Informed Search

Idea: be **smart** about what paths to try.
Expanding a Node

How should we implement this?

successor list
Blind Search vs. Informed Search

• What’s the difference?

• How do we formally specify this?
General Tree Search Paradigm
(adapted from Chapter 3)

function tree-search(root-node)
    fringe ← successors(root-node)
    while ( notempty(fringe) )
        {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node), fringe) }
    return failure
end tree-search

Does this look familiar?
function graph-search(root-node)
  closed ← { }
  fringe ← successors(root-node)
  while ( notempty(fringe) )
    {node ← remove-first(fringe)
     state ← state(node)
     if goal-test(state) return solution(node)
     ifnotin(state,closed)
       {add(state,closed)
        fringe ← insert-all(successors(node),fringe) }}
  return failure
end graph-search

What’s the difference between this and tree-search?
Tree Search or Graph Search

• What’s the key to the order of the search?
Best-First Search

- Use an evaluation function \( f(n) \).
- Always choose the node from fringe that has the lowest \( f \) value.
Best-First Search Example

```
3 5 1
```

```
4 6
```
Old Friends

- Breadth first = best first
  - with $f(n) = \text{depth}(n)$

- Dijkstra’s Algorithm = best first
  - with $f(n) = g(n)$
  - where $g(n) = \text{sum of edge costs from start to } n$
  - space bound (stores all generated nodes)
Heuristics

• What is a heuristic?

• What are some examples of heuristics we use?

• We’ll call the heuristic function $h(n)$. 
Greedy Best-First Search

- $f(n) = h(n)$
- What does that mean?
- Is greedy search optimal?
- Is it complete?
- What is its worst-case complexity for a tree with branching factor $b$ and maximum depth $m$?
A* Search

• Hart, Nilsson & Rafael 1968
  – Best first search with $f(n) = g(n) + h(n)$
    where $g(n) = \text{sum of edge costs from start to } n$
    and $h(n) = \text{estimate of lowest cost path } n\rightarrow\text{goal}$
  – If $h(n)$ is **admissible** then search will find optimal solution.

  \[
  \begin{align*}
  \text{Never overestimates the true cost of any solution which can be reached from a node.}
  \end{align*}
  \]

Space bound since the queue must be maintained.
Shortest Path Example

Start:

End:

Straight-line distance to Bucharest:
- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobrota: 242
- Eforie: 161
- Fagaras: 178
- Giurgiu: 77
- Hirsova: 151
- Iasi: 226
- Lugoj: 244
- Mehadia: 241
- Neamt: 234
- Oradea: 380
- Pitesti: 98
- Rimnicu Vilcea: 193
- Sibiu: 253
- Timisoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
A* Shortest Path Example

\[ 366 = 0 + 366 \]

**Arad**
A* Shortest Path Example

- Sibiu: 393 = 140 + 253
- Timisoara: 447 = 118 + 329
- Zerind: 449 = 75 + 374
A* Shortest Path Example
A* Shortest Path Example
A* Shortest Path Example
A* Shortest Path Example
8 Puzzle Example

• $f(n) = g(n) + h(n)$
• **What is the usual** $g(n)$?
• **two well-known** $h(n)$’s
  – $h_1 =$ the number of misplaced tiles
  – $h_2 =$ the sum of the distances of the tiles from their goal positions, using city block distance, which is the sum of the horizontal and vertical distances
8 Puzzle Using Number of Misplaced Tiles

\[
\begin{array}{cccc}
1 & 2 & 3 & \text{goal} \\
8 & 4 & & \\
7 & 6 & 5 & \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 8 & 3 \\
1 & 6 & 4 \\
7 & 5 & \\
\end{array}
\]
Continued
Suppose a suboptimal goal \( G_2 \) has been generated and is in the queue. Let \( n \) be an unexpanded node on the shortest path to an optimal goal \( G_1 \).

\[
f(n) = g(n) + h(n) \leq g(G_1) \]

\[
< g(G_2) \quad \text{G2 is suboptimal} \\
= f(G_2) \quad \text{f(G2) = g(G2)}
\]

So \( f(n) < f(G_2) \) and A* will never select \( G_2 \) for expansion.
Algorithms for A*

- Since Nilsson defined A* search, many different authors have suggested algorithms.
- Using Tree-Search, the optimality argument holds, but you search too many states.
- Using Graph-Search, it can break down, because an optimal path to a repeated state can be discarded if it is not the first one found.
- One way to solve the problem is that whenever you come to a repeated node, discard the longer path to it.
The Rich/Knight Implementation

• a node consists of
  – state
  – g, h, f values
  – list of successors
  – pointer to parent

• OPEN is the list of nodes that have been generated and had h applied, but not expanded and can be implemented as a priority queue.

• CLOSED is the list of nodes that have already been expanded.
1) /* Initialization */

OPEN <- start node

Initialize the start node

g:
h:
f:

CLOSED <- empty list
2) repeat until goal (or time limit or space limit)

- if OPEN is empty, fail
- BESTNODE <- node on OPEN with lowest f
- if BESTNODE is a goal, exit and succeed
- remove BESTNODE from OPEN and add it to CLOSED
- generate successors of BESTNODE
for each successor s do
  1. set its parent field
  2. compute g(s)
  3. if there is a node OLD on OPEN with the same state info as s
     { add OLD to successors(BESTNODE)
       if g(s) < g(OLD), update OLD and throw out s }

Rich/Knight
4. if (s is not on OPEN and there is a node \textbf{OLD} on CLOSED with the same state info as \textit{s} \\
{ add \textbf{OLD} to \texttt{successors(BESTNODE)} \\
if \texttt{g(s)} < \texttt{g(OLD)}, update \texttt{OLD},  
throw out \textit{s}, 
***propagate the lower costs to \texttt{successors(OLD)} } 

That sounds like a LOT of work. What could we do instead?
Rich/Knight

5. If \( s \) was not on OPEN or CLOSED
   
   \[
   \{ \text{add } s \text{ to OPEN} \\
   \text{add } s \text{ to successors(BESTNODE)} \\
   \text{calculate } g(s), h(s), f(s) \} 
   \]

end of repeat loop
The Heuristic Function $h$

- If $h$ is a **perfect estimator** of the true cost then $A^*$ will always pick the correct successor with no search.

- If $h$ is **admissible**, $A^*$ with TREE-SEARCH is guaranteed to give the optimal solution.

- If $h$ is **consistent**, too, then GRAPH-SEARCH without extra stuff is optimal.
  \[ h(n) \leq c(n,a,n') + h(n') \] for every node $n$ and each of its successors $n'$ arrived at through action $a$.

- If $h$ is not admissible, no guarantees, but it can work well if $h$ is not often greater than the true cost.