Search

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

Search Heuristics

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

Greedy Search
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

What can go wrong?
Greedy: 140 + 99 + 211 = 450
Better: 140+80+97+101 = 418

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?

\[ S \quad h = 3 \]
\[ A \quad h = 2 \]
\[ B \quad h = 1 \]
\[ G \quad h = 0 \]

G in queue when B expanded.
No: only stop when we dequeue a goal.

Is \( A^* \) Optimal?

\[ h = 6 \]
\[ h = 7 \]
\[ h = 0 \]

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

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.
  - Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe
  - Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

- Largest out of place pancake.
  - Admissible? No. For number of flips.
  - Yes. For height of stack.

- Manhattan distance.
  - Admissible? Yes!

- Euclidean Distance.
  - Admissible? Yes.

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

Proof:

- **Definition of f-cost:** $g(n) + h(n)$
- **Admissibility of $h(n)$:** $h(n) \leq h^*(n)$
  $\Rightarrow$ $h = 0$ 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 since $f(n) = g(n) + h(n) < f(A)$.

Since $g(A) + 0 < g(B) + h(B)$

All ancestors of A expand before B. A expands before B. $\rightarrow$ A* search is optimal.
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 (L3D6)]
Creating Heuristics

Manhattan Distance: 15
Straight Line Distance: 366

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

\[
\begin{align*}
\text{UCS} & \quad 4 \text{ steps} & 8 \text{ steps} & 12 \text{ steps} \\
\text{TILES} & \quad 112 & 6300 & 3.6 \times 10^6
\end{align*}
\]

8 Puzzle II

Easier 8-puzzle: tile could slide any direction at any time, ignoring other tiles.

Total Manhattan distance
Why is it admissible? \( h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \).
Average nodes expanded when the optimal path has:

\[
\begin{align*}
\text{TILES} & \quad 13 & 39 & 227 \\
\text{MANHATTAN} & \quad 12 & 25 & 73
\end{align*}
\]
8 Puzzle III

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

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

Trivial Heuristics, Dominance

Dominance: \( h_a \geq h_c \) if
\[
\forall n: h_a(n) \geq h_c(n)
\]

Graph Search

In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)

Tree Search: Extra Work!

Failure to detect repeated states can cause exponentially more work.

Search Tree
State Graph
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?

<table>
<thead>
<tr>
<th>State space graph</th>
<th>Search tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
</tr>
<tr>
<td>E</td>
<td>H</td>
</tr>
</tbody>
</table>

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.

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.
\(g(v) + \text{pathCost}(v, s) < g(x) + \text{cost}(x, s)\)
\(\Rightarrow f(v) < f(s)\).
But then \(v\) would have been expanded before \(s\)!

Consistency of Heuristics

- **Main idea**: est. heuristic costs \(\leq\) actual costs
- **Admissibility**: \(h(x) \leq\) cost to goal.
- **Consistency**: \(h(x) - h(y) \leq \text{cost}(x, y)\).
- heuristic "arc" cost \(\leq\) actual arc cost
- Consistent \(\Rightarrow\) admissible? Yes? No?
- Consistent: f value along a path never decreases

Admissible:
\(f(C) = h(C) + 1 = 3\).
\(f(A) = h(A) = 4\).
Consistent: \(f(A) = 2 < 3 = f(C)\).

Claim: If \(y\) is expanded due to \(x\), \(f(y) \geq f(x)\).
Proof:
\(f(y) = g(x) + \text{cost}(x, y) + h(y)\)
\(\geq g(x) + h(x) - h(y) + h(y) = g(x) + h(x) = f(x)\)

The "estimate" of plan cost keeps rising as you progress.
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* uses both backward costs and (estimates of) forward costs
A* is optimal with admissible / consistent heuristics
Heuristic design is key: often use relaxed problems

Tree Search Pseudo-Code

```plaintext
function TREE-SEARCH(problem, f) returns a solution, or failure
    closed ← empty set
    fringe ← [INCOMPLETE-STATE(problem), fringe]
    loop do
        if fringe is empty; then return failure
        node ← fringe.removeMin()
        if is-solution(node) then return node
        for child in children(node) do
            fringe.add(node with cost to child)
    end
end
```

Graph Search Pseudo-Code

```plaintext
function GRAPH-SEARCH(problem, f) returns a solution, or failure
    closed ← empty set
    fringe ← [INCOMPLETE-STATE(problem), fringe]
    loop do
        if fringe is empty; then return failure
        node ← fringe.removeMin()
        if is-solution(node) then return node
        if closed contains fringe then return failure
        if fringe contains node then return failure
        for child in children(node) do
            fringe.add(node with cost to child)
    end
end
```

Yaay!