Lecture 6 and 7

• Chapter 4 – making use of problem details beyond problem specification
• More efficient than uninformed search
• Uninformed search approaches can enhance basic A* algorithm
• This lecture, basic A* with examples
  – Robot navigation, Towers of Hanoi, Rubik’s cube
• Valentine’s day homework
• Next lecture, advanced A* and iterative algorithms
Best First Search

- Take the basic tree search algorithm
- $f(n)$ evaluation function determines expansion order
- $f(n)=g(n)+h(n)$

```plaintext
function Tree-Search(problem, fringe) returns 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] applied to State(node) succeeds return node
        fringe ← InsertAll(Expand(node, problem), fringe)

function Expand(node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in Successor-Fn[problem](State[node]) do
        s ← a new Node
        Parent-Node[s] ← node; Action[s] ← action; State[s] ← result
        Path-Cost[s] ← Path-Cost[node] + STEP-COST(node, action, s)
        Depth[s] ← Depth[node] + 1
        add s to successors
    return successors
```
Simplest Informed Search

• \( f(n) = h(n) \)
  – \( h \) is some heuristic function that estimates cost from \( n \) to goal, \( h(n) = 0 \), if \( n = \) goal

• Results in a greedy search technique that finds local minima
Greedy Algorithm and Properties

Evaluation function $h(n)$ (heuristic)

$= \text{estimate of cost from } n \text{ to goal}$

E.g., $h_{\text{SLD}}(n) = \text{straight-line distance from } n \text{ to Bucharest}$

Greedy search expands the node that \textit{appears} to be closest to goal

\underline{Complete}\textit{??}

\underline{Time}\textit{??}

\underline{Space}\textit{??}

\underline{Optimal}\textit{??}
A* in Relationship to Other Search Algorithms

- What if $h(\cdot) = 0$ for every state?
  - Uniform-cost search (Dijkstra’s algorithm)
- What if we make $g(\cdot) = 0$?
  - Greedy search
- A* combines the best properties of uniform-cost and greedy search
A* Search

Idea: avoid expanding paths that are already expensive

Evaluation function \( f(n) = g(n) + h(n) \)

\( g(n) \) = cost so far to reach \( n \)
\( h(n) \) = estimated cost to goal from \( n \)
\( f(n) \) = estimated total cost of path through \( n \) to goal

A* search uses an admissible heuristic
i.e., \( h(n) \leq h^*(n) \) where \( h^*(n) \) is the true cost from \( n \).

E.g., \( h_{SLD}(n) \) never overestimates the actual road distance

**Theorem:** A* search is optimal

A* complete and optimal, proof of optimality is trivial for trees
A* graph search

1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes on OPEN do:
   (a) Select the node on OPEN w/ the lowest f-value.
   (b) Generate its successors.
   (c) For each successor do:
      i. If it hasn't been generated before (i.e., it's not in CLOSED) evaluate it, add it to OPEN, and record its parent.
      ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have. Then add node to CLOSED list.
Example on Graphs

Numbers in parentheses are \( h(n) \)

Numbers on edges are operator costs
Completeness of A*

- A* expands nodes in increasing order of $f$.
- So it must eventually reach a goal state unless there are infinitely many nodes with $f(n) < f^*$, which is possible only if
  - there is a node with an infinite branching factor
  - there is a path with a finite path cost but an infinite number of nodes along it.

A* is complete on locally finite graphs (graphs with a finite branching factor) provided there is some positive constant $\delta$ such that every operator costs at least $\delta$. 
Optimality of A*

- We want to prove that A* is optimal (i.e., it will always find the best solution).

- Terminology.
  - $f^*$ is the cost of the optimal solution path
  - $f^* \geq f(n)$ because the heuristic is admissible

- Will use a proof by contradiction.

- Remember that all goal states have $h(n) = 0$. 
Let $G$ be an optimal goal state and $G_2$ be a suboptimal goal state. Assume that $n$ is on the optimal path to $g$ and has not been expanded. Assume that $G_2$ is about to be expanded.

Since $h$ is admissible: $f^* \geq f(n)$
If $n$ is not chosen for expansion, then
$$f(n) \geq f(G_2).$$
Therefore
$$f^* \geq f(G_2)$$
But, since $G_2$ is a goal state
$$h(G_2) = 0$$
$$f(G_2) = g(G_2)$$
$$f^* \geq g(G_2)$$
This contradicts the assumption that $G_2$ is suboptimal. Therefore, $A^*$ never selects a suboptimal goal for expansion.
A* is maximally efficient

- For a given heuristic function, no optimal algorithm is guaranteed to do less work.
- Aside from ties in f( ), A* expands every node necessary for the proof that we’ve found the shortest path, and no unnecessary nodes.
Properties of heuristics

- The efficiency of A* depends on the quality of the heuristic that guides the search.
- Given two admissible heuristics h1 and h2, if h1(n) ≥ h2(n) for all n, then h1 dominates h2.
  - If h1 dominates h2, then A* guided by h1 expands fewer nodes (and so is more efficient) than A* guided by h2.
- Consistent (monotone) heuristic
  - the f() value never decreases along any path.
  - use of a consistent heuristic improves the efficiency of A* graph search because it ensures that once you expand a node, you never have to re-expand it due to finding a better path to that node.
A* Search Properties

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand $f_{i+1}$ until $f_i$ is finished
8 Tile Puzzle and A* Search

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]
\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

\[ h_1(S) = ?? \quad 7 \]
\[ h_2(S) = ?? \quad 2+3+3+2+4+2+0+2 = 18 \]
Dominance of Search

If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$ and is better for search

Typical search costs:

$d = 14$  $\text{IDS} = 3,473,941$ nodes

$A^*(h_1) = 539$ nodes

$A^*(h_2) = 113$ nodes

$d = 14$  $\text{IDS} = \text{too many nodes}$

$A^*(h_1) = 39,135$ nodes

$A^*(h_2) = 1,641$ nodes
Inventing Admissible Heuristics

- List constraints
- Relax constraints to obtain admissible heuristics (constraints increase cost)
Examples

• For each, specify the state space, operators, constraints of the problem and admissible heuristics
  – Towers of Hanoi
  – Robot path planning
  – TSP
  – Rubik’s Cube
Towers of Hanoi
Valentine’s Weekend Homework

• Formulate the TSP problem or your problem from homework #1 as a search problem.
• Define the state space, constraints, goal test, operators
• Relax one or more constraints and state an admissible heuristic or state why no admissible heuristic exists