Best-First Search

- Idea: select the path whose end is closest to a goal according to the heuristic function.

- **Best-First search** selects a path on the frontier with minimal *h*-value (for the end node).

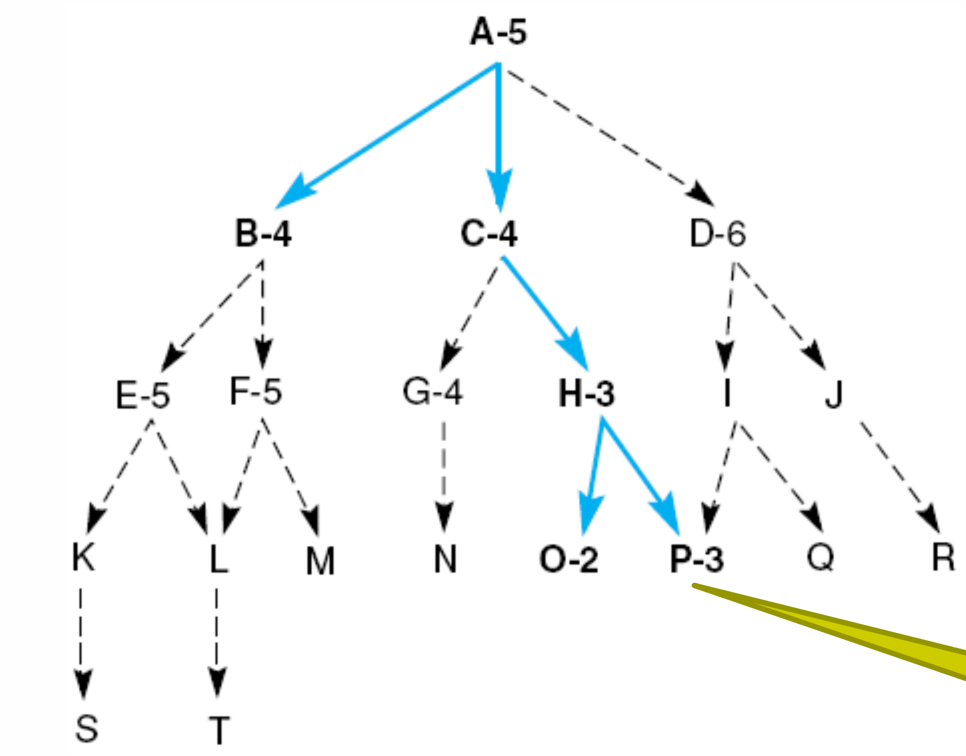
- **Necessary data structures**

  Open (to be visited) queue as a **priority queue** (max or min heap) and **Closed** queue (already visited) are maintained and **avoid** duplicate states. Optionally **SL** (to maintain the current path) for the shortest path

Best-First Search Algorithm

- **Algorithm sketch for Best-First Search**
  1. Check if the current state is a solution, if so return it else go to next step.
  2. Expand the current state of the search.
  3. Evaluate its children states.
  4. **Select** the **most promising** state (from **all states seen**) in **Open** priority queue.
  5. **If** a path to the current state is **NOT shortest**, **backtrack** to a previous state.
     - By simply selecting the **next best state** from Open priority queue
  - **Continue** steps 1 – 5 **until** it finds a **solution**.
- Comment on Best-First Search
  - works like a generic least cost search taking only good strategies used in DFS, BFS, Hill climbing, Backtracking algorithm that utilizes heuristics.
- What's the **main difference** between Backtracking and Best-First Search?

Best-First Search of a Hypothetical State Space



Letters represent states

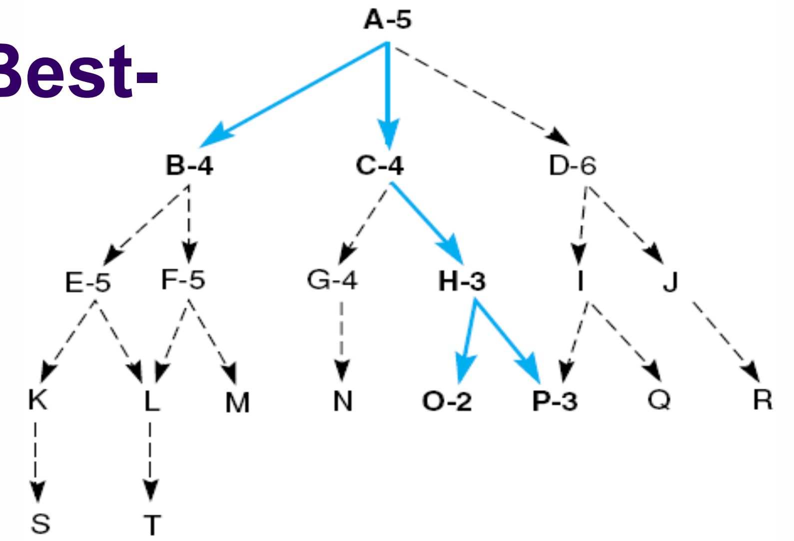
Numbers represent heuristic values as ***cost***, e.g., *the smaller the better* in this example.

Bold states indicate the states expanded by the heuristic algorithm

Assume P is the goal

Compare this search with hill-climbing and backtracking

A Trace of the Execution of Best-First Search



*Maintain a priority queue by heuristic value

1. **open = [A5]; closed = [ ]** How can we implement a priority queue?
2. **evaluate A5; open = [B4,C4,D6]; closed = [A5]**
3. **evaluate B4; open = [C4,E5,F5,D6]; closed = [B4,A5]**
4. **evaluate C4; open = [H3,G4,E5,F5,D6]; closed = [C4,B4,A5]**
5. **evaluate H3; open = [O2,P3,G4,E5,F5,D6]; closed = [H3,C4,B4,A5]**
6. **evaluate O2; open = [P3,G4,E5,F5,D6]; closed = [O2,H3,C4,B4,A5]**
7. **evaluate P3; the solution is found!**

How can we maintain the shortest path?

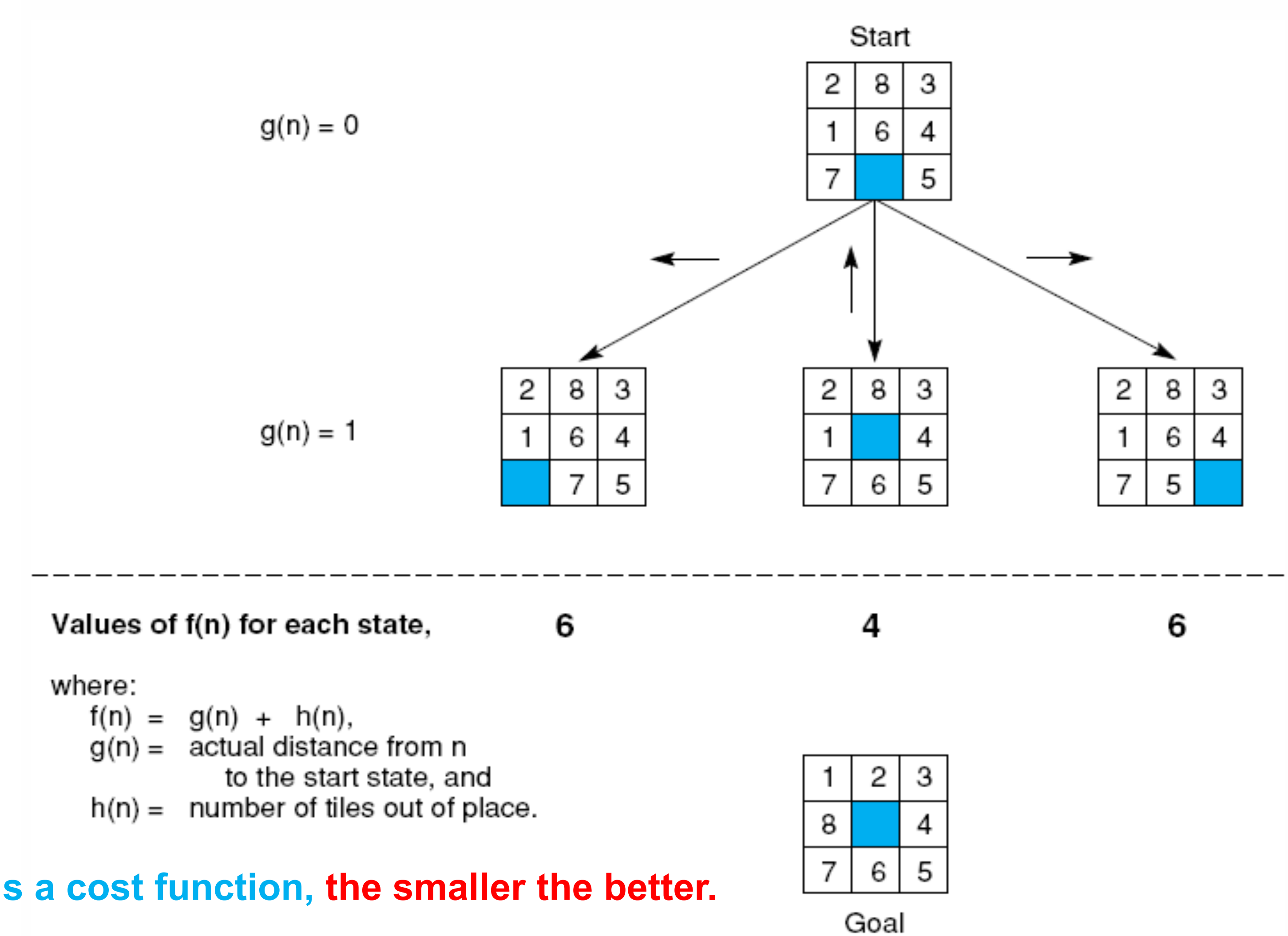
Best-First Search Algorithm

```
function best_first_search;

begin
   open := [Start];                                                    % initialize
   closed := [ ];
   while open ≠ [ ] do                                                 % states remain
      begin
         remove the leftmost state from open, call it X;
         if X = goal then return the path from Start to X
         else begin
                  generate children of X;
                  for each child of X do
                  case
                        the child is not on open or closed:
                           begin
                                 assign the child a heuristic value;
                                 add the child to open
                           end;
                        the child is already on open:
                           if the child was reached by a shorter path
                           then give the state on open the shorter path
                        the child is already on closed:
                           if the child was reached by a shorter path then
                              begin
                                 remove the state from closed;
                                 add the child to open
                              end;
                  end;                                                 % case
                  put X on closed;
                  re-order states on open by heuristic merit (best leftmost)
               end;
return FAIL                                                            % open is empty
end.
```

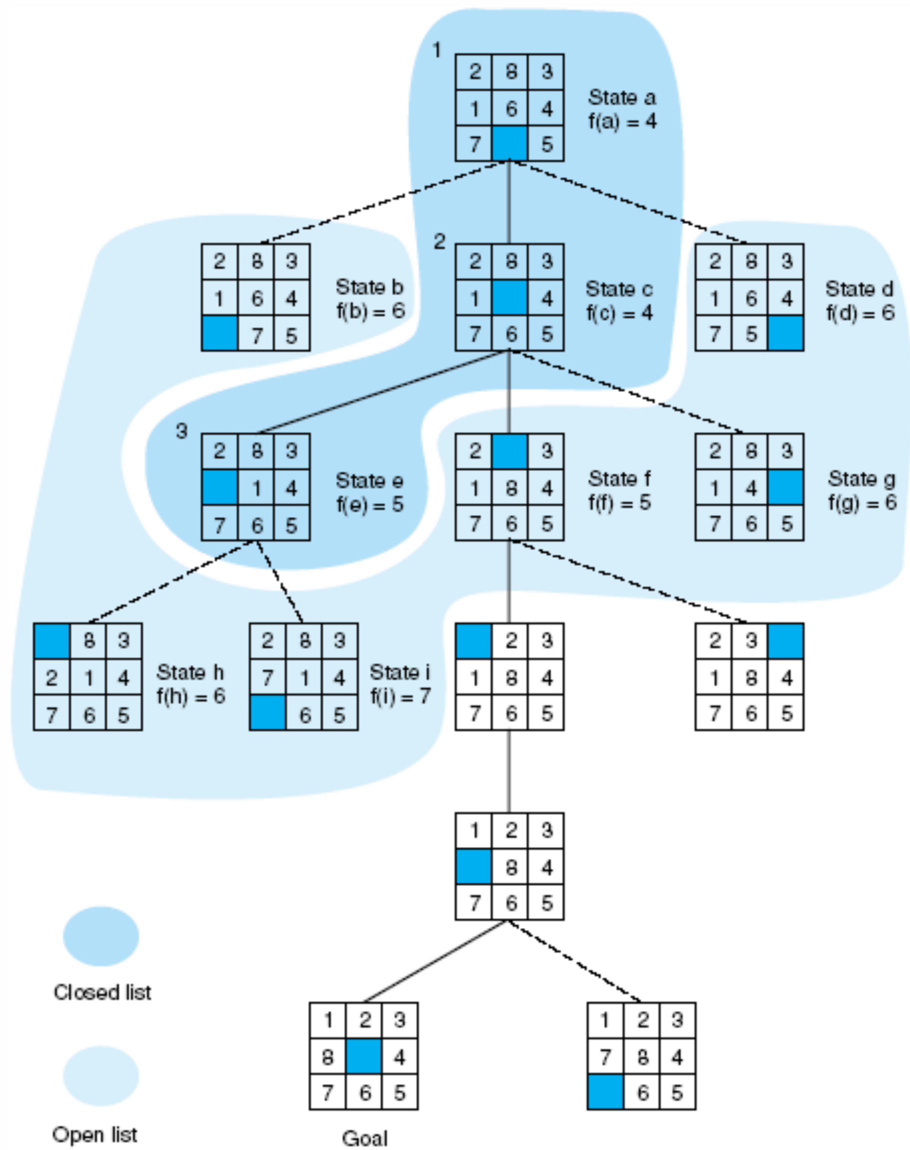
If path is important, we have to maintain the shortest path.
See the backtracking algorithm that maintains the current path.

Heuristic *f(n)* Applied to States in the 8-puzzle for Best-First Search

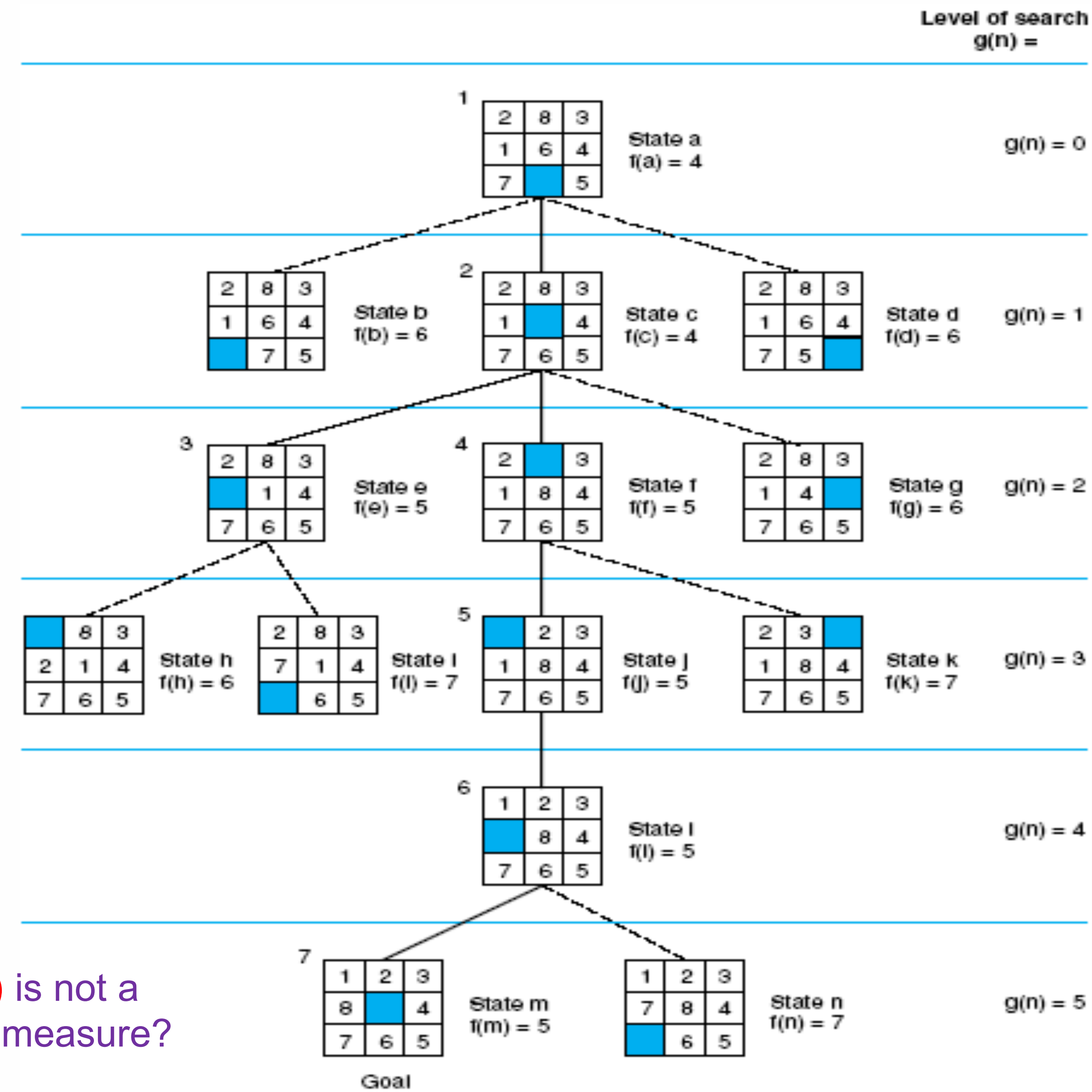


f(n) is a cost function, the smaller the better.

Open and Closed as they Appear after the 3rd Iteration of Best-first Search



State Space Generated during Best-first Search



What if **h(n)** is not a reasonable measure?

Admissibility and Optimal Solution

DEFINITION

ALGORITHM A, ADMISSIBILITY, ALGORITHM A*

Consider the evaluation function f(n) = g(n) + h(n), where

n is any state encountered in the search.

g(n) is the cost of n from the start state.

h(n) is the heuristic estimate of the cost of going from n to a goal.

If this evaluation function is used with the best_first_search algorithm the result is called *algorithm A*.

A search algorithm is *admissible* if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists.

If algorithm A is used with an evaluation function in which h(n) is less than or equal to the cost of the minimal path from n to the goal, the resulting search algorithm is called *algorithm* A* (pronounced "A STAR").

It is now possible to state a property of A* algorithms:

All A* algorithms are admissible.

Admissible heuristic f(n) = g(n) + h*(n) where h*(n) never overestimates the actual cost to reach the goal **AND** h(n)>=0 **AND** h(goal)=0.

Q1: Does this mean **h*()** has to be perfect?
Q2: Are those heuristic functions h1, h2, and h3 admissible?
Q3: Is BFS A*?

h2: Manhattan distance

A^* Search Algorithm

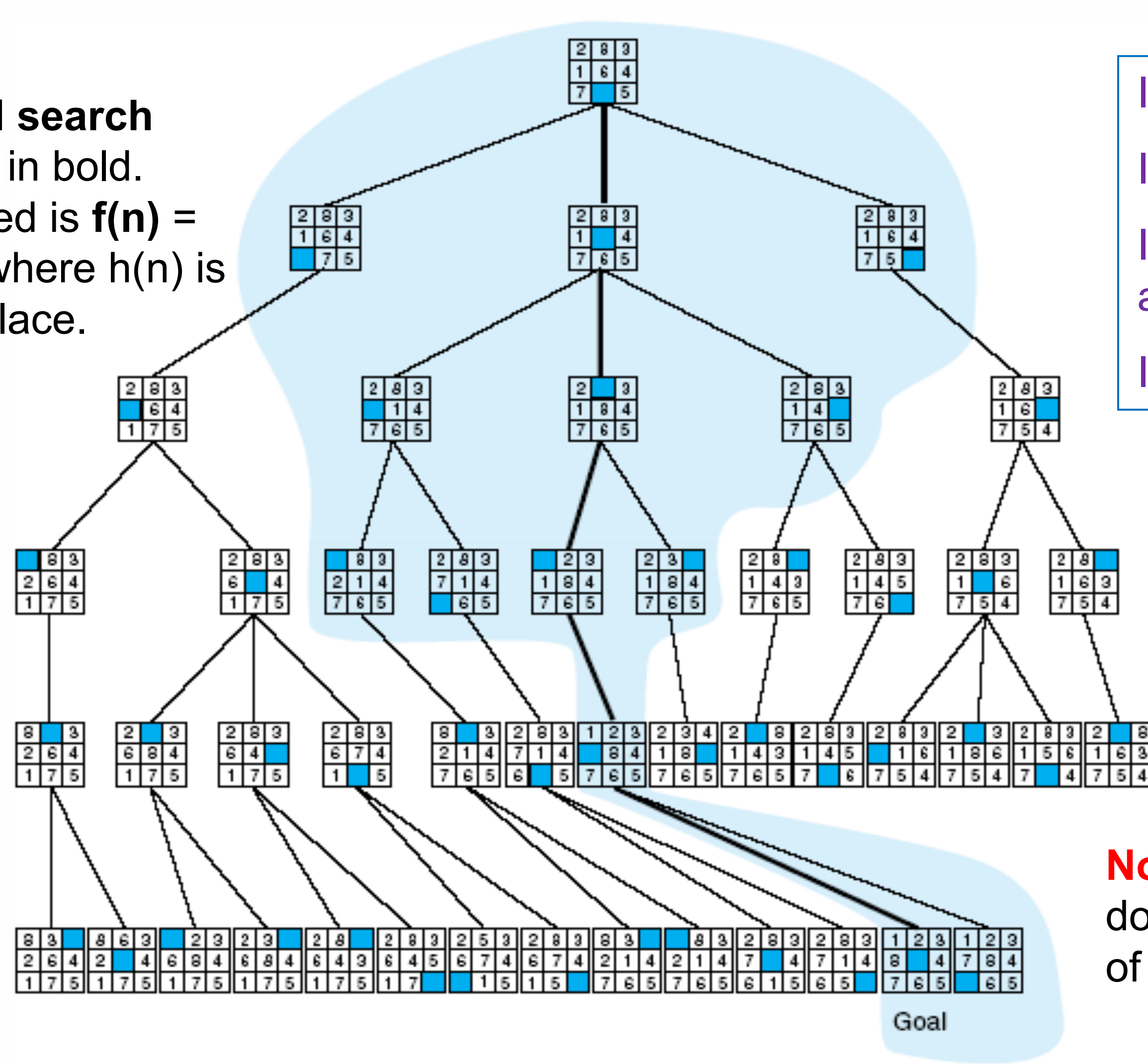
- A^* is a mix of:
  - **lowest-cost-first** and
  - **best-first search**

- A^* treats the frontier as a priority queue ordered by $f(p) = cost(p) + h(p)$

- It always selects the node on the frontier with lowest estimated total distance.

Comparison of State Space Searched using Breadth-First Search and A* Search

The **optimal search selection** is in bold. Heuristic used is **f(n)** = g(n) + h(n) where h(n) is tiles out of place.



Is BFS admissible?

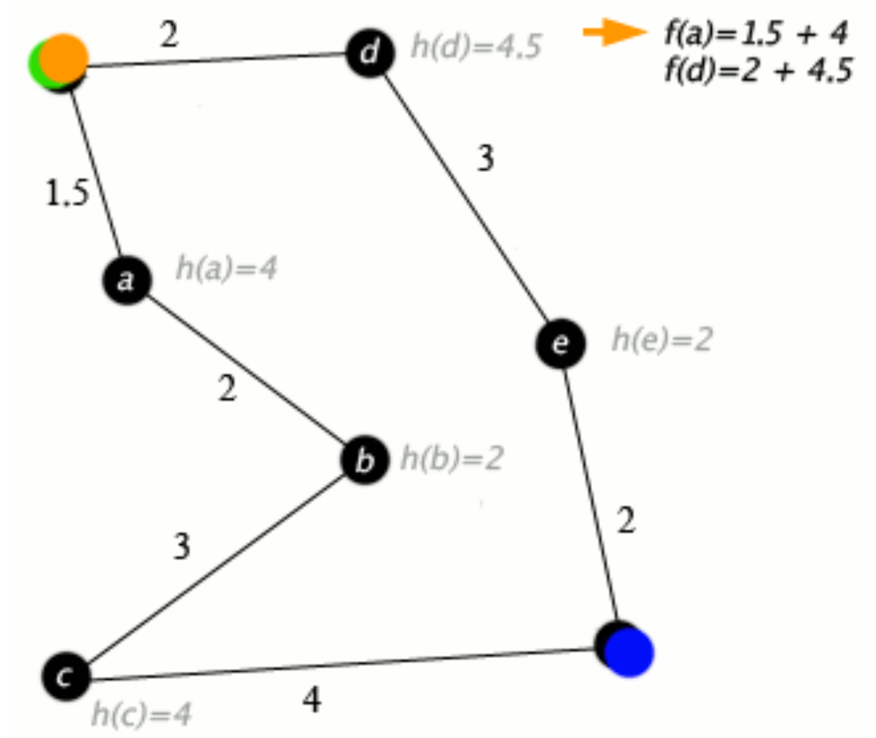
Is DFS admissible?

Is Hill-climbing admissible?

Is SA admissible?

Note: Admissibility doesn't require efficiency of a search method.

Example: cities



An example of an A* algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point

Example: robot motion planning

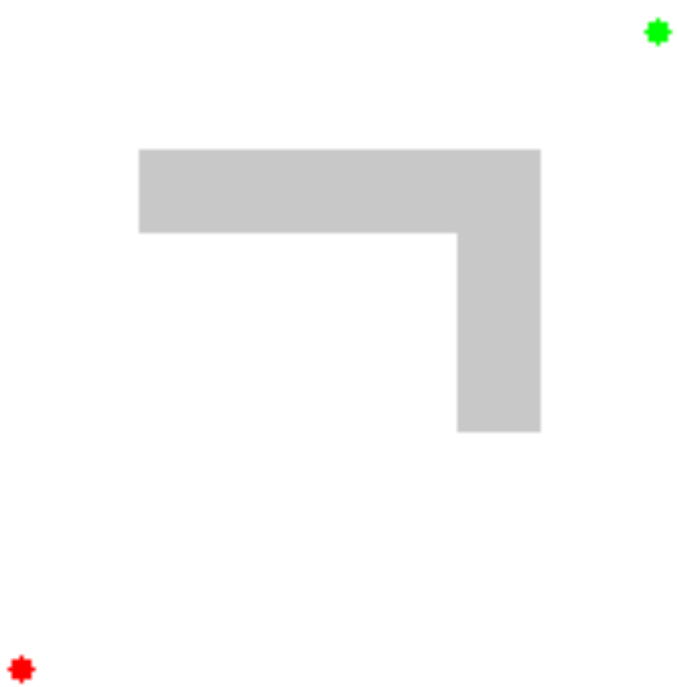
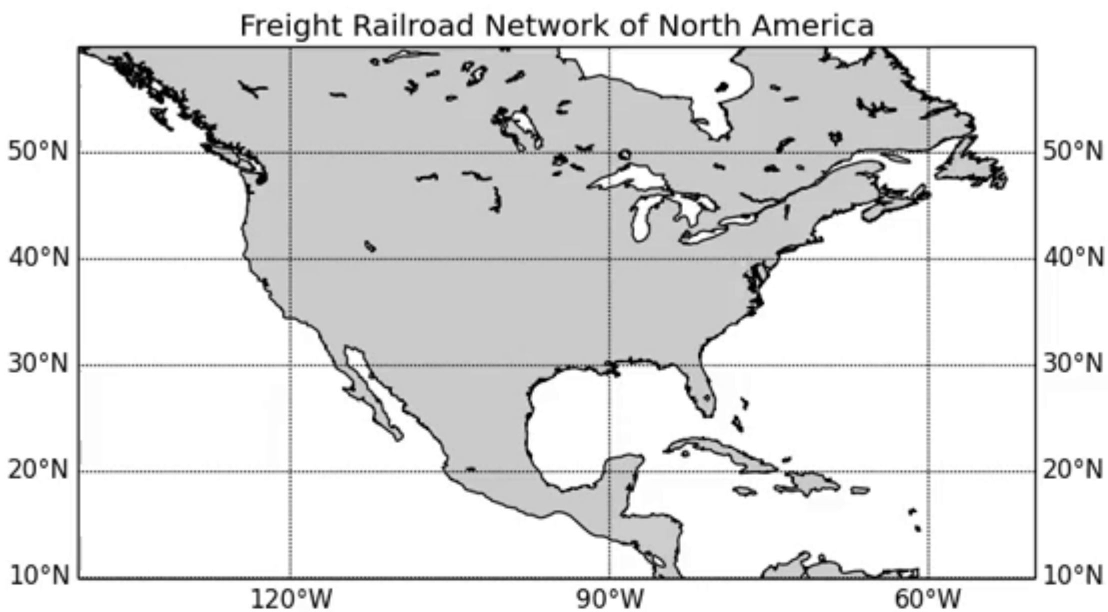


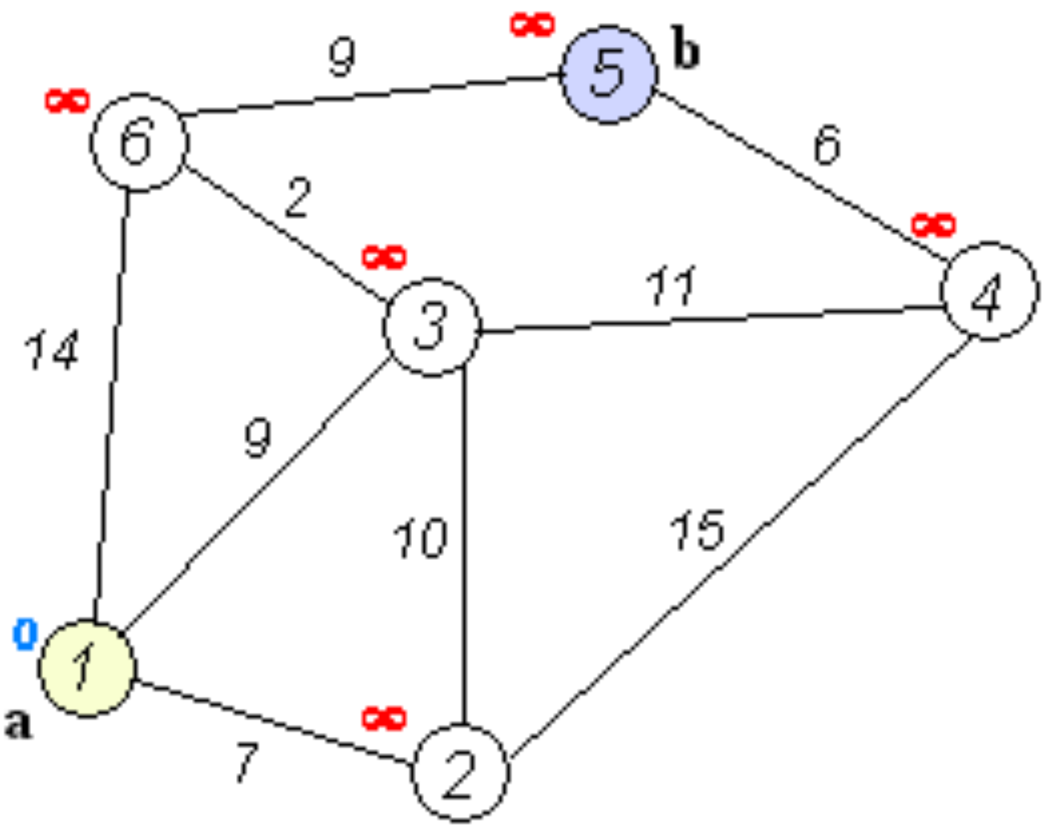
Illustration of A* search for finding path from a start node to a goal node in a robot motion planning problem. **The empty circles represent the nodes in the *open set***, i.e., those that remain to be explored, and the filled ones are in the closed set. **Color on each closed node indicates the distance from the start**: the greener, the farther. One can first see the A* moving in a straight line in the direction of the goal, then when hitting the obstacle, it explores alternative routes through the nodes from the open set.

Example: path between Washington, D.C. and Los Angeles



The A* algorithm also has real-world applications. In this example, edges are railroads and h(x) is the **great-circle distance (the shortest possible distance on a sphere**) to the target. The algorithm is searching for a path between Washington, D.C. and Los Angeles.

Dijkstra's algorithm is a special case of A*



Dijkstra's algorithm, is also called **A1** algorithm, to find the shortest path between *a* and *b*. It picks the unvisited vertex with the **lowest distance**, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

$h(n) = 0$

Local Admissibility of Heuristics

Question: Is there heuristics that are locally admissible or consistently find the minimal path to each state they encounter in the search (monotonicity)?

DEFINITION

MONOTONICITY **(Consistency)**

A heuristic function h is monotone if

1. For all states n_i and n_j, where n_j is a descendant of n_i,

   $h(n_i) - h(n_j) \le cost(n_i,n_j)$,

   where $cost(n_i,n_j)$ is the actual cost (in number of moves) of going from state n_i to n_j.

2. The heuristic evaluation of the goal state is zero, or h(Goal) = 0.

Monotonicity requires local admissibility of h*() between states n_i to n_j, **not only** h*(n).
Admissible heuristic function: f(n) = g*(n) + h*(n) where g*(n) is the cost of the shortest path from the start to n **AND** h*(n) returns the actual cost of the shortest path from n to the goal (never overestimate the actual path) **AND** h(n) >=0 **AND** h(goal)=0).
f*(n) is monotonically NON-decreasing and the actual cost of the optimal path from start to goal.

Use of Information in Heuristic Search

DEFINITION

INFORMEDNESS

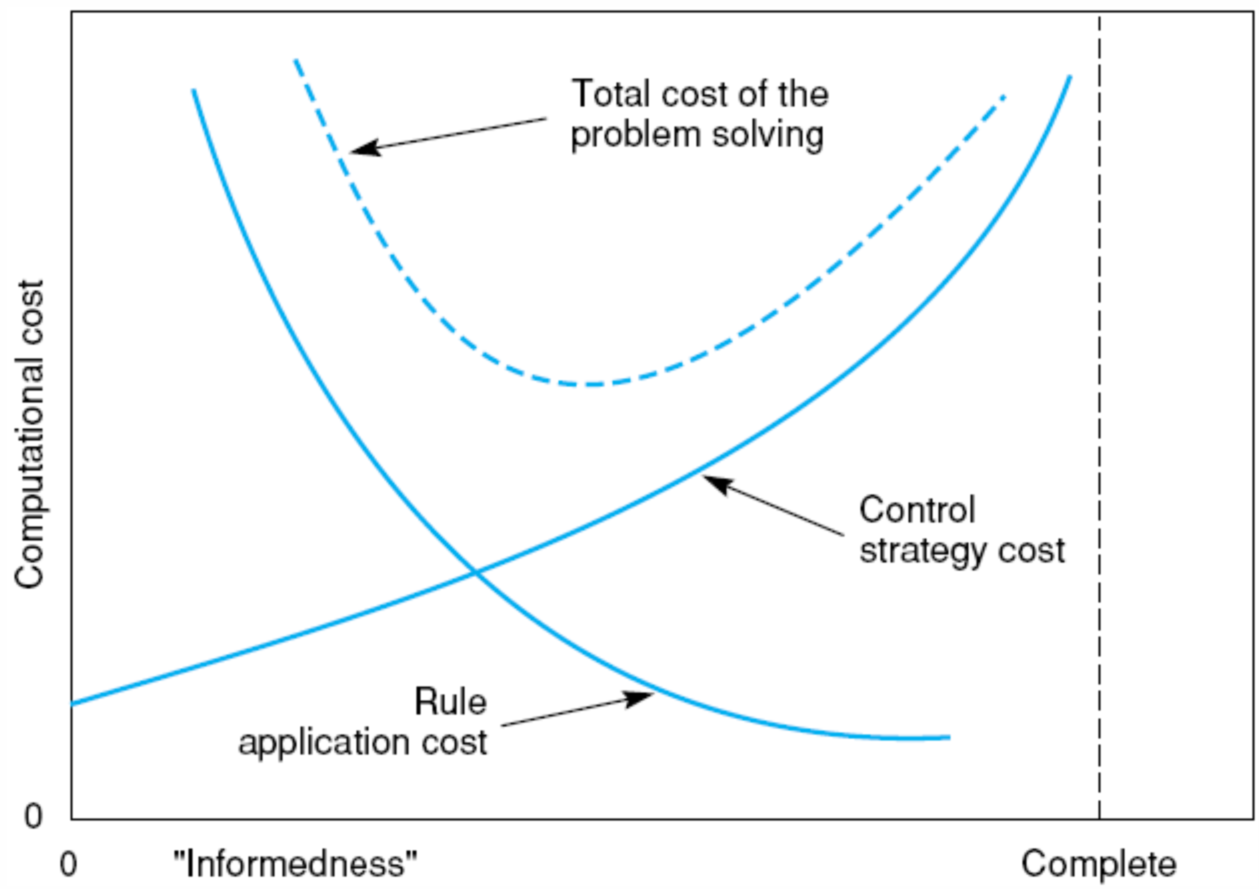
For two A* heuristics h_1 and h_2, if $h_1(n) \le h_2(n)$, for all states n in the search space, heuristic h_2 is said to be *more informed* than h_1.

*In general **the more informed heuristic is better in decision making**.

*But we should consider the computational cost to use the more information, e.g., playing a chess in limited time and resources.

*Using too much of information may not help any longer at certain point.

Computational Cost vs. Informedness



Informal plot of cost of searching and **cost of computing** heuristic evaluation against **informedness** of heuristic, (from Nilsson,1980)

How to Evaluate the Behavior of Heuristic Search

- **Criteria for good heuristic search**
  - Completeness
    - Is the algorithm guaranteed to find a solution when there is one?
  - Optimality
    - Does the strategy find the optimal solution?
  - Time complexity
    - How long does it take to find a solution?
  - Space complexity
    - How much memory is needed to perform the search?
- But all heuristics are fallible since heuristic is an informed guess for the next step to be taken in solving a problem.
  - **Heuristics are still critical** in problem solving since
    - Many problems do not have exact solutions, e.g., medical diagnosis.
    - Problems may have an exact solution but the computational cost of finding it may be prohibitive.

- **To better understand heuristic search, try to solve more example problems** using Hill-climbing and A* algorithm
  - Auto pilot
  - Autonomous drone
  - Robot vacuum
  - Chess
  - Decision making for investment in financial markets
  - Etc.

- Can we solve all types of problems using A*?

Beyond the Types of Heuristic Search Methods We Have Discussed So Far

- **How expensive** backtracking is in problem solving?
- Can we use **backtracking** strategy for **all types** of problems?
- Can a heuristic function be learned from experience (adaptation) and executed during runtime?
- Searching under **observable** or **partially observable environments**?
  - State space search is deterministic.
- How about searching under unknown environments?
  - **Offline search** computes a complete solution before action but **online search** interleaves computation and action, e.g., first take an action then observes the environment and compute the next action, etc.
- How happens if multi-agents (instead of single-agent) are involved in searching for solution through cooperation?
- How can we solve problems involving game or competition?
  - We will talk about this type of problems next.

References

- George Fluger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition, **Chapter 4**, Addison Wesley, 2009.
- Russel and Norvig, Artificial Intelligence: A Modern Approach, 3rd edition, Prentice Hall, 2010.