
State space graph.
  Given a task, model of world appropriate to task.

Search Tree.
  Representation of all plans
  (also paths through state space graph.)

Uninformed Search.
  Depth First Search.
    Good space bound, “leftmost” exploration seems bad.
  Breadth First Search.
    Bad space bound, finds plan with fewest number of actions.

Uniform Cost Search.
  Bad space bounds, finds optimal plan.

Demo: L3D1
Informed Search
A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Euclidean distance for pathing. Manhattan distance.
Example: Heuristic Function

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>Dobroț</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Făgăraș</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>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamț</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>98</td>
</tr>
<tr>
<td>RimNICul Vîlcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place
Greedy Search
Greedy Search

Expand the node that seems closest to the goal?

What can go wrong?

Greedy: $140 + 99 + 211 = 450$
Better: $140 + 80 + 97 + 101 = 418$
Greedy Search

Strategy: expand a node that (you think) is closest to a goal state.
Heuristic: estimate of distance to nearest goal for each state

Wildest dream:
Perfect heuristic. A common case:
Best-first takes you straight to the (wrong) goal

Worst-case:
Like a badly-guided DFS

[Demo: contours greedy empty (L3D3)]
[Demo: contours greedy pacman small maze (L3D4)]
Video of Demo Contours Greedy (Empty)
Video of Demo Contours Greedy (pacman/small)
A* Search
A* Search

UCS

Greedy

A*
Combining UCS and Greedy

**Uniform-cost** orders by path cost, or backward cost: $g(n)$.

**Greedy** orders by goal proximity, or forward cost: $h(n)$.

A* Search orders by the sum: $f(n) = g(n) + h(n)$.

Blue $g(n)$, Red $h(n)$.
When should A* terminate?

Should we stop when we enqueue a goal?  

No: only stop when we dequeue a goal.

G in queue when B expanded.
Is A* Optimal?

What goes wrong?

Actual goal cost < estimated goal cost

We need estimates to be less than actual costs!
   Lower bounds for actual costs.
Admissible Heuristics
Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe.

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs.
Admissible Heuristics

A heuristic $h$ is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal.

Note: Coming up with admissible heuristics is most of what’s involved in using $A^*$ in practice.

Examples:

Optimize: number of flips.

Largest out of place pancake.

Admissible?

No. For number of flips.

Yes. For height of stack.

Manhattan distance.

Admissible? Yes!

Euclidean Distance.

Admissible? Yes.
Optimality of A* Tree Search
Optimality of A* Tree Search: Blocking

Claim: Goal $A$ is expanded before $B$ with $g(B) > g(A)$.

Proof:

Definition of f-cost: $g(n) + h(n)$.
Admissibility of $h(n) \leq h^*(n)$
$\implies h = 0$ at a goal

Imagine $B$ is on the fringe.

Some ancestor $n$ of $A$ is on the fringe, too (maybe $A$!)

Claim: $n$ will be expanded before $B$

- $f(n)$ is less or equal to $f(A)$
  - since $f(n) = g(n) + h(n) < f(A)$,
- $f(A)$ is less than $f(B)$
  - since $g(A) + 0 < g(B) + h(B)$

All ancestors of $A$ expand before $B$.

$A$ expands before $B. \rightarrow A^*$ search is optimal.
Properties of $A^*$
Uniform-cost expands equally in all “directions”.

A* expands mainly toward the goal, but does hedge its bets to ensure optimality.

[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacmansmall maze (L3D5)]
Video of Demo Contours (Empty) – UCS
Video of Demo Contours (Empty) – Greedy
Video of Demo Contours (Empty) – A*
Video of Demo Contours: Pacman A*
Comparison

Uniform Cost

Greedy

A*
A* Applications
A* Applications

Video games
Pathing / routing problems
Resource planning problems
Robot motion planning
Language analysis
Machine translation
Speech recognition
[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]
Video of Demo Pacman (Tiny Maze) – UCS / A*
Video: Demo Water Shallow/Deep – Guess Algorithm

Total cost: 27
Number of nodes expanded: 182
Number of unique nodes expanded: 182
Pacman emerges victorious! Score: 979
{'numKills': 0, 'results': ['Win'], 'numMoves': [27], 'score': [979]}
Creating Heuristics

YOU GOT
HEURISTIC UPGRADE!
Creating Admissible Heuristics

Most of the work in solving hard search problems optimally is in coming up with admissible heuristics.

Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available.

Inadmissible heuristics are often useful too.
Example: 8 Puzzle

Start State

Goal State

Actions

What are the states?
How many states?
What are the actions?
How many successors from the start state?
What should the costs be?
8 Puzzle I

Heuristic: Number of tiles misplaced
Why is it admissible? $h(\text{start}) =$
This is a relaxed-problem heuristic
Average nodes expanded when the optimal path has

- 4 steps
- 8 steps
- 12 steps.

UCS: 112, 6300, $3.6 \times 10^6$
TILES: 13, 39, 227
Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance

Why is it admissible?

\[ h(start) = 3 + 1 + 2 + \ldots = 18. \]

Average nodes expanded when the optimal path has:

<table>
<thead>
<tr>
<th>TILES</th>
<th>MANHATTAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 steps</td>
<td>8 steps</td>
</tr>
<tr>
<td>13</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
</tr>
</tbody>
</table>
How about using the actual cost as a heuristic?
Would it be admissible?
Would we save on nodes expanded?
What’s wrong with it?

With A*: a trade-off between quality of estimate and work per node

- As heuristics get closer to the true cost, expand fewer nodes, but more work per node to compute the heuristic itself.
Semi-Lattice of Heuristics
Trivial Heuristics, Dominance

**Dominance:** $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

Heuristics form a semi-lattice:
Max of admissible heuristics is admissible.

Trivial heuristics
Bottom of lattice is the zero heuristic.
(what does this give us?)
Top of lattice is the exact heuristic
Graph Search
Tree Search: Extra Work!

Failure to detect repeated states can cause exponentially more work.
In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)
Graph Search

Idea: never expand a state twice

How to implement:
  Tree search + set of expanded states ("closed set")
  Expand the search tree node-by-node, but...
  Before expanding a node,
    check if state was never been expanded before
    If yes skip it, else add to closed set and expand.

Important: store the closed set as a set, not a list

Can graph search wreck completeness? Why/why not?
How about optimality?
A* Graph Search Gone Wrong?

Is \( h(\cdot) \) admissible? Yes.

Will exploring w.r.t \( h(B) + g(n) \) be optimal?

Expand \( S \).

\( A \) and \( B \) in fringe!

Expands \( B \), since \( h(B) + g(B) = 2 < 5 = h(A) + g(A) \).

\( C \) in fringe with key, \( 3 + h(C) = 4 \).

\( G \) in fringe with key, \( 5 \).

Could have been there in 4.
Consistency of Heuristics

Main idea: est. heuristic costs \( \leq \) actual costs

**Admissibility:** \( h(x) \leq \text{cost to goal} \).

**Consistency:** \( h(x) - h(y) \leq \text{cost}(x, y) \).

heuristic “arc” cost \( \leq \) actual arc cost

Consistent \( \implies \) admissible? Yes? No?

Consistent: f value along a path never decreases

Admissible:

\[
\begin{align*}
  f(C) &= h(C) + 1 = 3. \\
  f(A) &= h(A) = 4. \\
  \text{Consistent: } f(A) &= 2 < 3 = f(C).
\end{align*}
\]

Claim: If \( y \) is expanded due to \( x \), \( f(y) \geq f(x) \).

Proof:

\[
\begin{align*}
  f(y) &= g(x) + \text{cost}(x, y) + h(y) \\
  &\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x) \quad \square
\end{align*}
\]

The “estimate” of plan cost keeps rising as you progress.
Optimality of A* Graph Search
Optimality of A* Graph Search

Sketch: consider what A* does with a consistent heuristic:
- Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
- Fact 2: For every state s, the optimal path is discovered.

Result: A* graph search is optimal

Fact 1 Proof. Previous slide.
Proof of A* optimality.

Fact 2: The optimal path is discovered to every state $s$.

Proof: Consider first error. State $s$ discovered from $x$. Optimal path is from $y \neq x$.

There is a vertex $v$ in the optimal path to $y$ in fringe.

$s$ in fringe with key $f(s) = g(x) + \text{cost}(x, s) + h(s)$.

$v$ in fringe with key $f(v) = g(v) + h(v)$.

$h(v) - h(s) \leq \text{pathCost}(v, s)$ by induction.

$g(v) + \text{pathCost}(v, s) < g(x) + \text{cost}(x, s)$

$\implies f(v) < f(s)$.

But then $v$ would have been expanded before $s$!
Optimality

Tree search:
  A* is optimal if heuristic is admissible.
  UCS is a special case ($h = 0$)

Graph search:
  A* optimal if heuristic is consistent
  UCS optimal ($h = 0$ is consistent)

Consistency implies admissibility

In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems
A*: Summary
A*: Summary

A* uses both backward costs and (estimates of) forward costs
A* is optimal with admissible / consistent heuristics
Heuristic design is key: often use relaxed problems
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
end
function $\text{GRAPH-SEARCH}(\text{problem}, \text{fringe})$ return a solution, or failure

$\text{closed} \leftarrow \text{an empty set}$

$\text{fringe} \leftarrow \text{INSERT}(\text{MAKE-NODE}([\text{INITIAL-STATE}][\text{problem}]), \text{fringe})$

loop do
  if $\text{fringe}$ is empty then return failure

  $\text{node} \leftarrow \text{REMOVE-FRONT}(\text{fringe})$

  if $\text{GOAL-TEST}(\text{problem}, \text{STATE}[[\text{node}])]$ then return $\text{node}$

  if $\text{STATE}[[\text{node}]]$ is not in $\text{closed}$ then
    add $\text{STATE}[[\text{node}]]$ to $\text{closed}$
    for $\text{child-node}$ in $\text{EXPAND(STATE}[[\text{node}]], \text{problem})$ do
      $\text{fringe} \leftarrow \text{INSERT}(\text{child-node}, \text{fringe})$
    end
  end
end
Yaay!