Informed search algorithms

Chapter 4

Outline

- *Review*
- Best-first search
- Greedy best-first search
- A^* search
- IDA*
- Heuristics
- Summary

Search Algorithms

- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

Tree Search

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to 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
```

Graph search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
   closed ← an empty set
   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](STATE[node]) then return SOLUTION(node)
      if STATE[node] is not in closed then
         add STATE[node] to closed
         fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Search strategies

- A search strategy is defined by picking the order of node expansion

Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

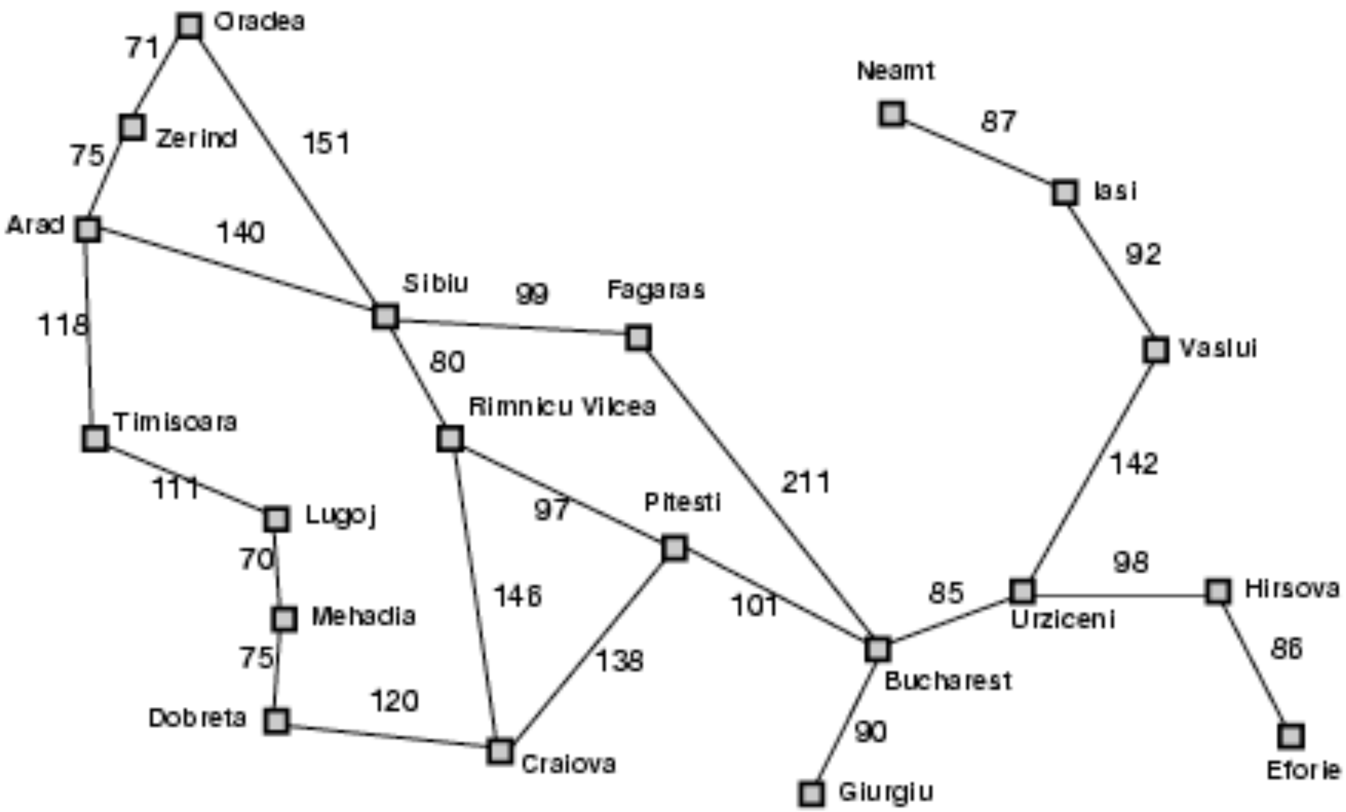
Outline

- **Review**
- *Best-first search*
- Greedy best-first search
- A^* search
- IDA*
- Heuristics
- Summary

Informed Search Strategies

- *Informed Search Strategies* use information that is in addition to the problem specification.
- In the informed strategies that we will look at, the additional information is in the form of a function that "guesses" how far a node is from the nearest goal node.

Romania with step costs in km



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"

  → Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability (i.e., the search strategy)

- Special cases:
  - greedy best-first search
  - A^* search
  - Iterative Deepening A* (IDA*)

Outline

- **Review**
- **Best-first search**
- *Greedy best-first search*
- A^* search
- IDA*
- Heuristics
- Summary

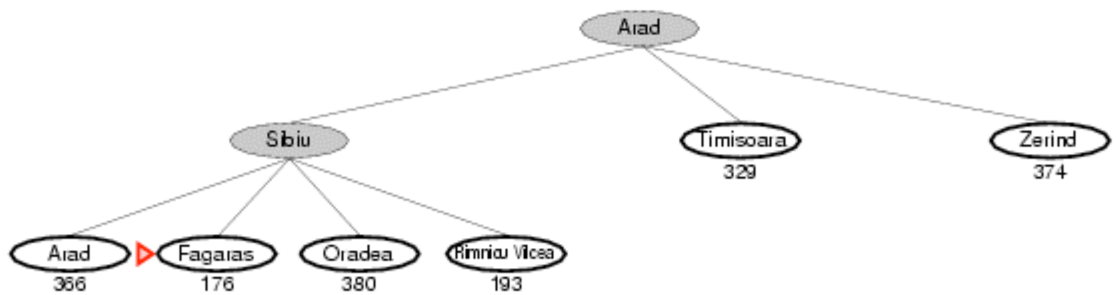
Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)
- = estimate of cost from n to *goal*
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal

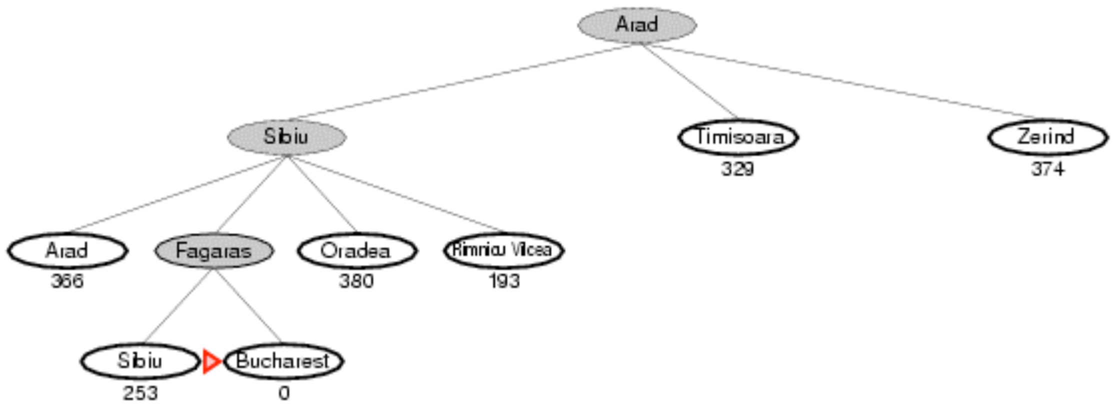
Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Greedy best-first search example



Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

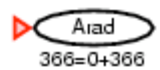
Outline

- **Review**
- **Best-first search**
- **Greedy best-first search**
- *A^* search*
- IDA*
- Heuristics
- Summary

A^* search

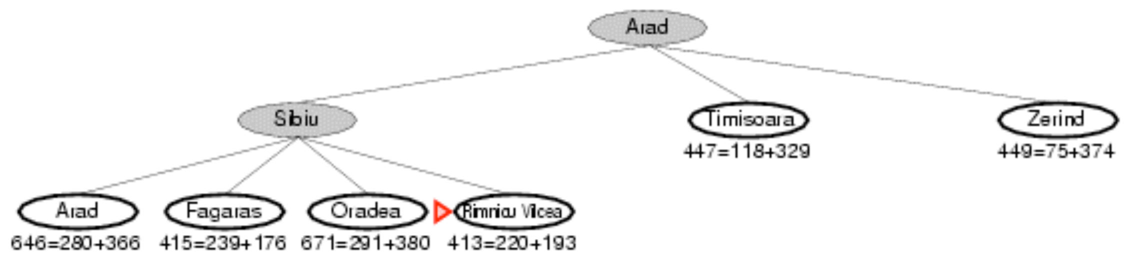
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

A* search example

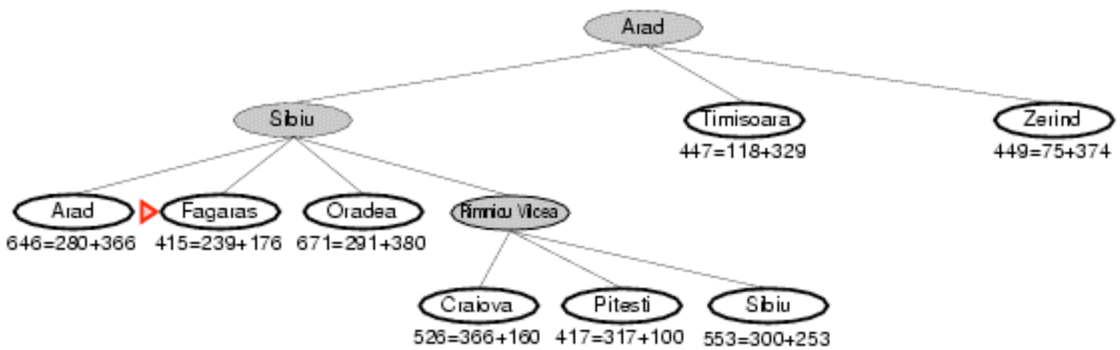


A* search example

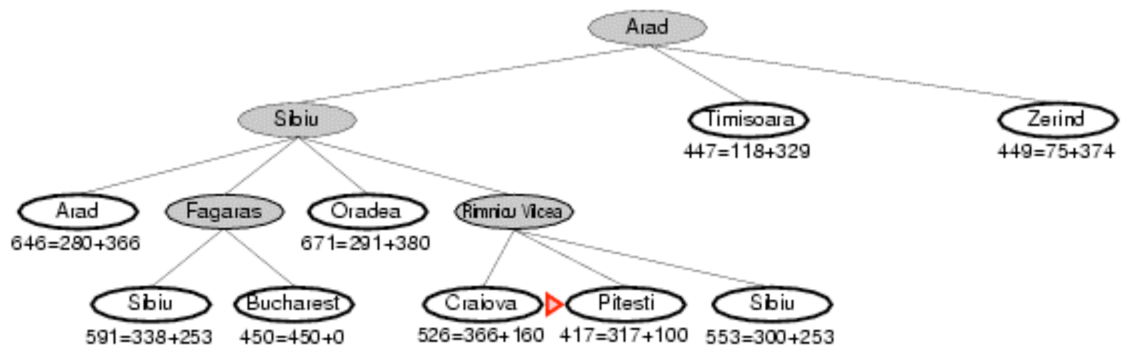
A* search example



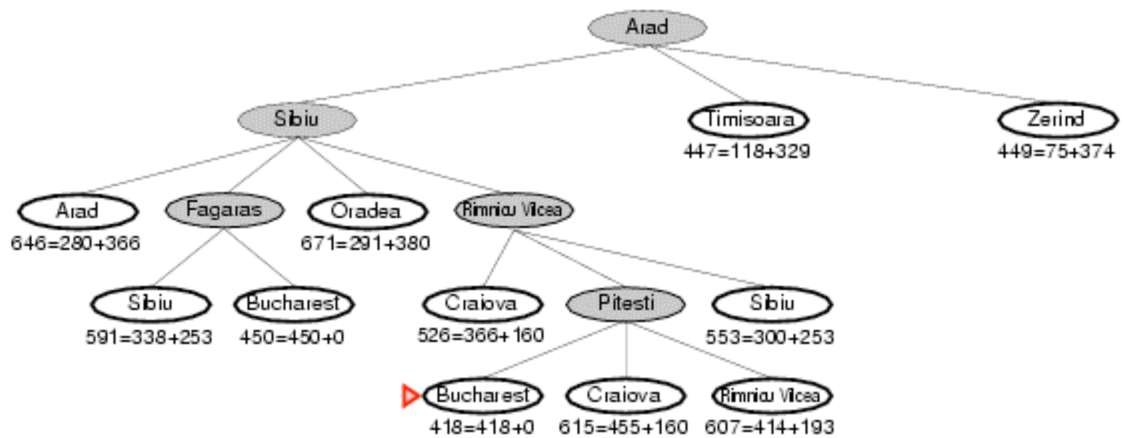
A* search example



A* search example



A* search example

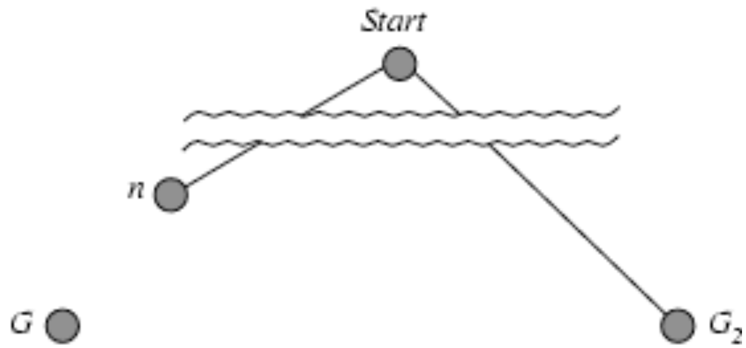


Admissible heuristics

- A heuristic *h(n)* is admissible if for every node *n*, $h(n) \le h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from *n*.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If *h(n)* is admissible, A* using `TREE-SEARCH` is optimal, however you need to keep track of the best path to every node.

Optimality of A* (proof)

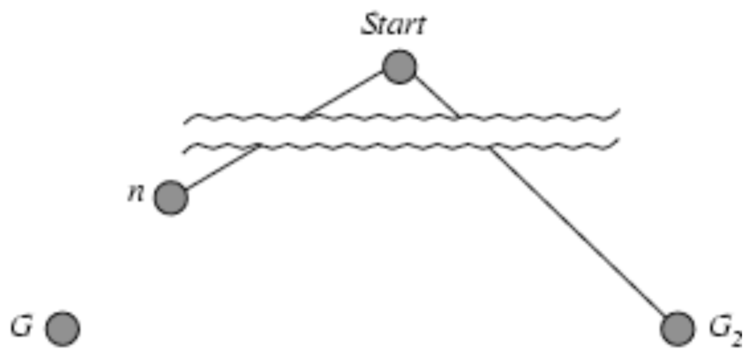
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A^* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



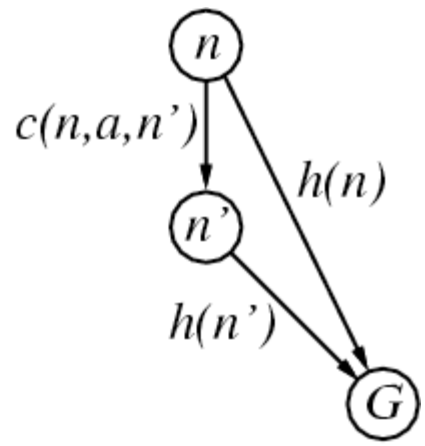
- $f(G_2)$ $> f(G)$ from above
- $h(n)$ $\leq h^*(n)$ since h is admissible
- $g(n) + h(n)$ $\leq g(n) + h^*(n)$
- $f(n)$ $\leq f(G)$

Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

  $h(n) \leq c(n,a,n') + h(n')$



- If *h* is consistent, we have

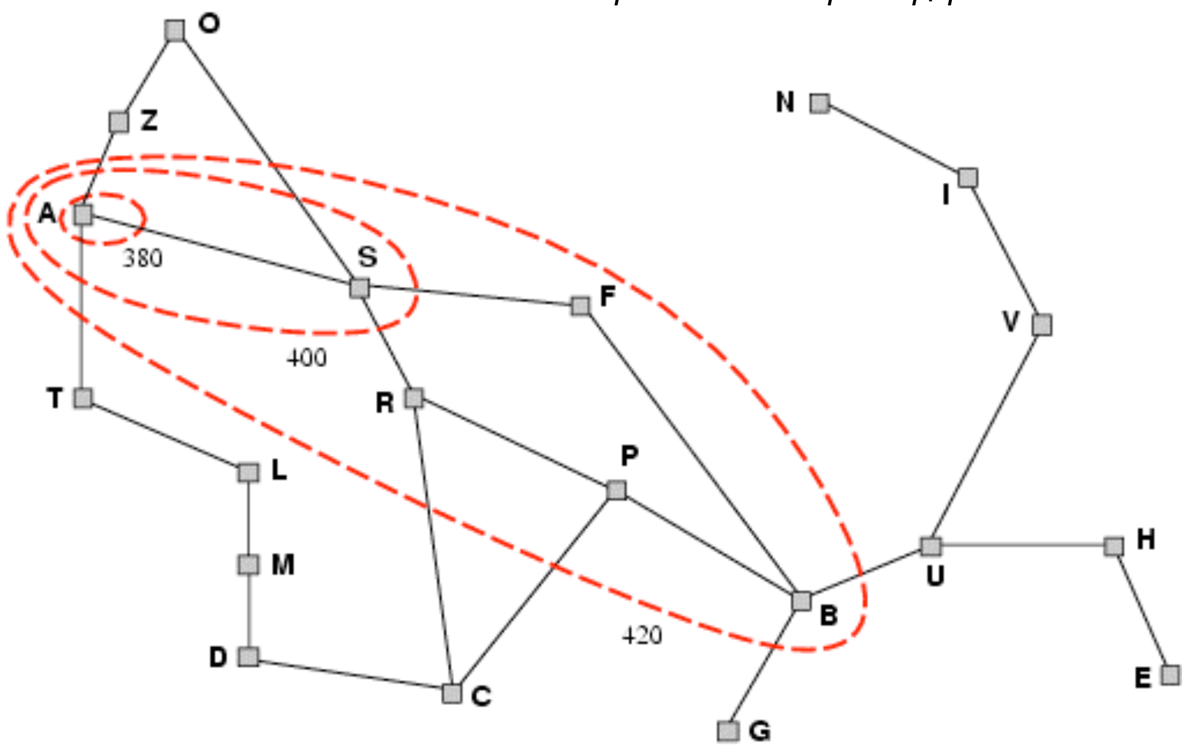
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

- i.e., *f(n)* is non-decreasing along any path.
- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal and the first time you reach a node, you have found the best path to that node!

Optimality of A*

- A* expands nodes in order of increasing *f* value

- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Properties of A*

- Complete? Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes
- Optimally efficient? Yes (sort of)

Outline

- **Review**
- **Best-first search**
- **Greedy best-first search**
- **A* search**
- *IDA****
- Heuristics
- Summary

IDA*

- Iterative Deepening A* is to A* what Iterative Deepening is to Breadth-first search.
- Instead of iterating on the depth, IDA* iterates on something called the *f-limit*.
- IDA* has all the optimality properties of A* but only uses linear space (because it does depth-first search).

Outline

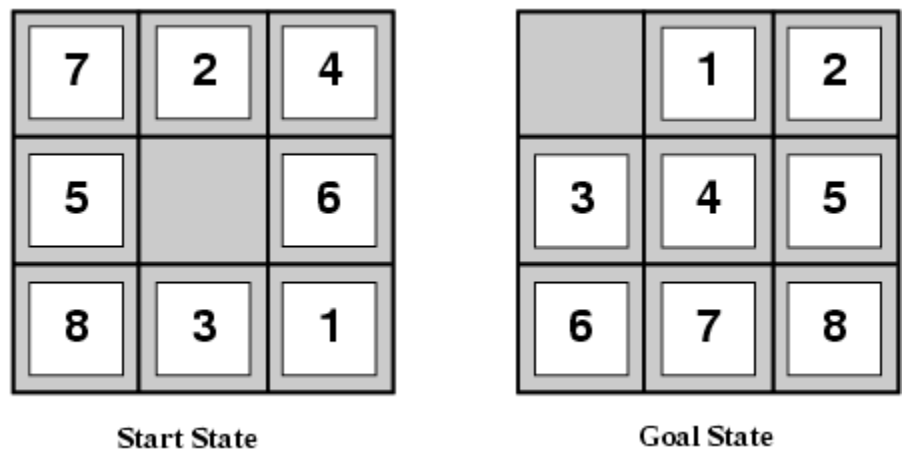
- **Review**
- **Best-first search**
- **Greedy best-first search**
- **A^* search**
- **IDA***
- *Heuristics*
- Summary

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

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



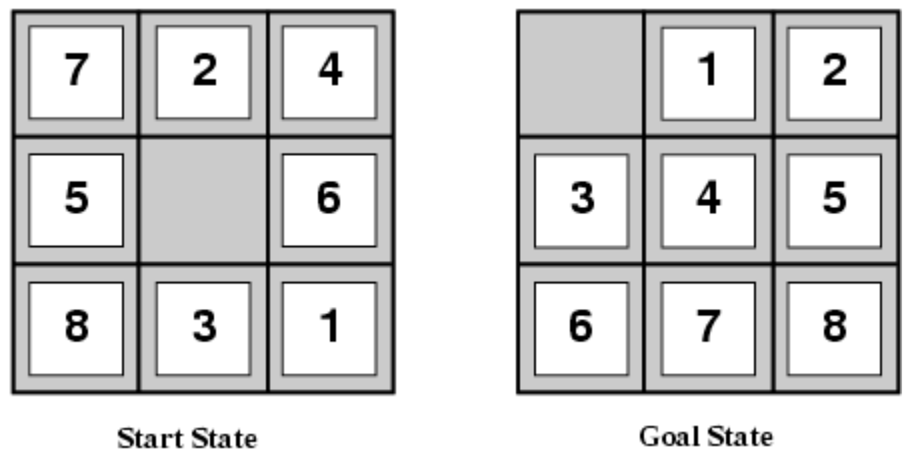
- $h_1(S) = ?$
- $h_2(S) = ?$

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

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



- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Dominance

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

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

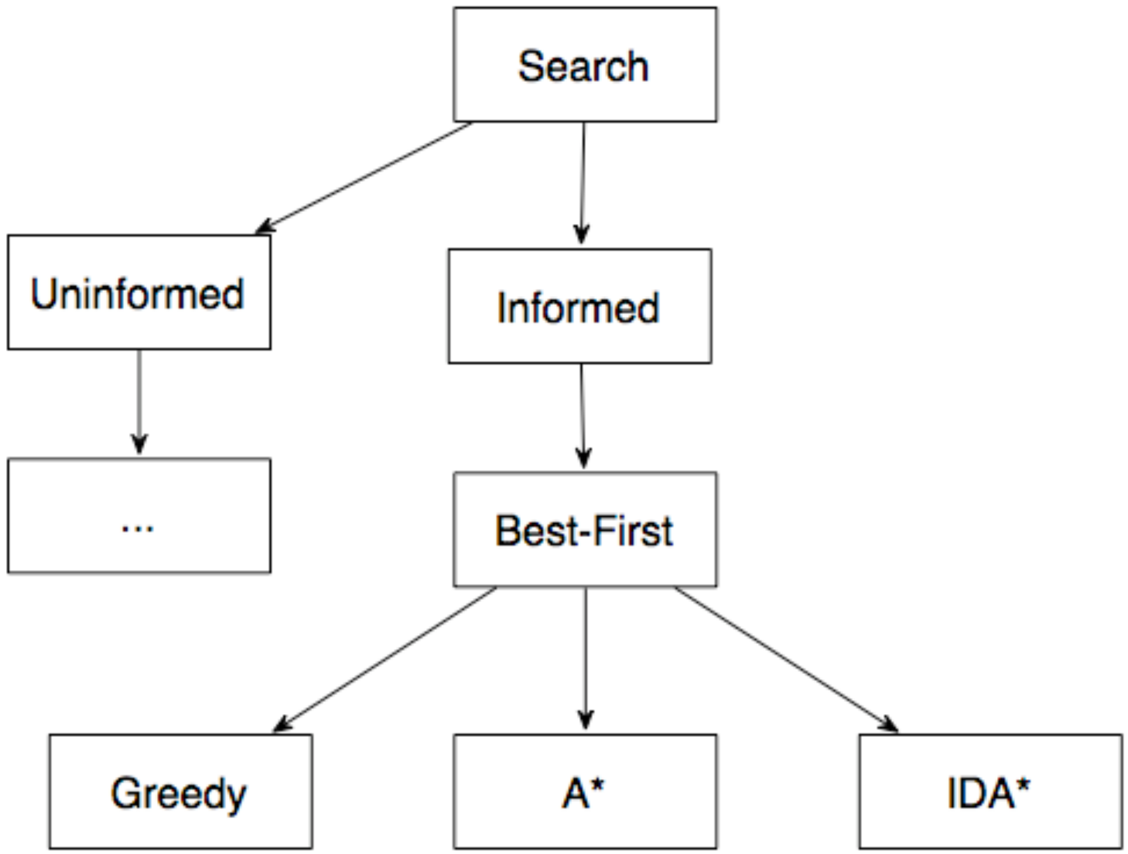
Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
- Note that while solutions to relaxed problems can never be longer than the solutions to the original problem, the effort to find a solution to a relaxed problem can be greater.

Outline

- **Review**
- **Best-first search**
- **Greedy best-first search**
- **A^* search**
- **IDA***
- **Heuristics**
- *Summary*

Search Hierarchy



Informed Search Strategies

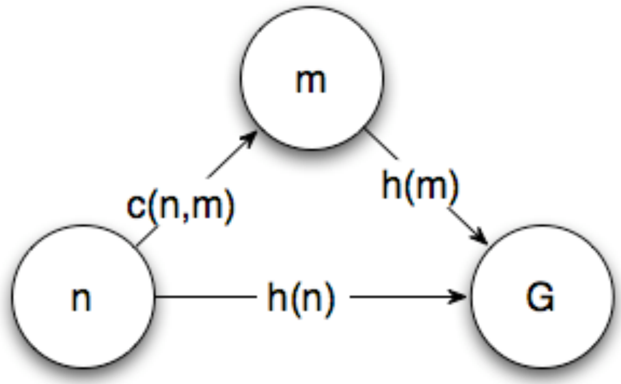
- Greedy: pick the next unexpanded node to expand based on how close it seems to be to its nearest goal state.
- A*: pick the next unexpanded node to expand based on how short is the optimal path through that node.
- IDA*: pick the next unexpanded node to expand based on the depth of the node.

A* & Admissibility

- A heuristic is *admissable* if it is guaranteed to never overestimate the distance from from that node to its nearest goal.
- A* is guaranteed to find an optimal solution if its heuristic is admissible.
- Admissible heuristic can be created by finding solutions to a relaxed version of the given problem.

A* & Consistency

- A *consistent* heuristic is one where the triangle inequality holds:



- In other words, $h(n) \leq h(m) + c(n,m)$
- If the heuristic is consistent then the graph version of A* always finds the shortest path to any node when it first finds that node.

A* & IDA*

- A* is optimal and optimally efficient, but uses a lot of space.
- IDA* is also optimal (and not quite optimally efficient) but only uses linear space.

The End