Informed (heuristic) search

- Assign a cost to each action/rule/arc

- Heuristic function:

  $h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state.

Example: 8-puzzle

Initial node n_I

5	4	
6	1	8
7	3	2

Goal node n_G

1	2	3
8		4
7	6	5

$h(n)$ = number of tiles out of place in node n

$h(n_I) = 7$

$h(n_G) = 0$

Example: 8-puzzle

Initial node n_I

5	4	
6	1	8
7	3	2

Goal node n_G

1	2	3
8		4
7	6	5

$h(n)$ = sum of Manhattan distance of tiles out of place

$h(n_I) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$

$h(n_G) = 0$

Greedy search algorithm

L $\leftarrow$ [start nodes]
while L $\neq$ empty do
    remove node with lowest h(n) value from L, call it p
    if p is a goal node, return(success)
    generate all successor states of p, and add them to L
endwhile
return(fail)

Greedy search algorithm

- Not guaranteed to find optimal path
- May not terminate

A* search: Minimizing total path cost

- Let

  $$f^*(n) = g^*(n) + h^*(n)$$

  be the cost of an optimal path going through node n,

  where

  $g^*(n)$ is cost of an optimal path from initial node to n,

  $h^*(n)$ is cost of an optimal path from n to a goal node

- But $f^*(n)$ is difficult to know, so we *estimate*

A* search: Minimizing total path cost

- Let

  $$f(n) = g(n) + h(n)$$

  be the cost of a path going through node n to a goal node, where

  $g(n)$ is cost of path from initial state to n that was found,

  $h(n)$ is a heuristic estimate of cost from n to a goal

Algorithm A* for searching a tree

```
L ← [start nodes]
while L ≠ empty do
    remove node with lowest f(n) value from L, call it p
    if p is a goal node, return(success)
    generate all successor states of p, and add them to L
endwhile
return(fail)
```

Admissible heuristics

Definition: A heuristic function h(n) is *admissible* if h(n) $\leq$ h*(n), for all n

Theorem: If h(n) is admissible, the A* algorithm is guaranteed to find an optimal path to a goal node

Dominating heuristics

Definition: Given admissible heuristics $h_1(n)$ and $h_2(n)$, $h_2(n)$ *dominates* if $h_1(n)$ if, for all nodes n,

$$h_2(n) \geq h_1(n),$$

and there exists a node n' such that

$$h_2(n') > h_1(n')$$

Theorem: A* with $h_1(n)$ expands at least as many nodes as A* with $h_2(n)$, if $h_2(n)$ dominates $h_1(n)$

Complexity results for A*

- Good news:
  - A* is complete, optimal, and optimally efficient
  - no algorithm with the same information can do better
- Bad news:
  - assume a single goal, search a tree, and each action is reversible
  - time complexity of A* is exponential,

    $$O((b^{\varepsilon})^{d}) = \mathrm{O}(b^{\varepsilon d})$$

    where ε is the maximum relative error $(h^*(n) - h(n)) / h^*(n)$
    b is the branching factor
    d is the depth of the goal node

Iterative-deepening A*

- Amount of memory needed is a problem for A*
- IDA*
  - each iteration is a complete DFS with a cutoff (so, no priority queue)
  - branch is cutoff (node is not added to L) if f(n) exceeds some threshold
  - next iteration sets threshold to be minimum of values that exceeded old threshold
  - time complexity depends strongly on number of different values the heuristic can take on