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?
General Graph Search Paradigm  
(adapted from Chapter 3)

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
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 $g(n) =$ sum of edge costs from start to $n$
    and $h(n) =$ estimate of lowest cost path $n-->goal$
  – If $h(n)$ is **admissible** then search will find optimal solution.
    
    Never overestimates the true cost of any solution which can be reached from a node.

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

Straight-line distance to Bucharest

- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobreta: 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

Arad
366 = 0 + 366
A* Shortest Path Example

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

![Diagram of A* shortest path example](image-url)
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
  - $h_1 = \text{the number of misplaced tiles}$
  - $h_2 = \text{the sum of the distances of the tiles from their goal positions, using city block distance, which is the sum of the horizontal and vertical distances}$
8 Puzzle Using Number of Misplaced Tiles

1  2  3
8  4
7  6  5

Goal

5+1=6

1st
283
164
7  5

0+4=4

2nd
283
164
7  5

5+1=6

5

283
164
7  5

283
164
7  5

5

283
164
7  5

283
164
7  5
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 \( G_1 \).

\[
f(n) = g(n) + h(n) \\
\leq g(G_1) \quad \text{Why?} \\
< g(G_2) \quad \text{G2 is suboptimal} \\
= f(G_2) \quad f(G_2) = g(G_2)
\]

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
Rich/Knight

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 $s$ }
4. if \((s)\) is not on OPEN and there is a node OLD on CLOSED with the same state info as \(s\)

\{ add OLD to successors(BESTNODE)

if \(g(s) < g(OLD)\), update OLD,

throw out \(s\),

***propagate the lower costs to successors(OLD) \}

That sounds like a LOT of work. What could we do instead?
5. If $s$ was not on OPEN or CLOSED
   
   \{ add $s$ to OPEN
     add $s$ to successors(BESTNODE)
     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 admissable, no guarantees, but it can work well if $h$ is not often greater than the true cost.