courtesy of F. Bacchus
Heuristic Search

• In uninformed search, we don't try to evaluate which of the nodes on OPEN are most promising. We never “look-ahead” to the goal.
  
  E.g., in uniform cost search we always expand the cheapest path. We don't consider the cost of getting to the goal from the end of the current path.

• Often we have some other knowledge about the merit of nodes, e.g., going the wrong direction in Romania.
Heuristic Search

Merit of an OPEN node: different notions of merit.

• If we are concerned about the cost of the solution, we might want to consider how costly it is to get to the goal from the terminal state of that node.

• If we are concerned about minimizing computation in search we might want to consider how easy it is to find the goal from the terminal state of that node.

• We will focus on the “cost of solution” notion of merit.
Heuristic Search

• The idea is to develop a domain specific heuristic function $h(n)$.

• $h(n)$ guesses the cost of getting to the goal from node $n$ (i.e., from the terminal state of the path represented by $n$).

• There are different ways of guessing this cost in different domains.
  • heuristics are domain specific.
Planning a path from Arad to Bucharest, we can utilize the *straight line distance from each city to our goal*. This lets us plan our trip by picking cities at each time point that minimize the distance to our goal.
Heuristic Search

• If $h(n_1) < h(n_2)$ this means that we guess that it is cheaper to get to the goal from $n_1$ than from $n_2$.

• We require that
  • $h(n) = 0$ for every node $n$ whose terminal state satisfies the goal.
  • Zero cost of achieving the goal from node that already satisfies the goal.
Using only $h(n)$: Greedy best-first search

- We use $h(n)$ to rank the nodes on OPEN
  - Always expand node with lowest $h$-value.
  - We are greedily trying to achieve a low cost solution.

- However, this method **ignores the cost of getting to** $n$, so it can be lead astray exploring nodes that cost a lot but seem to be close to the goal:

  - step cost = 10
  - step cost = 100

  $h(n1) = 20$

  $h(n3) = 10$
Greedy best-first search example
A* search

• Take into account the cost of getting to the node as well as our estimate of the cost of getting to the goal from the node.

• Define an evaluation function \( f(n) \)
  \[ f(n) = g(n) + h(n) \]
  • \( g(n) \) is the cost of the path represented by node \( n \)
  • \( h(n) \) is the heuristic estimate of the cost of achieving the goal from \( n \).

• Always expand the node with lowest \( f \)-value on OPEN.

• The \( f \)-value, \( f(n) \) is an estimate of the cost of getting to the goal via the node (path) \( n \).
  • I.e., we first follow the path \( n \) then we try to get to the goal. \( f(n) \) estimates the total cost of such a solution.
A* example

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobroța</td>
<td>242</td>
</tr>
<tr>
<td>Eforțe</td>
<td>161</td>
</tr>
<tr>
<td>Făgăras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hîrsova</td>
<td>151</td>
</tr>
<tr>
<td>Iași</td>
<td>226</td>
</tr>
<tr>
<td>Lupșa</td>
<td>244</td>
</tr>
<tr>
<td>Mehedinți</td>
<td>241</td>
</tr>
<tr>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>360</td>
</tr>
<tr>
<td>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>Râmnicu Vâlcea</td>
<td>195</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Uzurii</td>
<td>189</td>
</tr>
<tr>
<td>Vâlcea</td>
<td>374</td>
</tr>
<tr>
<td>Zalău</td>
<td>374</td>
</tr>
</tbody>
</table>
A* example
A* example
A* example

Straight-line distance to Bucharest

- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobroția: 242
- Eforie: 161
- Făgăraș: 178
- Giurgiu: 77
- Hâșmaș: 151
- Iași: 226
- Lugoj: 244
- Mehadia: 241
- Neamț: 234
- Oradea: 389
- Pitești: 98
- Rimnița: 193
- Sălaj: 253
- Timișoara: 329
- Urziceni: 59
- Vaslui: 199
- Zerind: 374
A* example
A* example
Properties of A* depend on conditions on \( h(n) \)

• We want to analyze the behavior of the resultant search.
  • Completeness, time and space, optimality?

• To obtain such results we must put some further conditions on the heuristic function \( h(n) \) and the search space.
Conditions on $h(n)$: Admissible

- We always assume that $c(s_1, a, s_2) \geq \varepsilon > 0$ for any two states $s_1$ and $s_2$ and any action $a$: the cost of any transition is greater than zero and can't be arbitrarily small.

Let $h^*(n)$ be the **cost of an optimal path** from $n$ to a goal node ($\infty$ if there is no path). Then an **admissible** heuristic satisfies the condition

$$h(n) \leq h^*(n)$$

- an admissible heuristic **never over-estimates** the cost to reach the goal, i.e., it is **optimistic**

- Hence $h(g) = 0$, for any goal node $g$

- Also $h^*(n) = \infty$ if there is no path from $n$ to a goal node
Consistency (aka monotonicity)

• A stronger condition than $h(n) \leq h^*(n)$.

• A monotone/consistent heuristic satisfies the triangle inequality: for all nodes $n_1$, $n_2$ and for all actions $a$

\[ h(n_1) \leq C(n_1,a,n_2) + h(n_2) \]

Where $C(n_1, a, n_2)$ means the cost of getting from the terminal state of $n_1$ to the terminal state of $n_2$ via action $a$.

• Note that there might be more than one transition (action) between $n_1$ and $n_2$, the inequality must hold for all of them.

• Monotonicity implies admissibility.
  • $(\forall n_1, n_2, a) \ h(n_1) \leq C(n_1,a,n_2) + h(n_2) \Rightarrow (\forall n) \ h(n) \leq h^*(n)$
Consistency $\Rightarrow$ Admissible

• **Assume consistency:** $h(n) \leq c(n,a,n_2) + h(n_2)$

  **Prove admissible:** $h(n) \leq h^*(n)$

**Proof:**

If no path exists from $n$ to a goal then $h^*(n) = \infty$ and $h(n) \leq h^*(n)$

Else let $n \rightarrow n_1 \rightarrow \ldots \rightarrow n^*$ be an OPTIMAL path from $n$ to a goal (with actions $a_1$, $a_2$). Note the cost of this path is $h^*(n)$, and each subpath $(n_i \rightarrow \ldots \rightarrow n^*)$ has cost equal to $h^*(n_i)$.

Otherwise prove $h(n) \leq h^*(n)$ by induction on the length of this optimal path.

**Base Case:** $n = n^*$

  By our conditions on $h$, $h(n) = 0 \leq h(n)^* = 0$

**Induction Hypothesis:** $h(n_1) \leq h^*(n_1)$

  $h(n) \leq c(n,a_1,n_1) + h(n_1) \leq c(n,a_1,n_1) + h^*(n_1) = h^*(n)$
Intuition behind admissibility

h(n) ≤ h*(n) means that the search won’t miss any promising paths.

• If it really is cheap to get to a goal via n (i.e., both g(n) and h*(n) are low), then f(n) = g(n) + h(n) will also be low, and the search won’t ignore n in favor of more expensive options.

• This can be formalized to show that admissibility implies optimality.

• Monotonicity gives some additional properties when it comes to cycle checking.
Consequences of monotonicity

1. The f-values of nodes along a path must be non-decreasing.

Let \(<\text{Start} \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_k>\) be a path. Let \(n_i\) be the subpath \(<\text{Start} \rightarrow s_1 \rightarrow \ldots \rightarrow s_i>\):

We claim that: \(f(n_i) \leq f(n_i+1)\)

Proof

\[
f(n_i) = c(\text{Start} \rightarrow \ldots \rightarrow n_i) + h(n_i) \\
\leq c(\text{Start} \rightarrow \ldots \rightarrow n_i) + c(n_i \rightarrow n_i+1) + h(n_i+1) \\
\leq c(\text{Start} \rightarrow \ldots \rightarrow n_i \rightarrow n_i+1) + h(n_i+1) \\
\leq g(n_i+1) + h(n_i+1) = f(n_i+1)
\]
Consequences of monotonicity

2. If \( n_2 \) is expanded immediately after \( n_1 \), then
\[ f(n_1) \leq f(n_2) \]
(the f-value of expanded nodes is **monotonic** non-decreasing)

Proof:

- If \( n_2 \) was on OPEN when \( n_1 \) was expanded, then \( f(n_1) \leq f(n_2) \) otherwise we would have expanded \( n_2 \).
- If \( n_2 \) was added to OPEN after \( n_1 \)'s expansion, then \( n_2 \) extends \( n_1 \)'s path. That is, the path represented by \( n_1 \) is a prefix of the path represented by \( n_2 \). By property (1) we have \( f(n_1) \leq f(n_2) \) as the f-values along a path are non-decreasing.
Consequences of monotonicity

3. Corollary: the sequence of f-values of the nodes expanded by A* is non-decreasing. I.e, if n2 is expanded after (not necessarily immediately after) n1, then $f(n1) \leq f(n2)$

(the f-value of expanded nodes is monotonic non-decreasing)

Proof:

- If n2 was on OPEN when n1 was expanded, then $f(n1) \leq f(n2)$ otherwise we would have expanded n2.
- If n2 was added to OPEN after n1's expansion, then let n be an ancestor of n2 that was present when n1 was being expanded (this could be n1 itself). We have $f(n1) \leq f(n)$ since A* chose n1 while n was present on OPEN. Also, since n is along the path to n2, by property (1) we have $f(n) \leq f(n2)$. So, we have $f(n1) \leq f(n2)$. 

Consequences of monotonicity

4. When n is expanded every path with lower f-value has already been expanded.

• **Proof:** Assume by contradiction that there exists a path <Start, n0, n1, ni-1, ni, ni+1, ..., nk> with f(nk) < f(n) and ni is its last expanded node.
  
  • ni+1 must be on OPEN while n is expanded, so
    
    a) by (1) f(ni+1) ≤ f(nk) since they lie along the same path.
    b) since f(nk) < f(n) so we have f(ni+1) < f(n)
    c) by (2) f(n) ≤ f(ni+1) because n is expanded before ni+1.
  
  • Contradiction from b&c!
Consequences of monotonicity

5. With a monotone heuristic, the first time A* expands a state, it has found the minimum cost path to that state.

Proof:

- Let $\text{PATH1} = \langle \text{Start, } s_0, s_1, \ldots, s_k, s \rangle$ be the first path to a state $s$ found. We have $f(\text{path1}) = c(\text{PATH1}) + h(s)$.
- Let $\text{PATH2} = \langle \text{Start, } t_0, t_1, \ldots, t_j, s \rangle$ be another path to $s$ found later. We have $f(\text{path2}) = c(\text{PATH2}) + h(s)$.
- Note $h(s)$ is dependent only on the state $s$ (terminal state of the path) it does not depend on how we got to $s$.
- By property (3), $f(\text{path1}) \leq f(\text{path2})$
- Hence: $c(\text{PATH1}) \leq c(\text{PATH2})$
Consequences of monotonicity

Complete.

- Yes, consider a least cost path to a goal node
  - SolutionPath = <Start → n1 → ... → G> with cost \( c(\text{SolutionPath}) \). Since \( h(G) = 0 \), this means that \( f(\text{SolutionPath}) = c(\text{SolutionPath}) \)
  - Since each action has a cost \( \geq \varepsilon > 0 \), there are only a finite number of paths that have \( f \)-value < \( c(\text{SolutionPath}) \). None of these paths lead to a goal node since SolutionPath is a least cost path to the goal.
  - So eventually SolutionPath, or some equal cost path to a goal must be expanded.

Time and Space complexity.

- When \( h(n) = 0 \), for all \( n \) \( h \) is monotone.
  - \( A^* \) becomes uniform-cost search!
  - It can be shown that when \( h(n) > 0 \) for some \( n \) and still admissible, the number of nodes expanded can be no larger than uniform-cost.
  - Hence the same bounds as uniform-cost apply. (These are worst case bounds). Still exponential unless we have a very good \( h \! \)
  - In real world problems, we sometimes run out of time and memory. IDA* can sometimes be used to address memory issues, but IDA* isn’t very good when many cycles are present.
Consequences of monotonicity

Optimality

- Yes, by (5) the first path to a goal node must be optimal.

5. With a monotone heuristic, the first time A* expands a state, it has found the minimum cost path to that state.

Cycle Checking

- We can use a simple implementation of cycle checking (multiple path checking)---just reject all search nodes visiting a state already visited by a previously expanded node. By property (5) we need keep only the first path to a state, rejecting all subsequent paths.
Admissibility without monotonicity

When “h” is admissible but not monotonic.

- Time and Space complexity remain the same. Completeness holds.
- Optimality still holds (without cycle checking), but need a different argument: don't know that paths are explored in order of cost.
Admissibility without monotonicity

What about Cycle Checking?

• No longer guaranteed we have found an optimal path to a node *the first time* we visit it.

• So, cycle checking might not preserve optimality.

• To fix this: for previously visited nodes, must remember cost of previous path. If new path is cheaper must explore again.
Space Problems with A*

- A* has the same potential space problems as BFS or UCS
- IDA* - Iterative Deepening A* is similar to Iterative Deepening Search and similarly addresses space issues.
Constructing Heuristics
Building Heuristics: Relaxed Problem

• One useful technique is to consider an easier problem, and let $h(n)$ be the cost of reaching the goal in the easier problem.

• 8-Puzzle moves.
  • Can move a tile from square A to B if
    • A is adjacent (left, right, above, below) to B
    • and B is blank

• Can relax some of these conditions
  1. can move from A to B if A is adjacent to B (ignore whether or not position is blank)
  2. can move from A to B if B is blank (ignore adjacency)
  3. can move from A to B (ignore both conditions).
Building Heuristics: Relaxed Problem

- **#3** “can move from A to B (ignore both conditions)”.
  leads to the **misplaced tiles heuristic**.
  - To solve the puzzle, we need to move each tile into its final position.
  - Number of moves = number of misplaced tiles.
  - Clearly $h(n) = \text{number of misplaced tiles} \leq h^*(n)$ the cost of an optimal sequence of moves from $n$.

- **#1** “can move from A to B if A is adjacent to B (ignore whether or not position is blank)”
  leads to the **manhattan distance heuristic**.
  - To solve the puzzle we need to slide each tile into its final position.
  - We can move vertically or horizontally.
  - Number of moves = sum over all of the tiles of the number of vertical and horizontal slides we need to move that tile into place.
  - Again $h(n) = \text{sum of the manhattan distances} \leq h^*(n)$
    - in a real solution we need to move each tile at least that far and we can only move one tile at a time.
Building Heuristics: Relaxed Problem

The **optimal** cost to nodes in the relaxed problem is an **admissible heuristic** for the original problem!

**Proof Idea**: the optimal solution in the original problem is a solution for relaxed problem, therefore it must be at least as expensive as the optimal solution in the relaxed problem.

So admissible heuristics can sometimes be constructed by finding a relaxation whose optimal solution can be easily computed.
Building Heuristics: Pattern databases

• Try to generate admissible heuristics by solving a subproblem and storing the exact solution cost for that subproblem

• See Chapter 3.6.3 if you are interested.