function $\text{TREE-SEARCH}(\text{problem, strategy})$ returns a solution, or failure
initialize the search tree using the initial state of $\text{problem}$
loop do
  if there are no candidates for expansion then return failure
  choose a leaf node for expansion according to $\text{strategy}$
  if the node contains a goal state then return the corresponding solution
  else expand the node and add the resulting nodes to the search tree
end

- Main variations:
  - Which leaf node to expand next
  - Whether to check for repeated states
  - Data structures for frontier, expanded nodes
Depth-First Search
Depth-First Search

Strategy: expand a deepest node first

Implementation:
Frontier is a LIFO stack
Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

Cartoon of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth
- solutions at various depths

Number of nodes in entire tree?
- $1 + b + b^2 + \ldots + b^m = O(b^m)$
Depth-First Search (DFS) Properties

- **What nodes does DFS expand?**
  - Some left prefix of the tree down to depth \( m \).
  - Could process the whole tree!
  - If \( m \) is finite, takes time \( O(b^m) \)

- **How much space does the frontier take?**
  - Only has siblings on path to root, so \( O(bm) \)

- **Is it complete?**
  - \( m \) could be infinite
  - Preventing cycles may help (more later)

- **Is it optimal?**
  - No, it finds the “leftmost” solution, regardless of depth or cost
Breadth-First Search
**Breadth-First Search**

**Strategy:** expand a shallowest node first

**Implementation:** Frontier is a FIFO queue
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be $s$
  - Search takes time $O(b^s)$

- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$

- Is it complete?
  - $s$ must be finite if a solution exists, so yes!

- Is it optimal?
  - If costs are equal (e.g., 1)
Iterative Deepening

- Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution...
  - Run a DFS with depth limit 2. If no solution...
  - Run a DFS with depth limit 3. .....  

- Isn’t that wastefully redundant?
  - Generally most of the work happens in the lowest level searched, so it’s not so bad!
  - Extra work is $O(b^{s-1})$
Uniform Cost Search
Uniform Cost Search

\( g(n) = \text{cost from root to } n \)

Strategy: expand lowest \( g(n) \)

Frontier is a priority queue sorted by \( g(n) \)

![Diagram of a graph with labeled nodes and edges showing cost contours and priority queue structure.](image_url)
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Expands all nodes with cost less than cheapest solution!
  - If that solution costs $C^*$ and arcs cost at least $\varepsilon$, then the “effective depth” is roughly $C^*/\varepsilon$
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)

- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

- Is it complete?
  - Assuming $C^*$ is finite and $\varepsilon > 0$, yes!

- Is it optimal?
  - Yes! (Proof next lecture via A*)
Assume known, discrete, observable, deterministic, atomic
Search problems defined by $S, s_0, A(s), \text{Result}(s,a), G(s), c(s,a,s')$
Search algorithms find action sequences that reach goal states
- Optimal = minimum-cost

Search algorithm properties:
- Depth-first: incomplete, suboptimal, space-efficient
- Breadth-first: complete, (sub)optimal, space-prohibitive
- Iterative deepening: complete, (sub)optimal, space-efficient
- Uniform-cost: complete, optimal, space-prohibitive
CS 188: Artificial Intelligence

Informed Search

Instructors: Stuart Russell and Dawn Song

University of California, Berkeley
Example: route-finding in Romania
What we would like to have happen

Guide search *towards the goal* instead of *all over the place*

Informed

Uninformed
A*: the core idea

- Expand a node $n$ most likely to be on an optimal path
- Expand a node $n$ s.t. the cost of the best solution through $n$ is optimal
- Expand a node $n$ with lowest value of $g(n) + h^*(n)$
  - $g(n)$ is the cost from root to $n$
  - $h^*(n)$ is the optimal cost from $n$ to the closest goal
- We seldom know $h^*(n)$ but might have a heuristic approximation $h(n)$
- $A^*$ = tree search with priority queue ordered by $f(n) = g(n) + h(n)$
Example: route-finding in Romania

\[ h(n) = \text{straight-line distance to Bucharest} \]
Example: pathing in Pacman

- $h(n) = \text{Manhattan distance} = |\Delta x| + |\Delta y|$
- Is Manhattan better than straight-line distance?
Is A* Optimal?

What went wrong?
- *Actual* bad solution cost < *estimated* good solution cost
- We need estimates to be less than actual costs!
Admissible Heuristics
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

- Example:

- Finding good, cheap admissible heuristics is the key to success
Optimality of A* Tree Search
Optimality of A* Tree Search

Assume:
- $A$ is an optimal goal node
- $B$ is a suboptimal goal node
- $h$ is admissible

Claim:
- $A$ will be chosen for expansion before $B$
Optimality of A* Tree Search: Blocking

Proof:

- Imagine $B$ is on the frontier
- Some ancestor $n$ of $A$ is on the frontier, too (maybe $A$ itself!)
- Claim: $n$ will be expanded before $B$
  1. $f(n) \leq f(A)$

\[ f(n) = g(n) + h(n) \]
\[ f(n) \leq g(A) \]
\[ g(A) = f(A) \]

Definition of $f$-cost

Admissibility of $h$

$h = 0$ at a goal
Optimality of A* Tree Search: Blocking

Proof:

- Imagine $B$ is on the frontier
- Some ancestor $n$ of $A$ is on the frontier, too (maybe $A$ itself!)
- Claim: $n$ will be expanded before $B$
  1. $f(n) \leq f(A)$
  2. $f(A) < f(B)$

$g(A) < g(B)$  
Suboptimality of $B$

$f(A) < f(B)$  
$h = 0$ at a goal
Proof:

- Imagine $B$ is on the frontier
- Some ancestor $n$ of $A$ is on the frontier, too (maybe $A$ itself!)
- Claim: $n$ will be expanded before $B$
  1. $f(n) \leq f(A)$
  2. $f(A) < f(B)$
  3. $n$ is expanded before $B$
- All ancestors of $A$ are expanded before $B$
- $A$ is expanded before $B$
- $A^*$ tree search is optimal
UCS vs A* Contours

- Uniform-cost expands equally in all “directions”

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

Greedy (h)  Uniform Cost (g)  A* (g+h)
A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- Protein design
- Chemical synthesis
- ...

...
Creating Heuristics

YOU GOT HEURISTIC UPGRADE!
Creating Admissible Heuristics

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

Problem $P_2$ is a relaxed version of $P_1$ if $A_2(s) \supseteq A_1(s)$ for every $s$.

Theorem: $h_2^*(s) \leq h_1^*(s)$ for every $s$, so $h_2^*(s)$ is admissible for $P_1$. 

366
Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- What are the step costs?
8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$

Start State

Goal State

Average nodes expanded when the optimal path has...

<table>
<thead>
<tr>
<th></th>
<th>...4 steps</th>
<th>...8 steps</th>
<th>...12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCS</td>
<td>112</td>
<td>6,300</td>
<td>$3.6 \times 10^6$</td>
</tr>
<tr>
<td>A*TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>

Statistics from Andrew Moore
What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

**Total Manhattan distance**

**Why is it admissible?**

\[ h(\text{start}) = 3 + 1 + 2 + \ldots = 18 \]

<table>
<thead>
<tr>
<th>Average nodes expanded when the optimal path has...</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
<tr>
<td>A*MANHATTAN</td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>
Combining heuristics

- **Dominance**: \( h_1 \geq h_2 \) if
  \[
  \forall n \quad h_1(n) \geq h_2(n)
  \]
  - Roughly speaking, larger is better as long as both are admissible
  - The zero heuristic is pretty bad (what does A* do with h=0?)
  - The exact heuristic is pretty good, but usually too expensive!

- **What if we have two heuristics, neither dominates the other?**
  - Form a new heuristic by taking the max of both:
    \[
    h(n) = \max( h_1(n), h_2(n) )
    \]
  - Max of admissible heuristics is admissible and dominates both!
Example: Knight’s moves

- Minimum number of knight’s moves to get from A to B?
  - $h_1 = \text{(Manhattan distance)}/3$
    - $h_1' = h_1$ rounded up to correct parity (even if A, B same color, odd otherwise)
  - $h_2 = \text{(Euclidean distance)}/\sqrt{5}$ (rounded up to correct parity)
  - $h_3 = \text{(max x or y shift)}/2$ (rounded up to correct parity)
  - $h(n) = \max(h_1'(n), h_2(n), h_3(n))$ is admissible!
Optimality of A* Graph Search

This part is a bit fiddly, sorry about that.
A* Graph Search Gone Wrong?

State space graph

Search tree

Simple check against expanded set blocks C
Fancy check allows new C if cheaper than old but requires recalculating C’s descendants
Consistency of Heuristics

- **Main idea:** estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal
    \[ h(A) \leq h^*(A) \]
  - Consistency: heuristic “arc” cost ≤ actual cost for each arc
    \[ h(A) - h(C) \leq c(A,C) \]
    or \[ h(A) \leq c(A,C) + h(C) \] (triangle inequality)
  - Note: \( h^* \) necessarily satisfies triangle inequality

- **Consequences of consistency:**
  - The \( f \) value along a path never decreases:
    \[ h(A) \leq c(A,C) + h(C) \implies g(A) + h(A) \leq g(A) + c(A,C) + h(C) \]
  - A* graph search is optimal
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, nodes that reach s optimally are expanded before nodes that reach s suboptimally
  - Result: A* graph search is optimal
Optimality

- Tree search:
  - A* is optimal if heuristic is admissible

- Graph search:
  - A* optimal if heuristic is consistent

- Consistency implies admissibility

- Most natural admissible heuristics tend to be consistent, especially if from relaxed problems
But...

- A* keeps the entire explored region in memory
- => will run out of space before you get bored waiting for the answer
- There are variants that use less memory (Section 3.5.5):
  - IDA* works like iterative deepening, except it uses an $f$-limit instead of a depth limit
    - On each iteration, remember the smallest $f$-value that exceeds the current limit, use as new limit
    - Very inefficient when $f$ is real-valued and each node has a unique value
  - RBFS is a recursive depth-first search that uses an $f$-limit = the $f$-value of the best alternative path available from any ancestor of the current node
    - When the limit is exceeded, the recursion unwinds but remembers the best reachable $f$-value on that branch
  - SMA* uses *all available memory* for the queue, minimizing thrashing
    - When full, drop worst node on the queue but remember its value in the parent
A*: Summary

- A* orders nodes in the queue by $f(n) = g(n) + h(n)$
- A* is optimal for trees/graphs with admissible/consistent heuristics
- Heuristic design is key: often use relaxed problems