COMP3411/9814: Artificial Intelligence

Solving problems by searching
Informed Search

Tatjana Zrimec, 2020

Lecture Overview

- Heuristics
- Informed Search Methods
  - Best-first search
  - Greedy best-first search
  - A^* search
  - Iterative Deepening A^* Search

Informed (Heuristics) Searches

- **Informed search** strategy
  - one that uses problem-specific knowledge beyond the definition of the problem itself
  - can find solutions more efficiently than can an uninformed strategy

- **Uninformed** search algorithms—algorithms that are given no information about the problem other than its definition.
  - some of these algorithms can solve any solvable problem, none of them can do so efficiently
- **Informed** search algorithms, can do quite well given some guidance on where to look for solutions.

Informed (Heuristics) Searches

- Uninformed methods of search are capable of systematically exploring the state space in finding a goal state
  - However, uninformed search methods are very inefficient
- With the aid of problem-specific knowledge, informed methods of search are more efficient
- All implemented using a priority queue to store frontier nodes

Heuristics

- Heuristics are “rules of thumb”
- Heuristics are criteria, methods or principles for deciding which among several alternative courses of action promises to be the most effective in order to achieve some goal. “Heuristics” (Pearl 1984)
- Can make use of heuristics in deciding which is the most “promising” path to take during search
- In search, heuristic must be an underestimate of actual cost to get from current node to any goal — an admissible heuristic
- Denoted $h(n)$; $\underline{h(n)=0}$ when ever n is a goal node

Heuristics — Example

- 8-Puzzle — number of tiles out of place

2	8	3		1	2	3
1	6	4	→	8		4
	7	5		7	6	5

- Therefore $h(n)$=5

Heuristics — Example

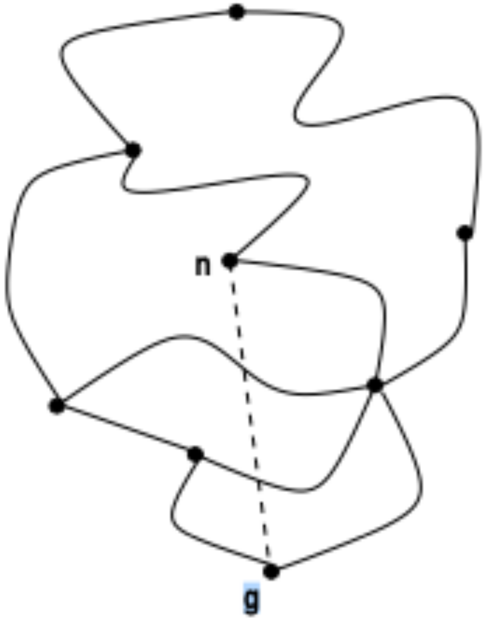
- 8-Puzzle — Manhattan distance (distance tile is out of place)

2	8	3		1	2	3
1	6	4	$\longrightarrow$	8		4
	7	5		7	6	5

- Therefore $h(n) = 1 + 1 + 0 + 0 + 0 + 1 + 1 + 2 = 6$

Heuristics — Example

- Another common heuristic is the straight-line distance (“as the crow flies”) from node to goal



- Therefore *h*(*n*)=distance from *n* to *g*

Heuristic Search

- Idea: don't ignore the goal when selecting paths.
- Often there is extra knowledge that can be used to guide the search: heuristics.
- h(n) is an estimate of the cost of the shortest path from node n to a goal node.
- h(n) needs to be efficient to compute.
- h can be extended to paths: $h(\langle n_0,\ldots,n_k\rangle) = h(n_k)$.
- h(n) is an underestimate if there is no path from n to a goal with cost less than h(n).
- An admissible heuristic is a nonnegative heuristic function that is an underestimate of the actual cost of a path to a goal.

Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, h(n) can be the straight-line distance from n to the closest goal.
- If the nodes are locations and cost is time, we can use the distance to a goal divided by the maximum speed.
- If the goal is to collect all of the coins and not run out of fuel, the cost is an estimate of how many steps it will take to collect the rest of the coins, refuel when necessary, and return to goal position.
- A heuristic function can be found by solving a simpler (less constrained) version of the problem.

Search Strategies

General Search algorithm:

- add initial state to queue
- repeat:
  - take node from front of queue
  - test if it is a goal state; if so, terminate
  - “expand” it, i.e. generate successor nodes and add them to the queue

Search strategies are distinguished by the order in which new nodes are added to the queue of nodes awaiting expansion.

Search Strategies

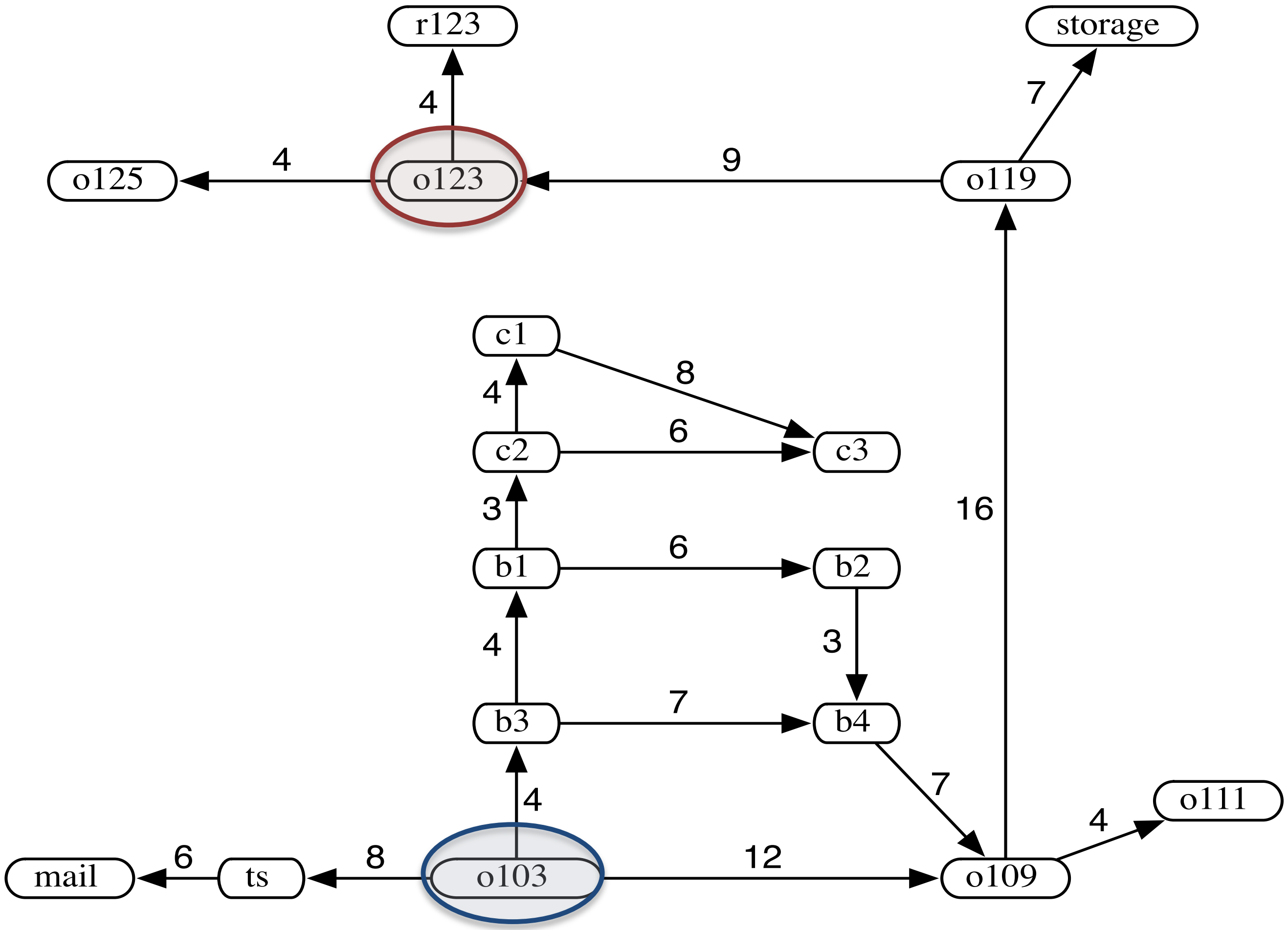
- BFS and DFS treat all new nodes the same way:
  - BFS add all new nodes to the back of the queue
  - DFS add all new nodes to the front of the queue
- Best First Search uses an evaluation function f () to order the nodes in the queue;
  - Similar to UCS $f(n)$ = cost $g(n)$ of path from root to node n
- Informed or Heuristic search strategies incorporate into f () an estimate of distance to goal
  - Greedy Search $f(n)$ = estimate $h(n)$ of cost from node n to goal
  - A∗ Search $f(n) = g(n) + h(n)$

Heuristic Function

There is a whole family of Best First Search algorithms with different evaluation functions *f* (). A key component of these algorithms is a heuristic function:

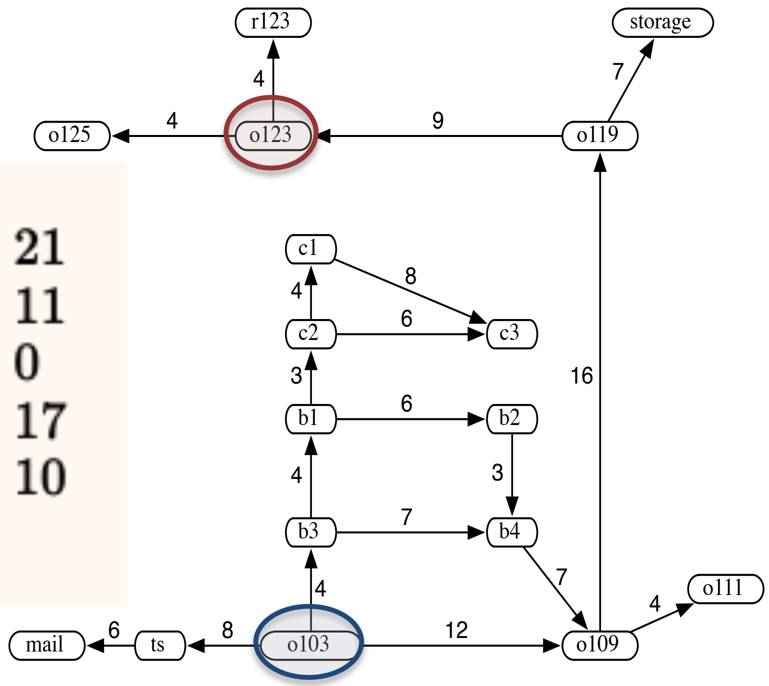
- Heuristic function *h*: {Set of nodes} → **R** :
  - *h*(*n*) = estimated cost of the cheapest path from current node *n* to *goal* node.
    - in the area of search, heuristic functions are problem specific functions that provide an estimate of solution cost.
    - nonnegative, with one constraint: if *n* is a goal node, then $h(n) = 0$.

State-Space Graph for the Delivery Robot



Delivery Robot - Heuristic Function

$$
\begin{aligned}
h(mail) &= 26 & h(ts) &= 23 & h(o103) &= 21\\
h(o109) &= 24 & h(o111) &= 27 & h(o119) &= 11\\
h(o123) &= 4 & h(o125) &= 6 & h(r123) &= 0\\
h(b1) &= 13 & h(b2) &= 15 & h(b3) &= 17\\
h(b4) &= 18 & h(c1) &= 6 & h(c2) &= 10\\
h(c3) &= 12 & h(storage) &= 12
\end{aligned}
$$



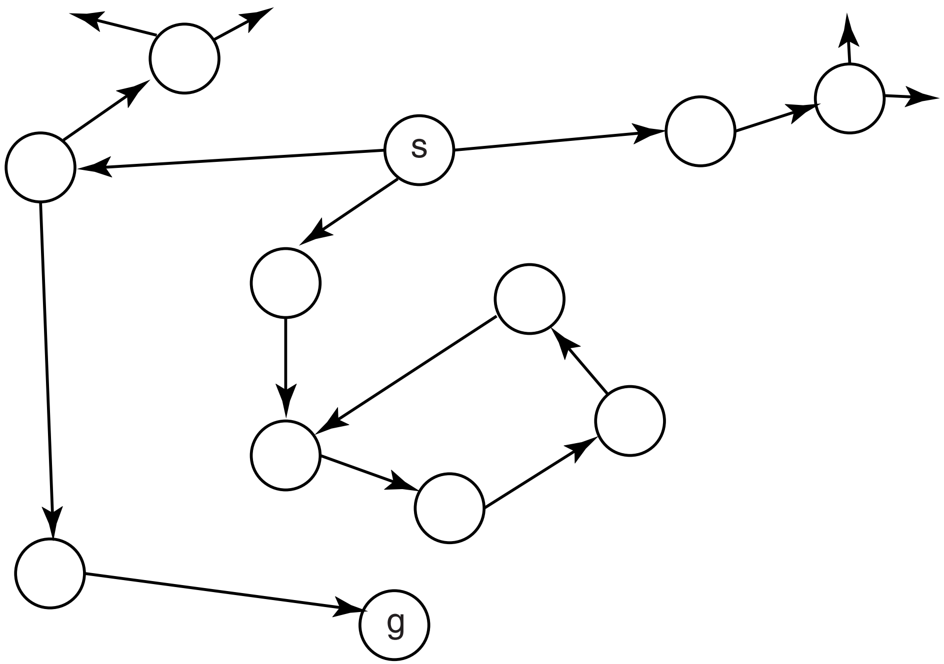
The h function can be extended to be applicable to paths by making the heuristic value of a path equal to the heuristic value of the node at the end of the path.

$$h(\langle n_o, \ldots, n_k \rangle) = h(n_k)$$

Greedy best-first search

- Idea: select the path whose end is closest to a goal according to the heuristic function.
- Greedy Best-First Search - Best-First Search selects the next node for expansion using the heuristic function for its evaluation function, i.e. $f(n) = h(n)$
  - $h(n)=0 \Rightarrow n$ is a goal state
  - i.e. greedy search minimizes the estimated cost to the goal; it expands whichever node n is estimated to be closest to the goal.
- It treats the frontier as a priority queue ordered by h.
- Greedy: tries to "bite off" as big a chunk of the solution as possible, without worrying about long-term consequences.

Illustrative Graph — Best-first Search

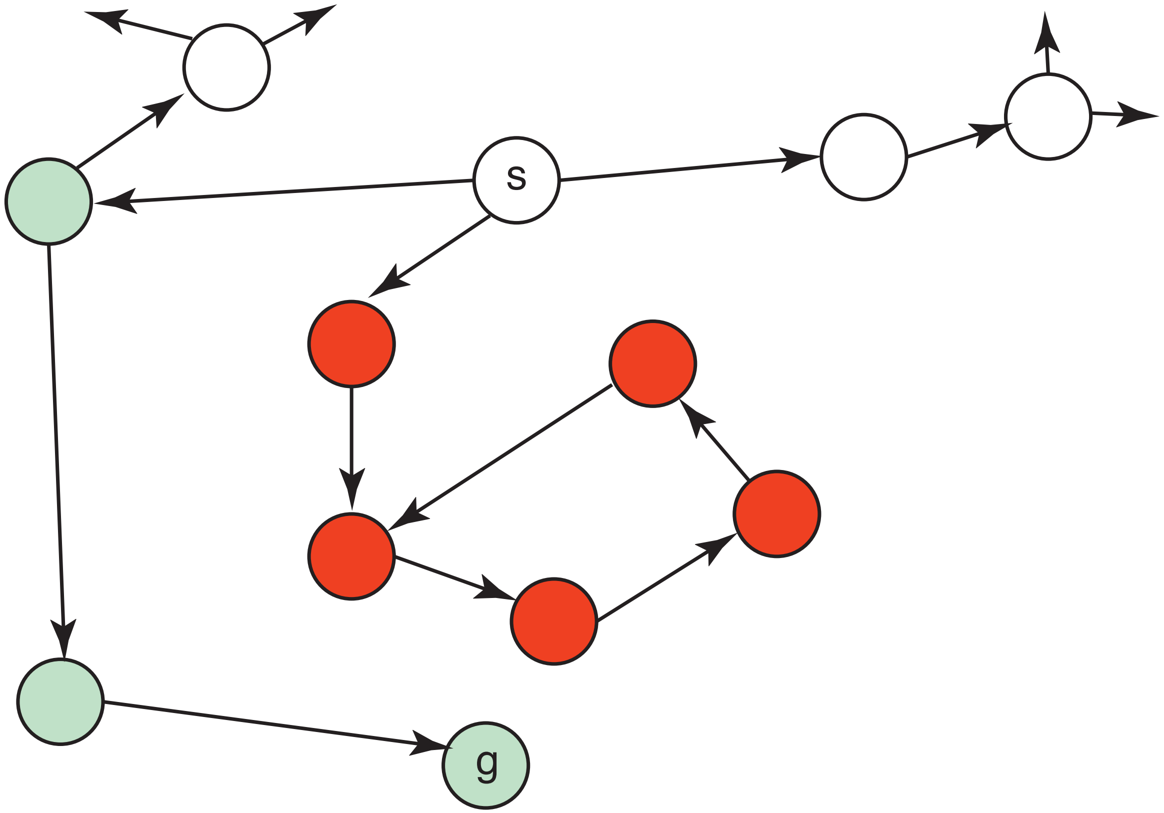


The h function - the heuristic value of a path equal to the heuristic value of the node at the end of the path. $h(\langle n_o, \ldots, n_k \rangle) = h(n_k)$

The graph is drawn to scale, where the cost of an arc is its length.
The aim is to find the shortest path from s to g.

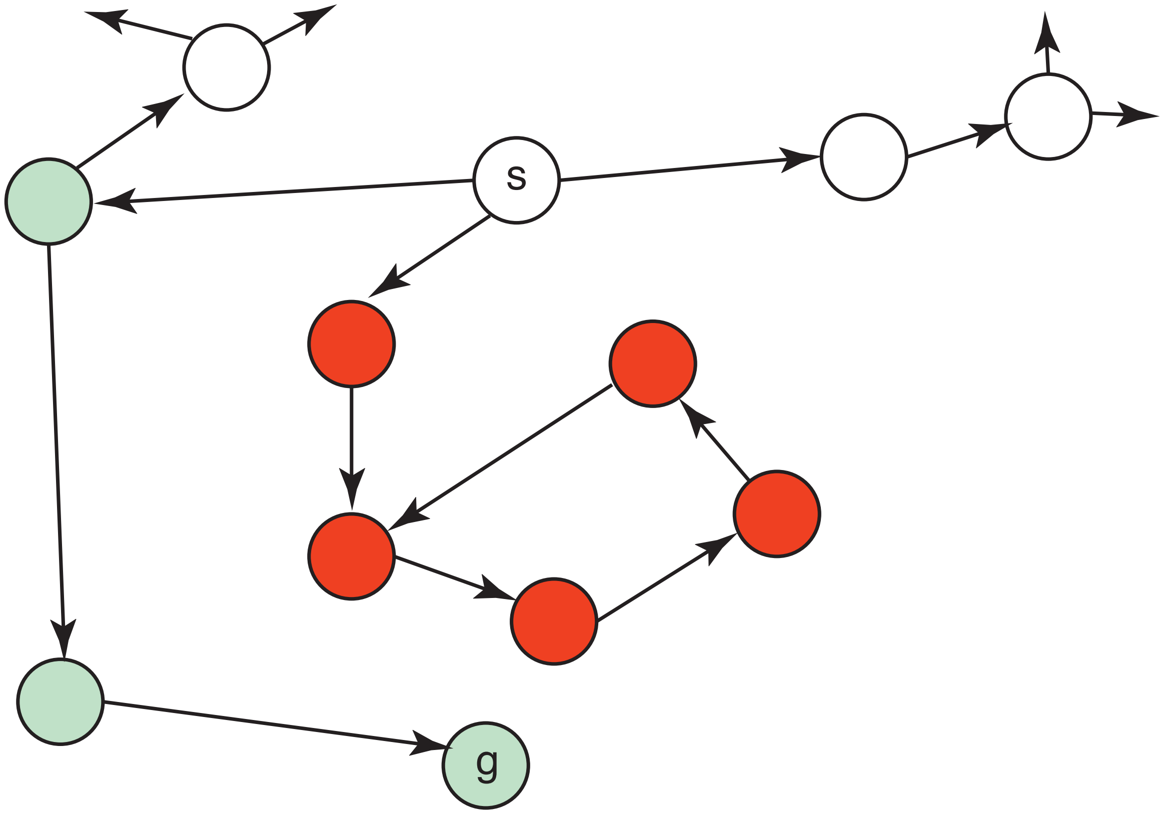
Illustrative Graph — Best-first Search

Suppose the Euclidean straight-line distance to the goal g is used as the heuristic function.



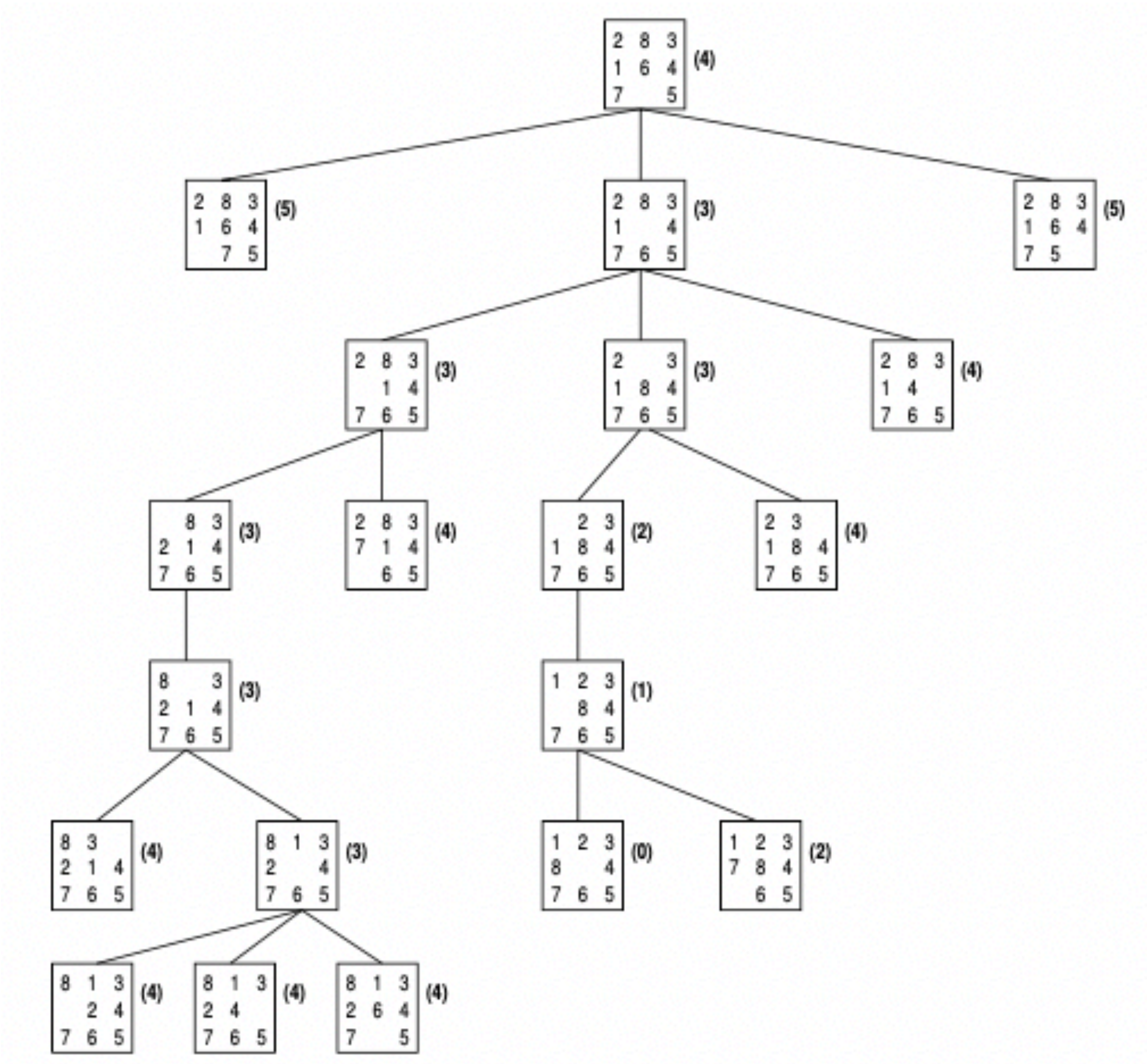
Illustrative Graph — Best-first Search

Suppose the Euclidean straight-line distance to the goal g is used as the heuristic function.



A heuristic depth-first search will select the node below s and will never terminate. Similarly, because all of the nodes below s look good, a greedy best-first search will cycle between them, never trying an alternate route from s.

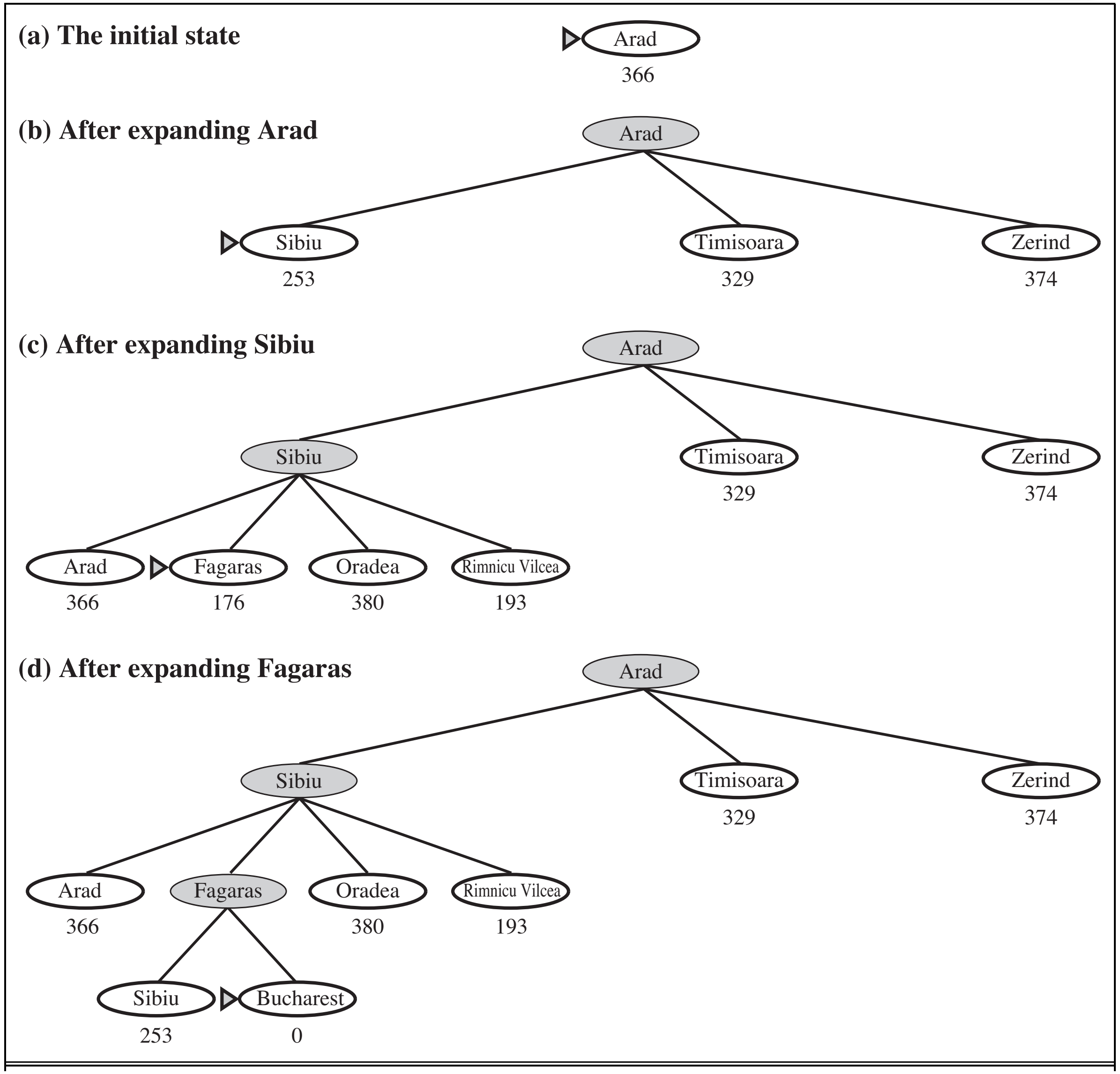
Examples of Greedy Best-First Search



Examples of Greedy Best-First Search

Implementation:

Order the nodes in decreasing order of desirability



Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD} .

Nodes are labeled with their h-values.

Properties of Greedy Best-First Search

- Complete: No! can get stuck in loops, e.g.,
  Complete in finite space with repeated-state checking

- Time: $O(b^m)$, where m is the maximum depth in search space.

- Space: $O(b^m)$ (retains all nodes in memory)

- Optimal: No!

Therefore Greedy Search has the same deficits as Depth-First Search.
However, a good heuristic can reduce time and memory costs substantially.

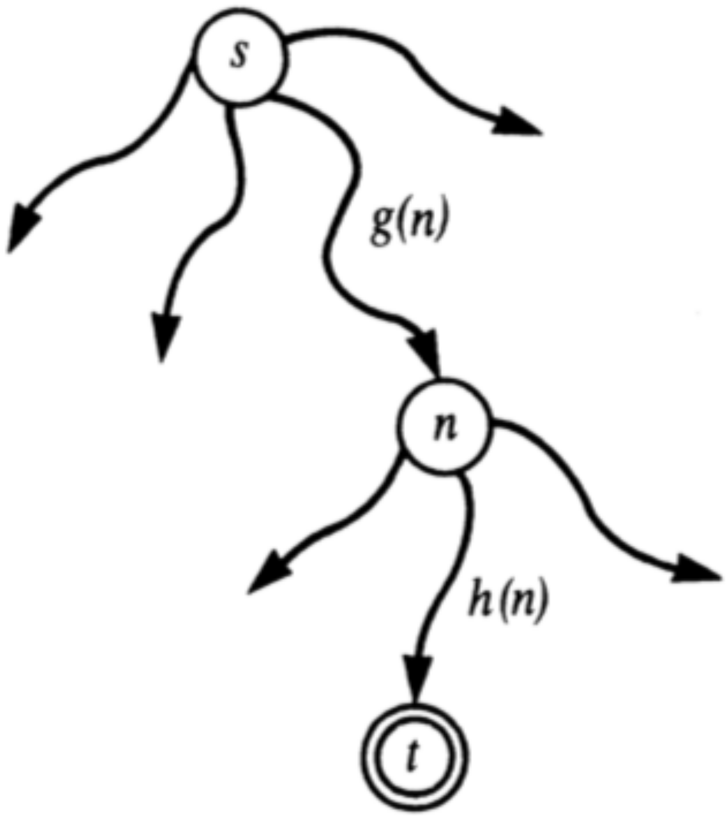
A* Search

- Idea: Use both cost of path generated and estimate to goal to order nodes on the frontier
- $g(n)$ = cost of path from start to n; $h(n)$ = estimate from n to goal
- Order priority queue using function $f(n) = g(n) + h(n)$
- $f(n)$ is the estimated cost of the cheapest solution extending this path

A* Search

- A* Search uses evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost from initial node to node n
  - $h(n)$ = estimated cost of cheapest path from n to goal
  - $f(n)$=estimated total cost of cheapest solution through node n
- Combines uniform-cost search and greedy search
- Greedy Search minimizes $h(n)$
  - efficient but not optimal or complete
- Uniform Cost Search minimizes $g(n)$
  - optimal and complete but not efficient

Heuristic function

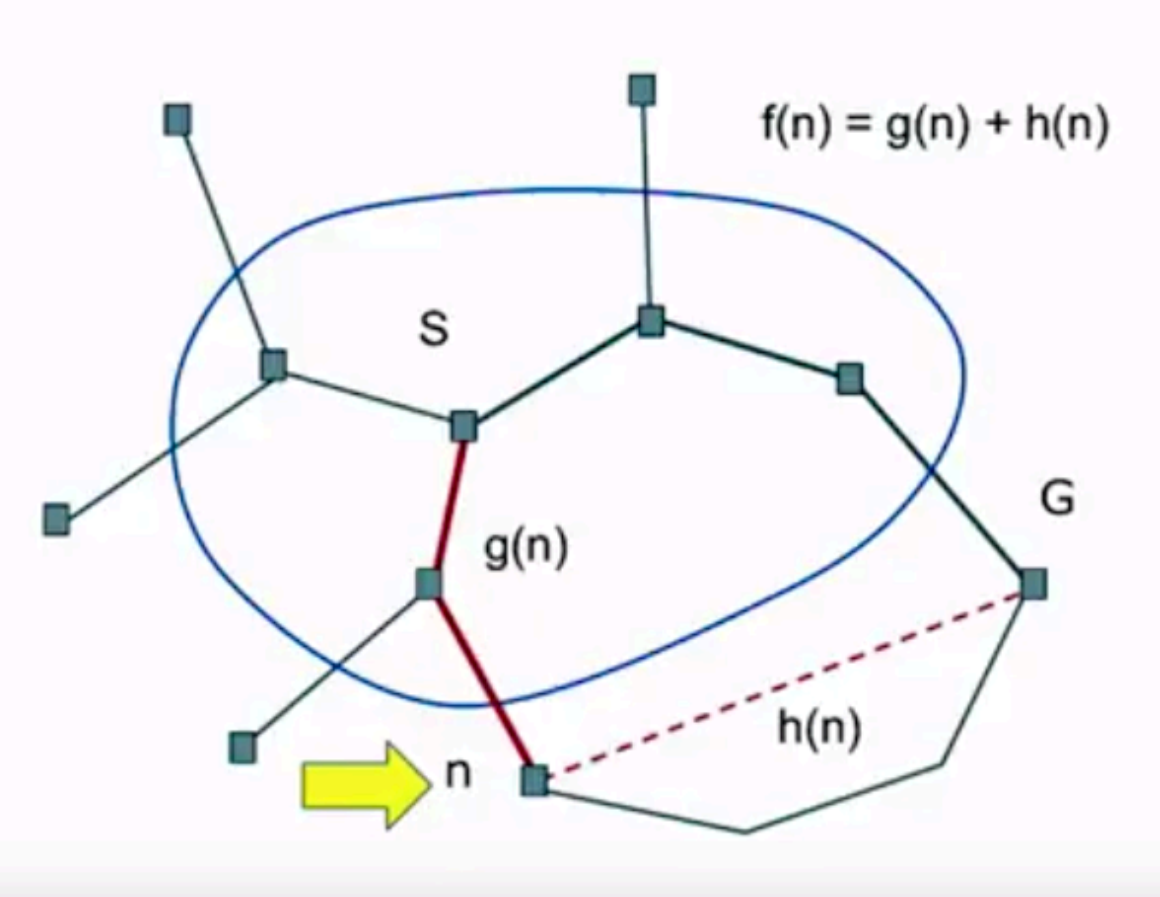


Heuristic estimate *f(n) of* the cost of the cheapest paths from ***s*** to ***t*** via ***n****:* $f(n) = g(n) + h(n)$

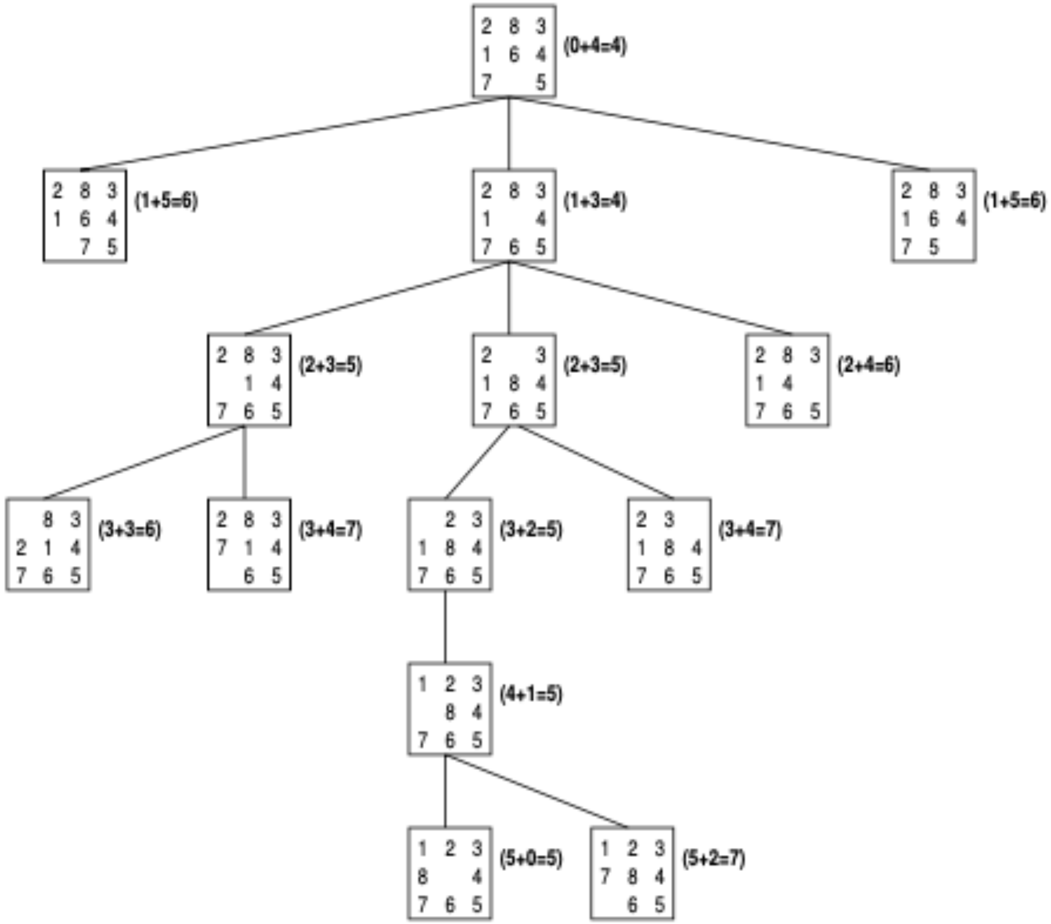
g(n) is an estimate of the cost of an optimal path from s to n

h(n) is an estimate of the cost of an optimal path from n to t.

A* Search

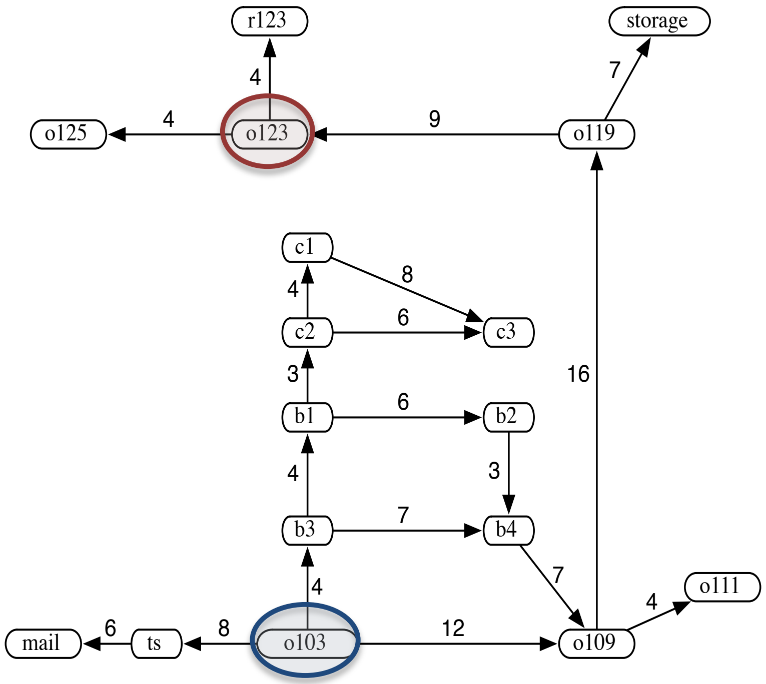


A∗ Search



Delivery Robot - Heuristic Function

$$
\begin{array}{rclrclrcl}
h(mail) & = & 26 & h(ts) & = & 23 & h(o103) & = & 21 \\
h(o109) & = & 24 & h(o111) & = & 27 & h(o119) & = & 11 \\
h(o123) & = & 4 & h(o125) & = & 6 & h(r123) & = & 0 \\
h(b1) & = & 13 & h(b2) & = & 15 & h(b3) & = & 17 \\
h(b4) & = & 18 & h(c1) & = & 6 & h(c2) & = & 10 \\
h(c3) & = & 12 & h(storage) & = & 12 & & &
\end{array}
$$



A∗ Search - the Delivery Robot

$[o103_{21}]$ $[b3_{21}, ts31, o10935]$

$$f(\langle o103, b3\rangle) = cost(\langle o103, b3\rangle) + h(b3) = 4 + 17 = 21.$$

b3	$[b3_{21}, b4_{29}, ts_{31}, o109_{36}]$
b1	$[c2_{21}, b2_{29}, b4_{29}, ts_{31}, o109_{36}]$
c2	$[c2_{21}, b2_{29}, b4_{29}, c3_{29}, ts_{31}, o109_{36}]$
c1	$[b2_{29}, b4_{29}, c3_{29}, ts_{31}, c3_{35}, o109_{36}]$
c3	$[b2_{29}, b4_{29}, c3_{29}, o109_{36}]$
b2	$[b4_{29}, ts_{31}, c3_{35}, o109_{36}]$
b4	$[ts_{31}, c3_{35}, b4_{35}, o109_{36}, o109_{42}]$

$$\begin{array}{rclrclrcl}
h(mail) &=& 26 & h(ts) &=& 23 & h(o103) &=& 21 \\
h(o109) &=& 24 & h(o111) &=& 27 & h(o119) &=& 11 \\
h(o123) &=& 4 & h(o125) &=& 6 & h(r123) &=& 0 \\
h(b1) &=& 13 & h(b2) &=& 15 & h(b3) &=& 17 \\
h(b4) &=& 18 & h(c1) &=& 6 & h(c2) &=& 10 \\
h(c3) &=& 12 & h(storage) &=& 12 & & &
\end{array}$$

A∗ Search - the Delivery Robot

[o103$_{21}$] [b3$_{21}$, ts31, o10935]

$$f(\langle o103, b3\rangle) = cost(\langle o103, b3\rangle) + h(b3) = 4 + 17 = 21.$$

b3	[b3$_{21}$, b4$_{29}$, ts$_{31}$, o109$_{36}$]
b1	[c2$_{21}$, b2$_{29}$, b4$_{29}$, ts$_{31}$, o109$_{36}$]
c2	[c2$_{21}$, b2$_{29}$, b4$_{29}$,c3$_{29}$, ts$_{31}$, o109$_{36}$]
c1	[b2$_{29}$, b4$_{29}$, c3$_{29}$, ts$_{31}$, c3$_{35}$, o109$_{36}$]
c3	[b2$_{29}$, b4$_{29}$, c3$_{29}$, o109$_{36}$]
b2	[b4$_{29}$, ts$_{31}$, c3$_{35}$, o109$_{36}$]
b4	[ts$_{31}$, c3$_{35}$, b4$_{35}$, o109$_{36,}$ o109$_{42}$]

$$\begin{array}{rclrclrcl}
h(mail) & = & 26 & h(ts) & = & 23 & h(o103) & = & 21 \\
h(o109) & = & 24 & h(o111) & = & 27 & h(o119) & = & 11 \\
h(o123) & = & 4 & h(o125) & = & 6 & h(r123) & = & 0 \\
h(b1) & = & 13 & h(b2) & = & 15 & h(b3) & = & 17 \\
h(b4) & = & 18 & h(c1) & = & 6 & h(c2) & = & 10 \\
h(c3) & = & 12 & h(storage) & = & 12 & & &
\end{array}$$

A lowest-cost path to the goal is eventually found. The algorithm is forced to try many different paths, because several of them temporarily seemed to have the lowest cost. It still does better than either lowest-cost-first search or greedy best-first search.

A* Search

- A* Search minimizes $f(n) = g(n) + h(n)$
  - idea: preserve efficiency of Greedy Search but avoid expanding paths that are already expensive

- Q: is A* Search optimal and complete ?

A* Search

- A* Search minimizes $f(n) = g(n) + h(n)$
  - idea: preserve efficiency of Greedy Search but avoid expanding paths that are already expensive

Q: is A* Search optimal and complete ?

A: Yes! provided $h()$ is admissible in the sense that it never overestimates the cost to reach the goal.

A∗ Search — Analysis

Conditions for optimality: Admissibility and consistency

- The first condition we require for optimality is that $h(n)$ be an admissible heuristic.
- An admissible heuristic is one that *never overestimates* the cost to reach the goal.
  - Because $g(n)$ is the actual cost to reach n along the current path, and $f(n) = g(n) + h(n)$, we have as an immediate consequence that $f(n)$ never overestimates the true cost of a solution along the current path through n.
- Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

A* Search – conditions for optimality

- Heuristic $h()$ is called admissible if
  $\forall n\ h(n) \leq h^*(n)$ where $h^*(n)$ is true cost from n to goal

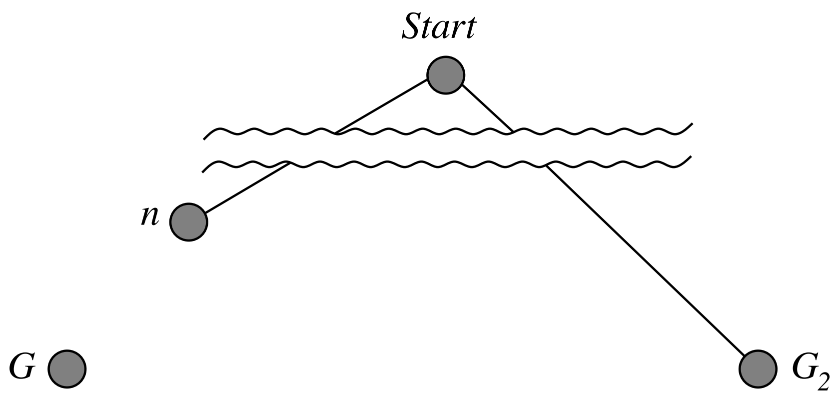
- If h is admissible then $f(n)$ never overestimates the actual cost of the best solution through n.

- Example: $h_{SLD}()$ is admissible because the shortest path between any two points is a line.

- Theorem: A* Search is optimal if $h()$ is admissible.

Optimality of A* Search

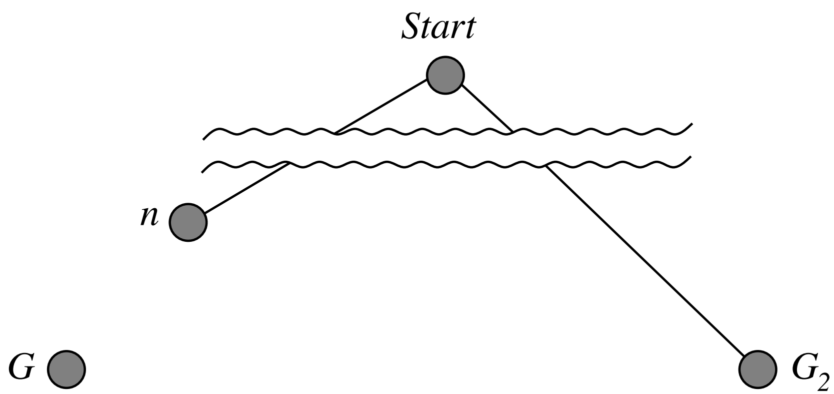
Suppose a suboptimal goal node *G*2 has been generated and is in the queue. Let *n* be the last unexpanded node on a shortest path to an optimal goal node *G*.



$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible.} \end{aligned}$$

Optimality of A* Search

Suppose a suboptimal goal node *G*2 has been generated and is in the queue. Let *n* be the last unexpanded node on a shortest path to an optimal goal node *G*.



$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &> g(G) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible.} \end{aligned}$$

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

Optimality of A* Search

- Since $f(G2) > f(n)$, A^* will never select $G2$ for expansion.
- Note: suboptimal goal node $G2$ may be generated, but it will never be expanded.
- In other words, even after a goal node has been generated, A^* will keep searching so long as there is a possibility of finding a shorter solution.
- Once a goal node is selected for expansion, we know it must be optimal, so we can terminate the search.

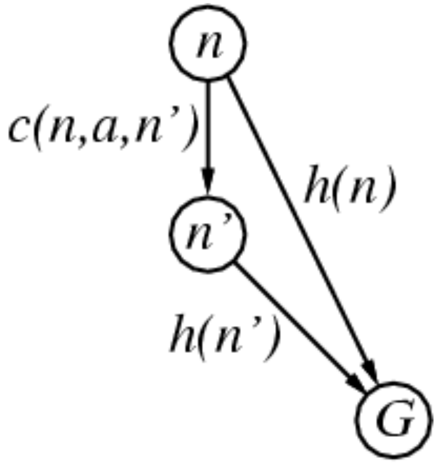
Consistent heuristics

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

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

  If *h* is consistent, we have

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



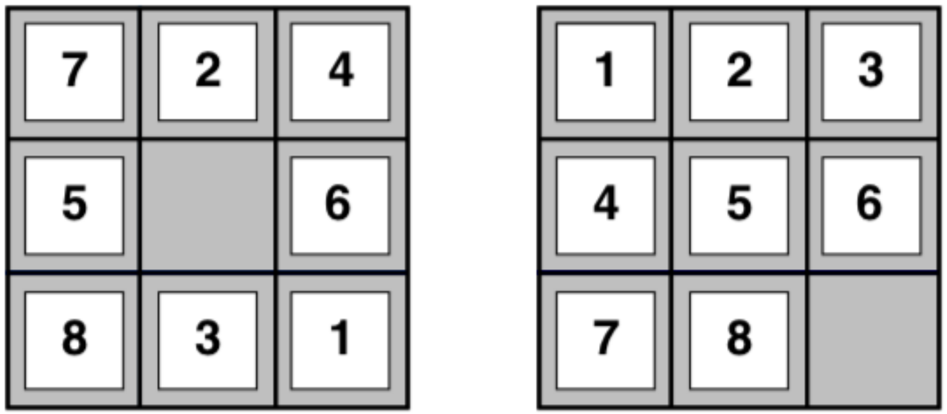
Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h1(n)$ = total number of misplaced tiles

$h2(n)$ = total Manhattan distance = ∑distance from goal position



Start State Goal State

$h1(S)$ = ?

$h2(S)$ = ?

- Why are $h1$, $h2$ admissible?

Optimality of A* Search

- **Complete:** Yes, unless there are infinitely many nodes with $f \leq$ cost of solution
- **Time:** Exponential in [relative error in $h \times$ length of solution]
- **Space:** Keeps all nodes is memory
- **Optimal:** Yes (assuming $h()$ is admissible).

Iterative Deepening A* Search

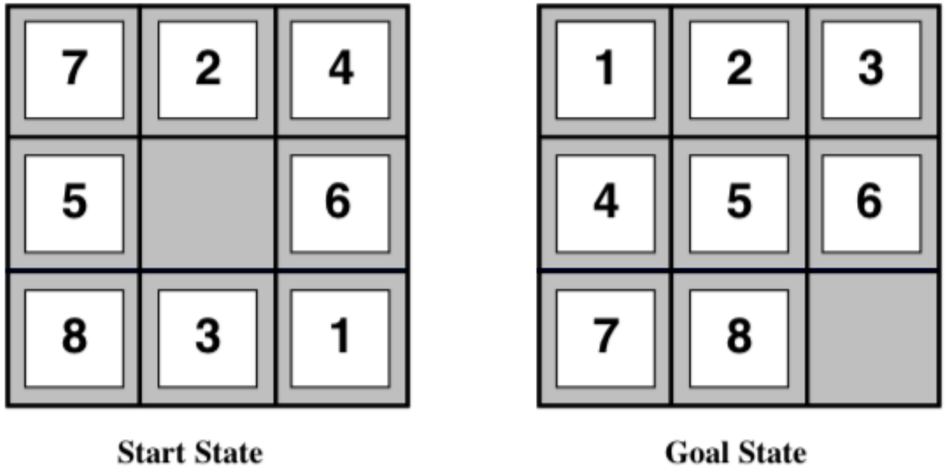
- Iterative Deepening A* is a low-memory variant of A* which performs a series of depth-first searches but cuts off each search when the sum $f() = g() + h()$ exceeds some pre-defined threshold.
- The threshold is steadily increased with each successive search.
- IDA* is asymptotically as efficient as A* for domains where the number of states grows exponentially.

Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h1(n)$ = total number of misplaced tiles

$h2(n)$ = total Manhattan distance = ∑distance from goal position



$h1(S)$ = ?

$h2(S)$ = ?

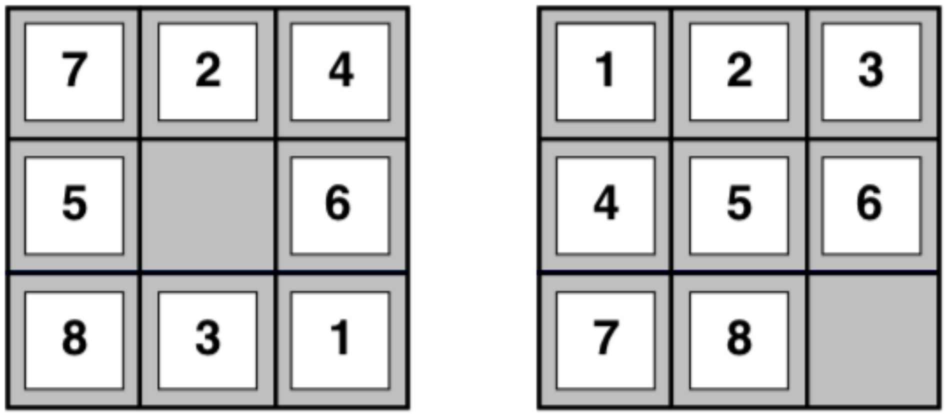
- Why are $h1$, $h2$ admissible?

Examples of Admissible Heuristics

e.g. for the 8-puzzle:

$h1(n)$ = total number of misplaced tiles

$h2(n)$ = total Manhattan distance = ∑distance from goal position



Start State Goal State

$h1(S) = 6$

$h2(S) = 4+0+3+3+1+0+2+1 = 14$

- $h1$: every tile must be moved at least once.
- $h2$: each action can only move one tile one step closer to the goal.

How to Find Heuristic Functions ?

- Admissible heuristics can often be derived from the exact solution cost of a simplified or “relaxed” version of the problem. (i.e. with some of the constraints weakened or removed)
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h1(n)$ gives the shortest solution.
  - If the rules are relaxed so that a tile can move to any adjacent square, then $h2(n)$ gives the shortest solution.

Dominance

- if $h2(n) \geq h1(n)$ for all n (both admissible) then $h2$ dominates $h1$ and is better for search. So the aim is to make the heuristic $h()$ as large as possible, but without exceeding $h^*()$.
- typical search costs:

14-puzzle	IDS	= 3,473,941 nodes
	$A^*(h_1)$	= 539 nodes
	$A^*(h_2)$	= 113 nodes
24-puzzle	IDS	$\approx 54 \times 10^9$ nodes
	$A^*(h_1)$	= 39,135 nodes
	$A^*(h_2)$	= 1,641 nodes

Composite Heuristic Functions

- Let $h_1, h_2, \ldots, h_m$ be admissible heuristics for a given task.
- Define the composite heuristic

$$h(n) = \max(h_1(n), h_2(n), \ldots, h_m(n))$$

- h is admissible
- h dominates $h_1, h_2, \ldots, h_m$

Summary of Informed Search

- Heuristics can be applied to reduce search cost.
- Greedy Search tries to minimize cost from current node *n* to the goal.
- A^* combines the advantages of Uniform-Cost Search and Greedy Search
- A^* is complete, optimal and optimally efficient among all optimal search algorithms.
- Memory usage is still a concern for A^*. A^* is a low-memory variant.

Summary

- Informed search makes use of problem-specific knowledge to guide progress of search
- This can lead to a significant improvement in performance
- Much research has gone into admissible heuristics
  - Even on the automatic generation of admissible heuristics