Artificial Intelligence and Evolutionary Computing

Lecture 3

IT/T/414A
Winter Semester 2014

Instructor: Kartick Ch. MONDAL

Search Process

- Search Algorithm
  - Uninformed Search:
    - Where we have no problem specific reason to prefer one part of the search space to any other as far as finding a path to the goal is concerned.
      - Breadth First Search
      - Depth First Search
      - Uniformed Cost Search
  - Informed Search:
    - We have the information of goal location or how far the goal is.
    - Heuristic Search
      - Best First Search
    - A* Search

Search Process

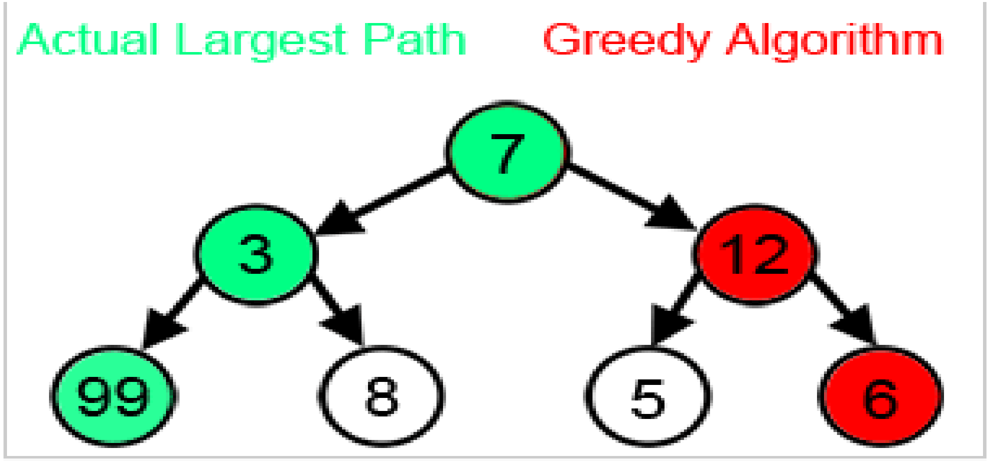
- Heuristic Search
  - Where we have problem specific information to help focus the search
  - Use a function to estimates how close a state is to a goal.
    - Best First Search
- Greedy Search
  - Follows the problem solving heuristic of making the locally optimal choice (UCS) at each stage with the hope of finding a global optimum.
  - Local search: Look at your action, see which one improves the heuristics and then you take it.
  - Greedy Strategy:
    - Expand a node that is closest to a goal state.
    - Heuristic information helps to find the goal.

Best First Search

- Also known as Heuristic Search
- Like breadth first search, it does not proceed uniformly outward from the start node.
- It proceeds through node that is heuristic, means problem specific information indicates might be on the best path to a goal.
- It attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first.

Best First Search

- Main idea is:
  - We have a heuristic or evaluation function to help decide which node is the best one to expand next.
  - Expand the node decided by the first step by resolving ties arbitrarily.
  - Terminate when the node to be expanded next is a goal node.



With a goal of reaching the largest-sum, at each step, the greedy algorithm will choose what appears to be the optimal immediate choice, so it will choose 12 instead of 3 at the second step, and will not reach the best solution, which contains 99.

Best First Search

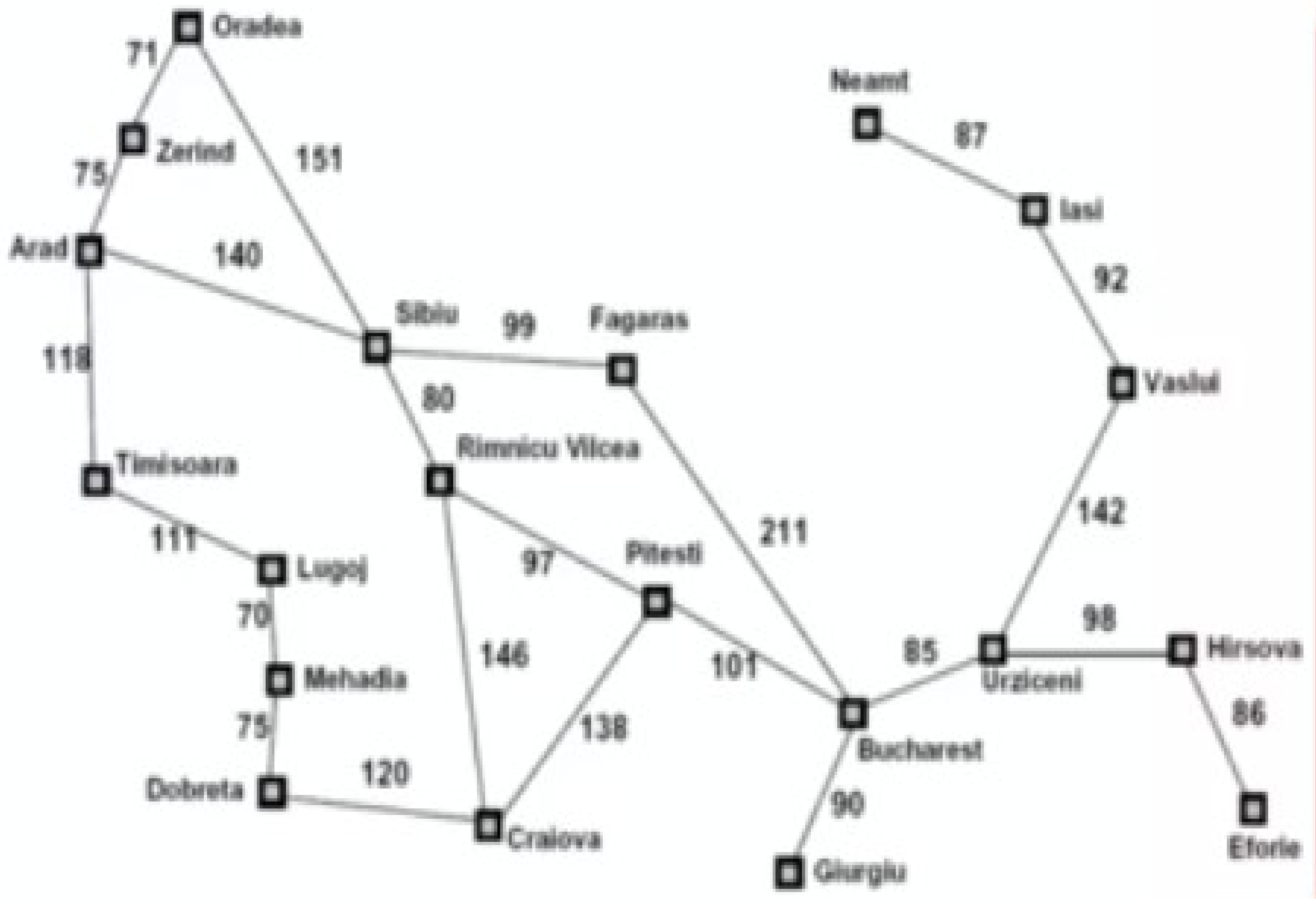
```
OPEN = [initial state]
While OPEN is not empty or until a goal is found
    Do
        1. Remove the best node from OPEN, call it n.
        2. If n is the goal state, backtrace path to n (through recorded parents)
and return path.
        3. Create n's successors.
        4. Evaluate each successor, add it to OPEN, and record its parent.
    Done
```

- This version of the algorithm is not complete
- It does not always find a possible path between two nodes, even if there is one.
- For example, it gets stuck in a loop if it arrives at a dead end, that is a node with the only successor being its parent.

Best First Search

```
OPEN = [initial state]
CLOSED = []
While OPEN is not empty
    Do
        1. Remove the best node from OPEN, call it n, add it to CLOSED.
        2. If n is the goal state, backtrace path to n (through recorded parents)
           and return path.
        3. Create n's successors.
        4. For each successor do:
            a. If it is not in CLOSED and it is not in OPEN: evaluate it, add it to
               OPEN, and record its parent.
            b. Otherwise, if this new path is better than previous one, change
               its recorded parent.
                    i.  If it is not in OPEN add it to OPEN.
                    ii. Otherwise, adjust its priority in OPEN using this new
                        evaluation.
Done
```

Example: Heuristic Function



Straight−line distance to Bucharest

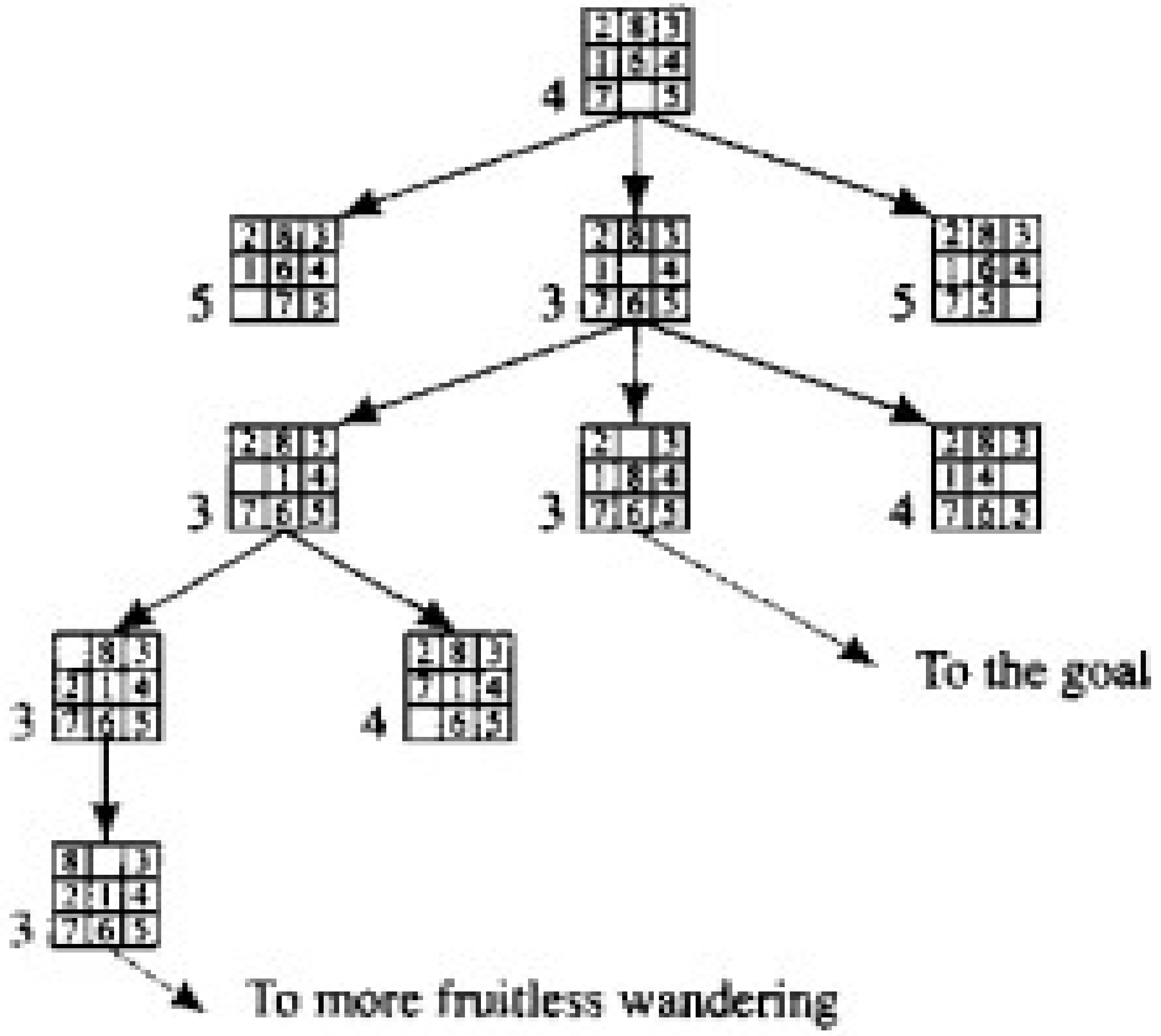
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Best First Search

- In case of 8 Puzzle problem the heuristic function could be number of tiles out of place compared to the goal.
  - Here f(n) = g(n)
- We can add a depth factor to avoid biasing the search in favor of going back to explore early paths.
  - Here f(n) = g(n) + h(n)

Best First Search



Greedy Search

- Best First Search takes you straight to the (wrong) goal.
- In worst case, like a badly guided DFS.
- Now the challenge is to consider all of these about heuristics and put it together.
-

Slow and Steady

Go fast and May be lost at some point of time

UCS

Greedy

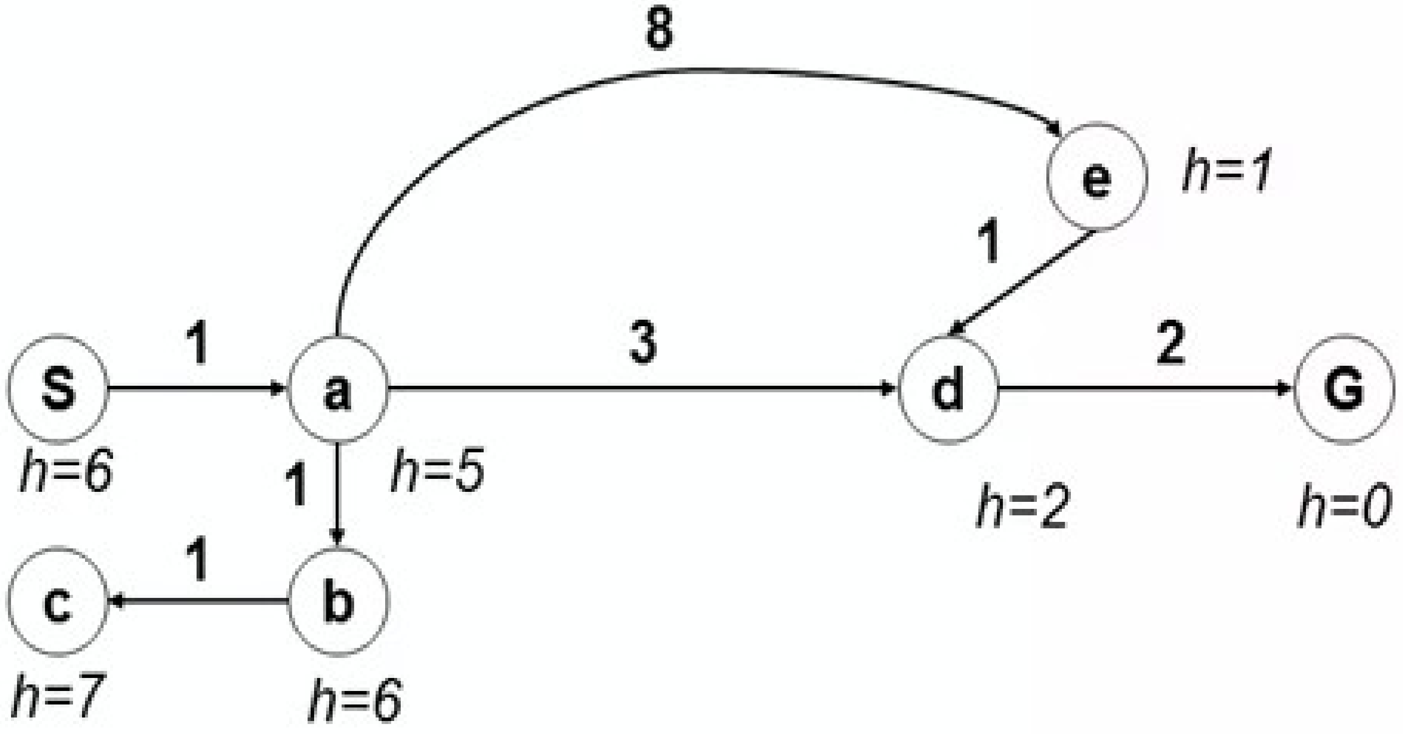
Turtle

Rabbit

A* Search

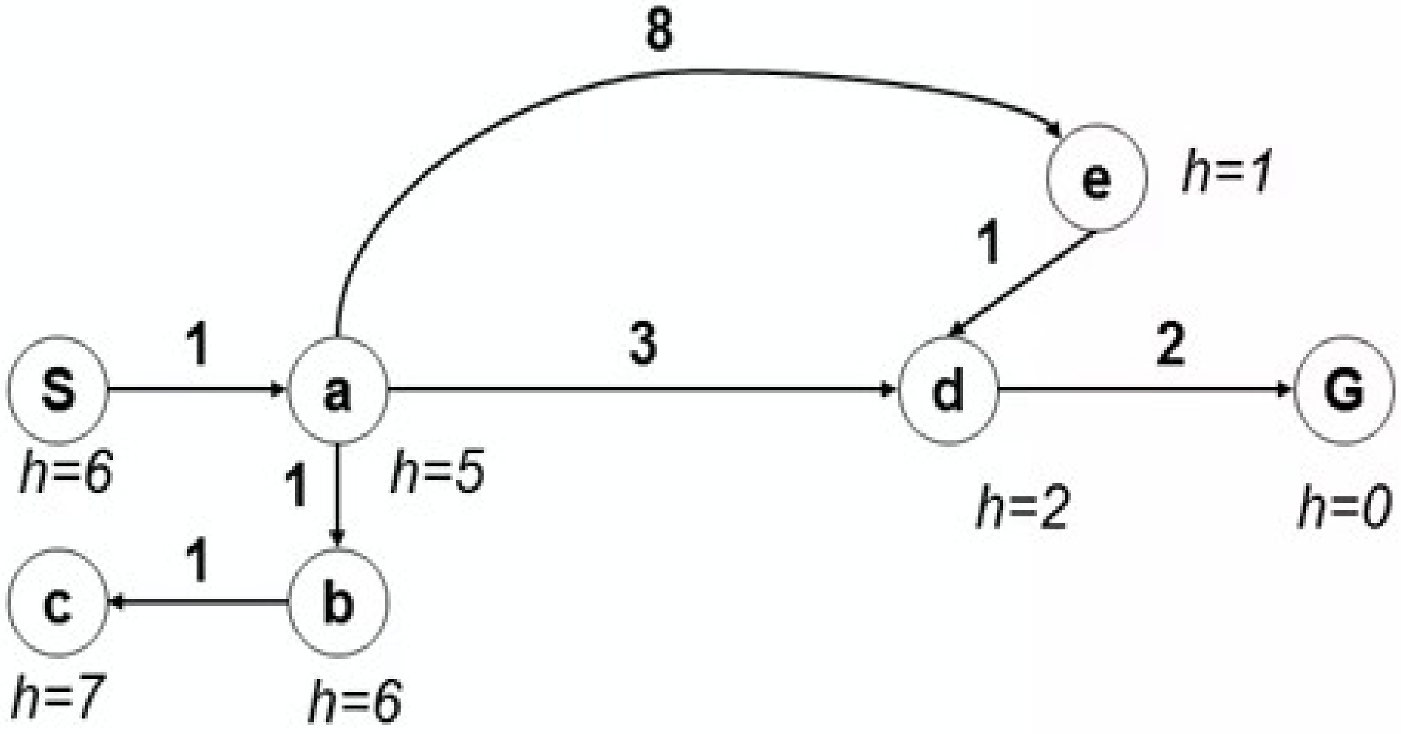
- We have to head towards the goal and head for the best.
- Slowly and steadily check that might be optimal
-

Combining UCS and Greedy



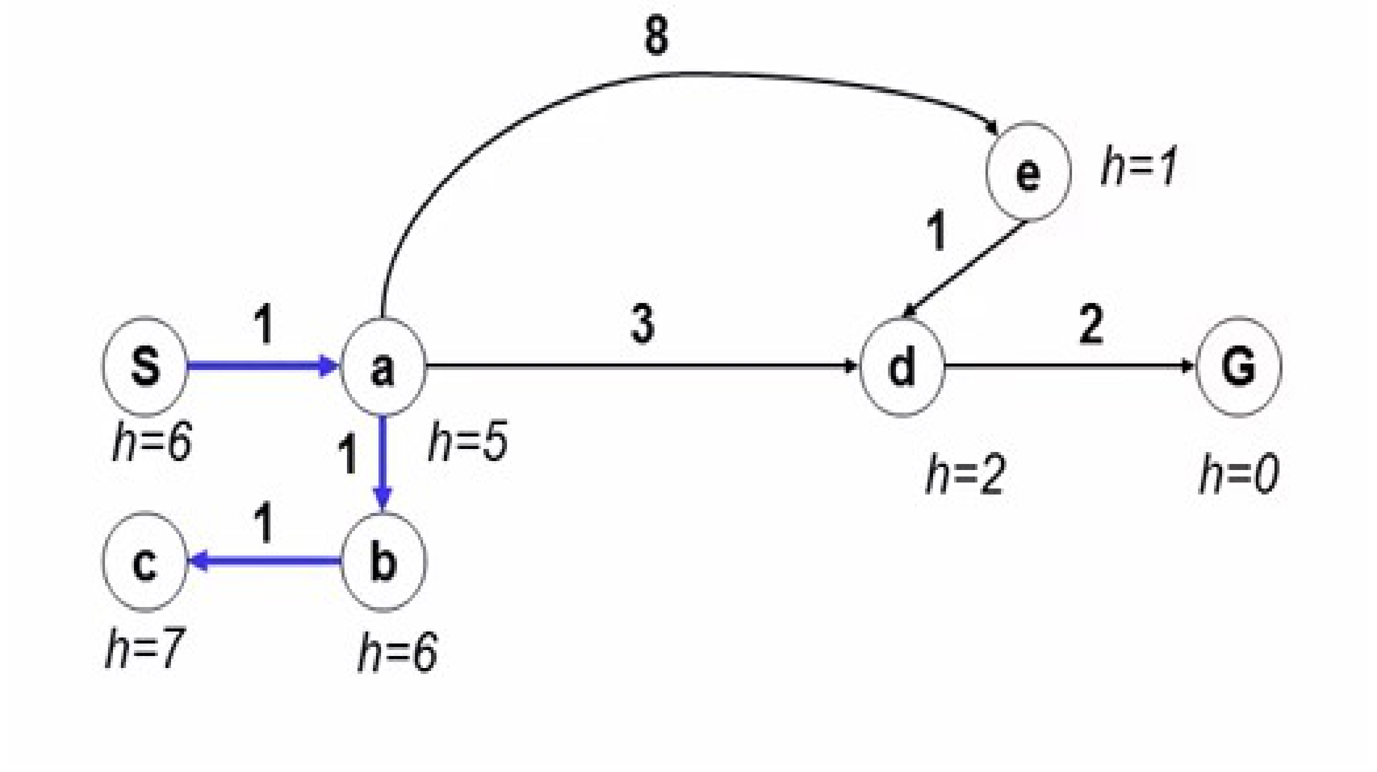
Combining UCS and Greedy

- Uniform cost orders by path cost, or backward cost g(n)



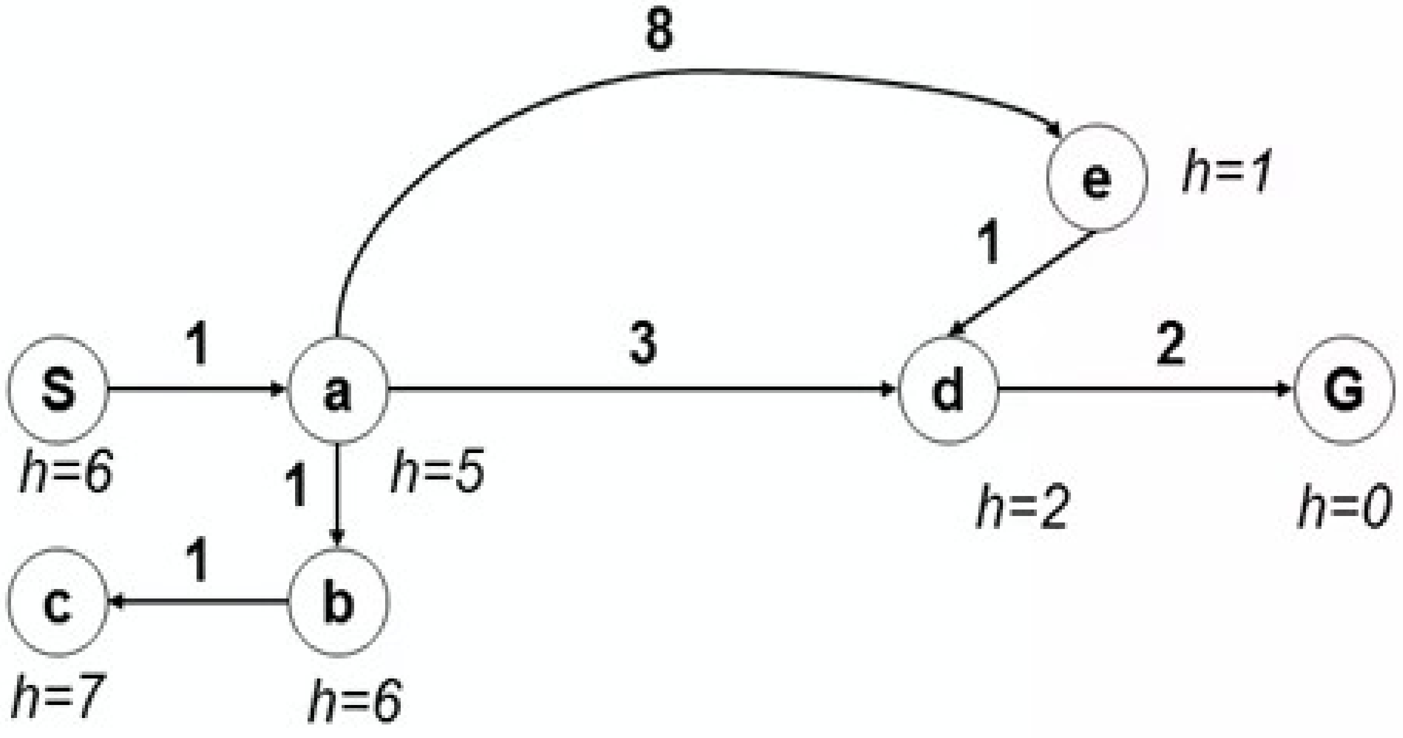
Combining UCS and Greedy

- Uniform cost orders by path cost, or backward cost g(n)



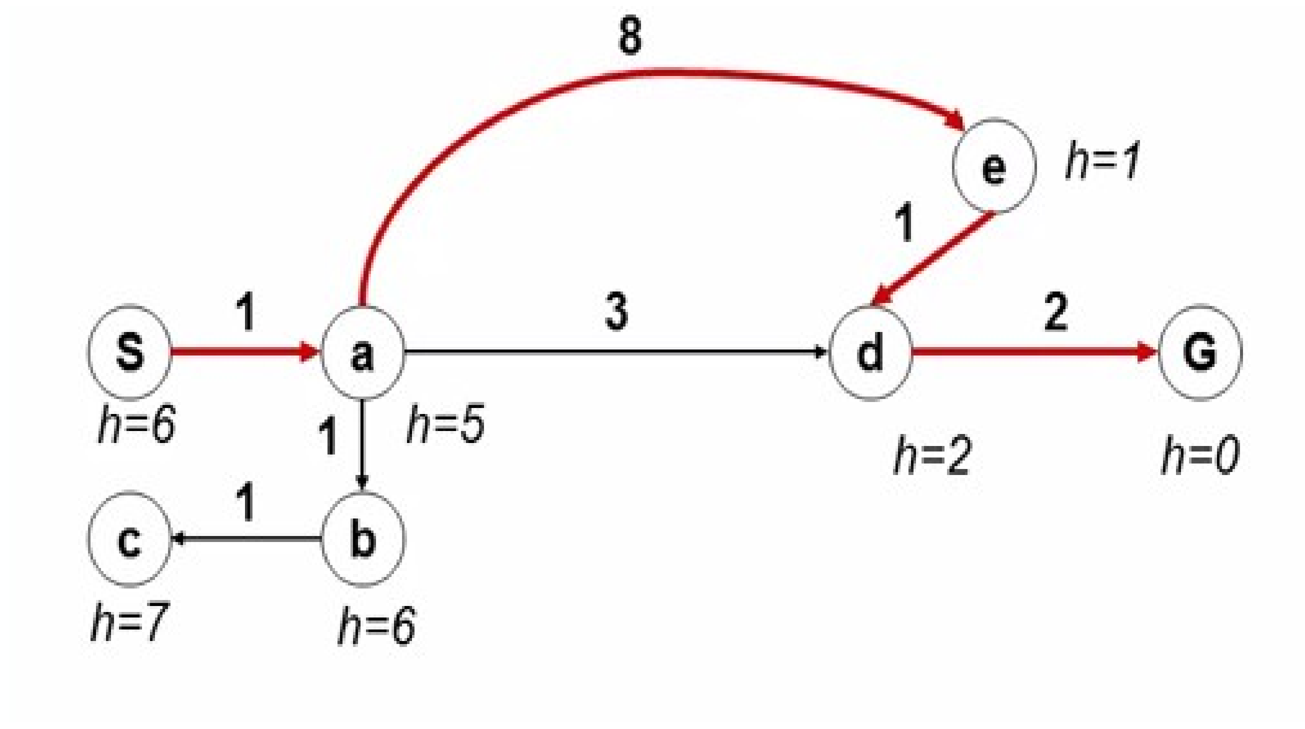
Combining UCS and Greedy

- Uniform cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



Combining UCS and Greedy

- Uniform cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



A* Algorithm

- It is an example of heuristic algorithm
- These are conditions which guarantees that the algorithm always finds minimal cost paths:
  - Each node in the graph has a finite number of successors (in any).
  - All arcs in the graph have costs greater than some positive threshold value.
  - The condition on cost of optimal path from a certain node n is:
    - For all node n in the search graph the estimate of the minimal cost path never over estimate the actual value.

Graph Searching Algorithm

1. Create a search tree T consisting solely of the start node n0.
    1. Put n0 on an ordered list called OPEN.
2. Create a list called CLOSED that is initially empty.
3. If OPEN is empty, exit with failure.
4. Select the first node on OPEN, remove it from OPEN, and put it on CLOSED.
    1. Call this node n.
5. If n is a goal node, exit successfully with the solution obtained by tracing a path backward along the arcs in T from n0.
6. Expand node n, generating a set, M, of successors. Install M as successors of n in T by creating arcs from n to each member of M.
7. Reorder the list OPEN, either according to some arbitrary scheme or according to heuristic merit.
8. Go to step 3.

Graph Searching Algorithm

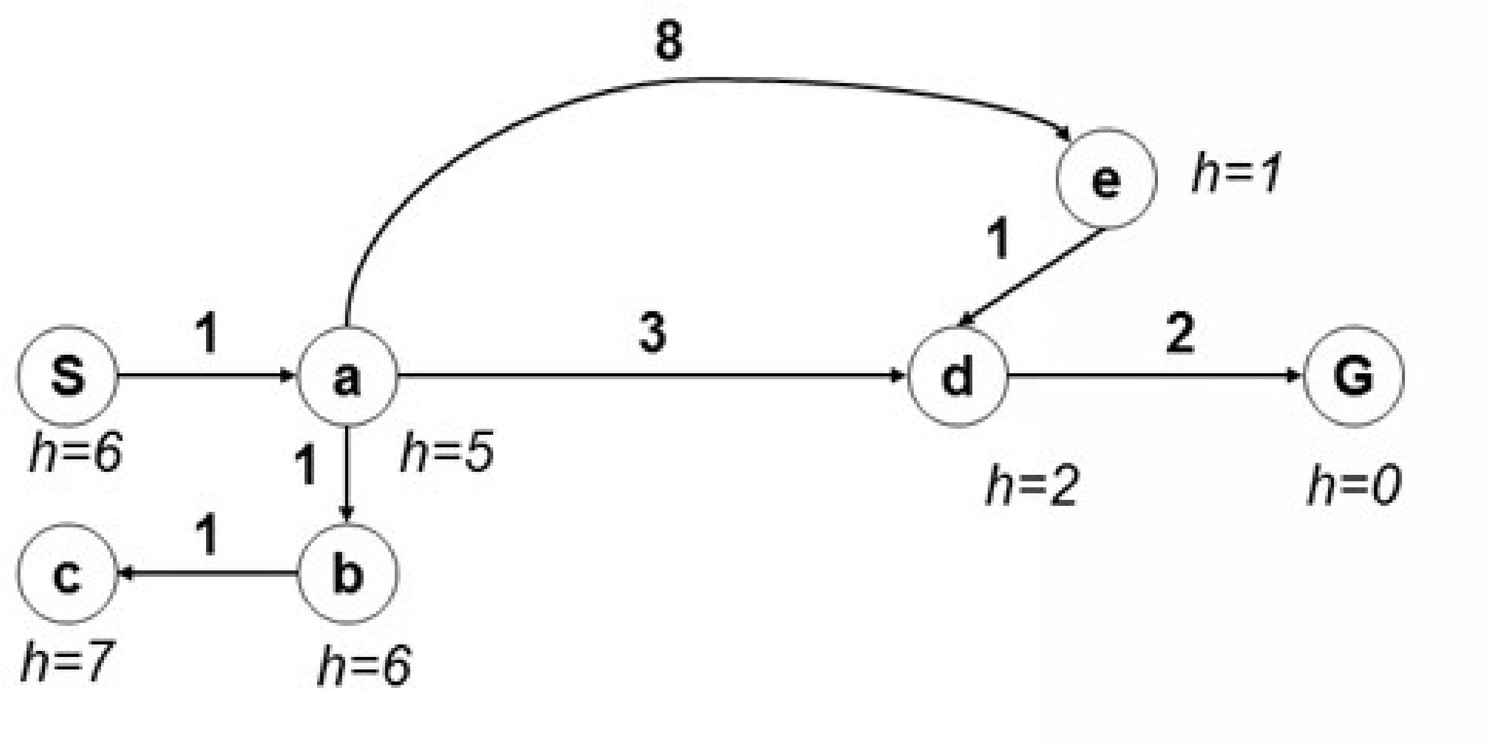
- This graph search algorithm can be used to perform best first search, breadth first search, or depth first search.
- In breadth first search:
  - New nodes are simply put at the end of OPEN (FIFO order) and nodes are not reordered.
- In depth first search:
  - New nodes are put at the beginning of OPEN (LIFO order)
- In best first search:
  - OPEN is reordered according to the heuristic merit of the nodes
- Step 6 create a loop in the search tree/graph. To avoid it could be written as
  - Expand node n, generating a set, M, of successors that are not already parents of n in T. Install M as successors of n in T by creating arcs from n to each member of M.

A* Algorithm

1. Create a search graph G consisting solely of the start node n0. Put n0 on a list called OPEN.
2. Create a list called CLOSED that is initially empty.
3. If OPEN is empty, exit the failure.
4. Select the first node on OPEN, remove it from OPEN, and put it on CLOSED. Call this node n.
5. If n is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from n to n0 in G.
    1. The pointers define a search tree and are established in step 7.
6. Expand node n, generating the set M of its successors that are not already ancestors of n in G. Install these members of M as successors of n in G.
7. Establish a pointer to n from each of these members of M that were not already in G (i.e., not already on either OPEN or CLOSED). Add these members of M to OPEN. For each member m of M that was already on OPEN or CLOSED, redirect its pointer to n if the best path to m found so far is through n. For each member of M already on CLOSED, redirect the pointers of each of its descendants in G so that they point backward along the best paths found so far to these descendants.
8. Reorder the list OPEN in order of increasing heuristic function values. The ties among minimal function values are resolved in favor of the deepest node in the search tree.
9. Go to step 3.

Combining UCS and Greedy

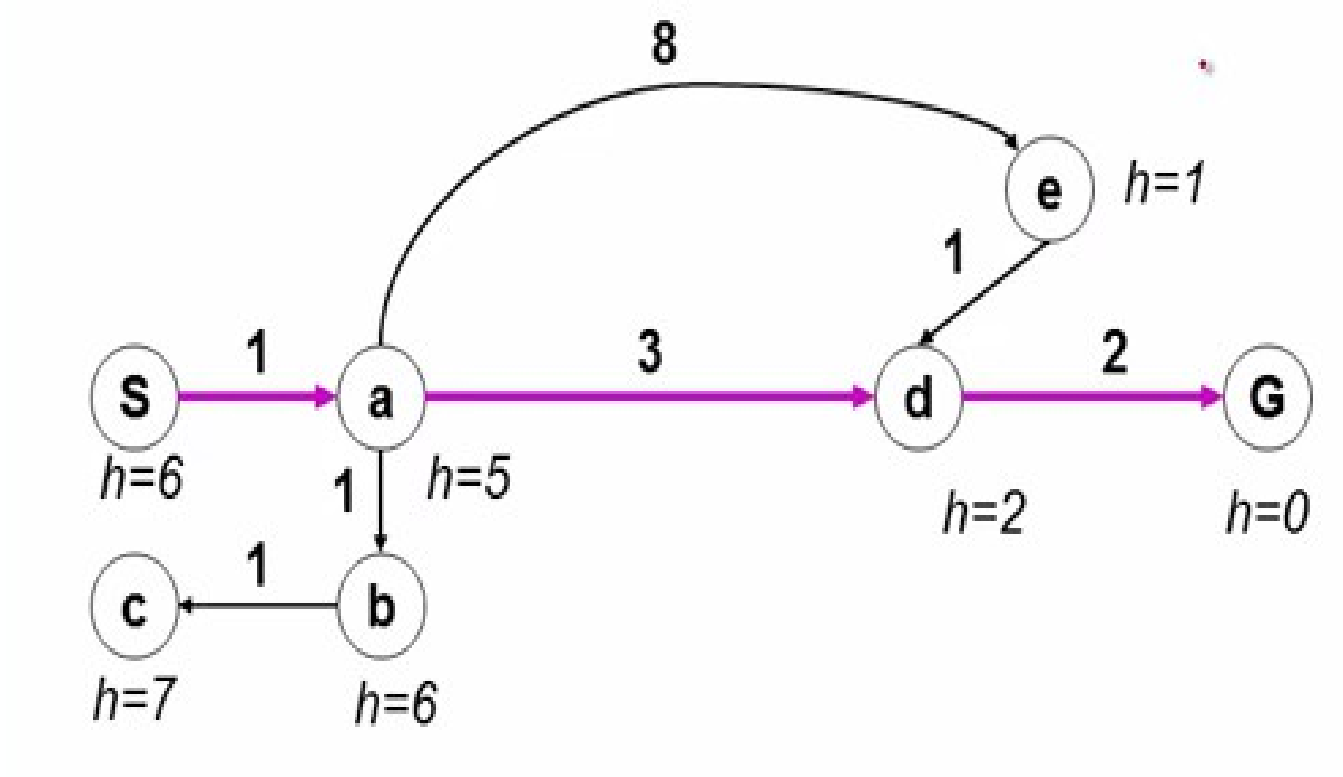
- Uniform cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



- A* search orders by the sum: f(n) = g(n) + h(n)

Combining UCS and Greedy

- Uniform cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



- A* search orders by the sum: f(n) = g(n) + h(n)

A* Search

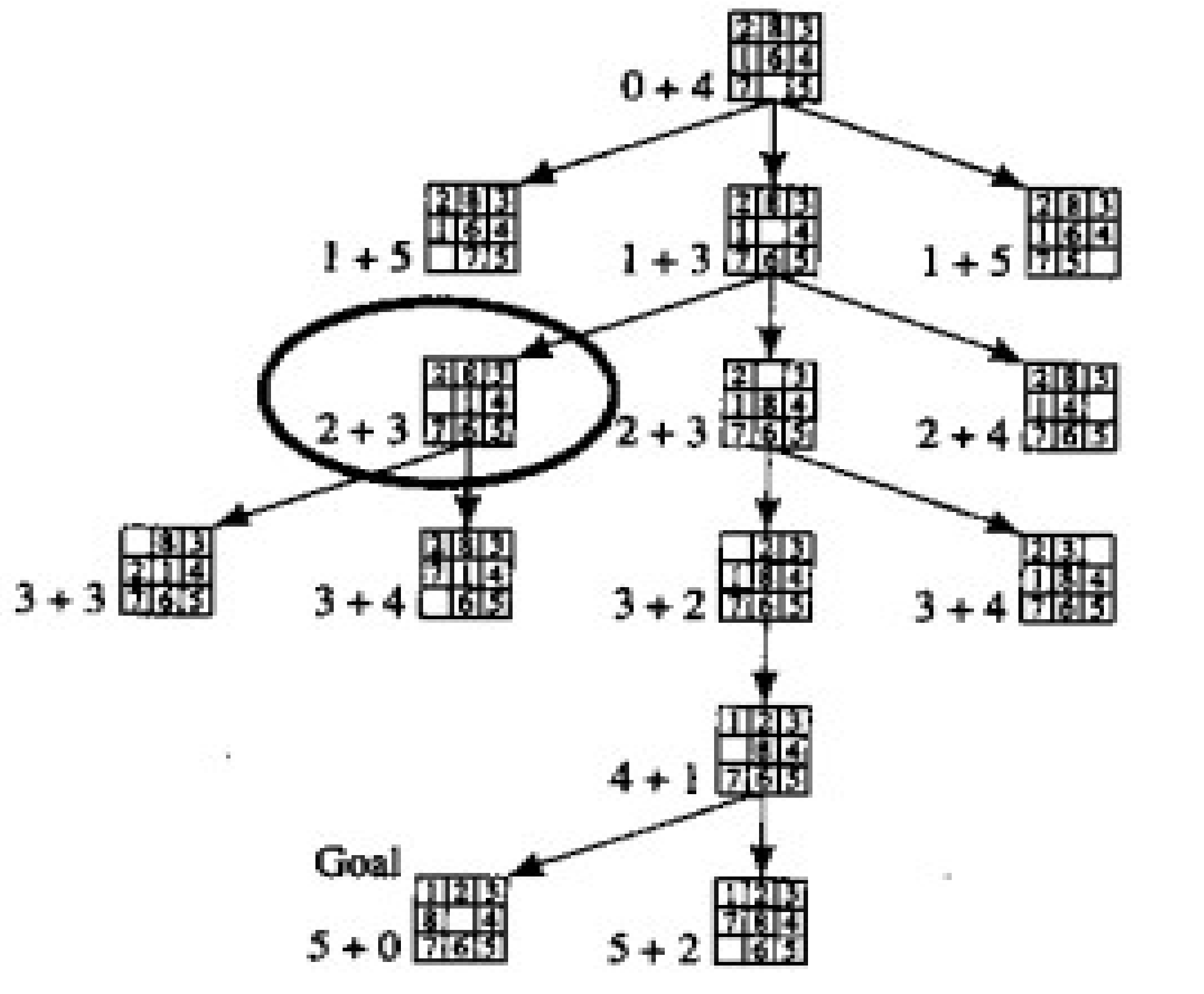
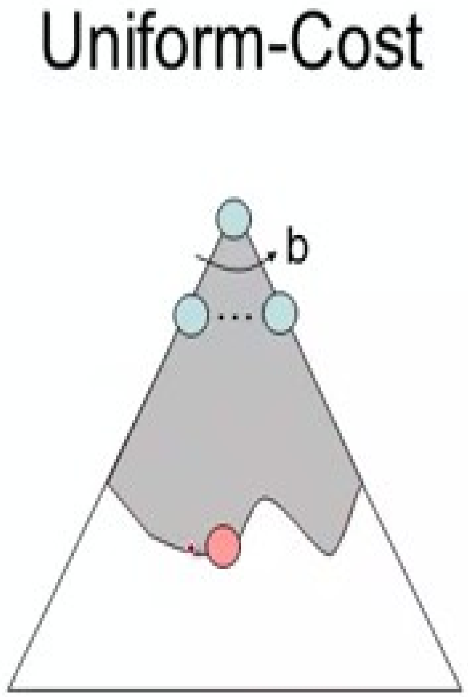
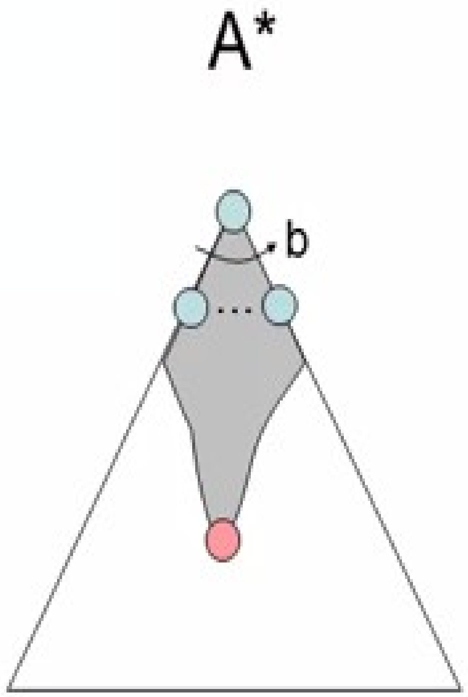


Figure 9.2

Heuristic Search Using $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$

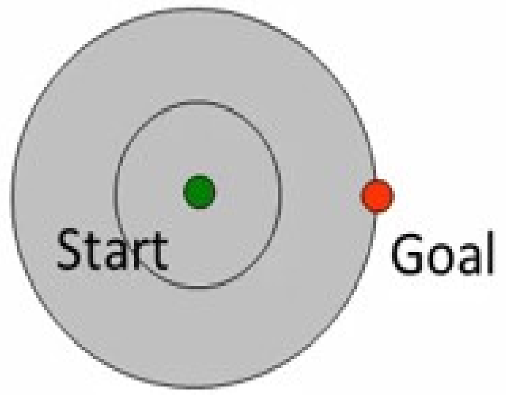
UCS vs. A*



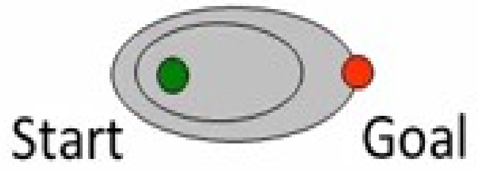


UCS vs. A*

- Uniform-cost expands equally in all "directions"

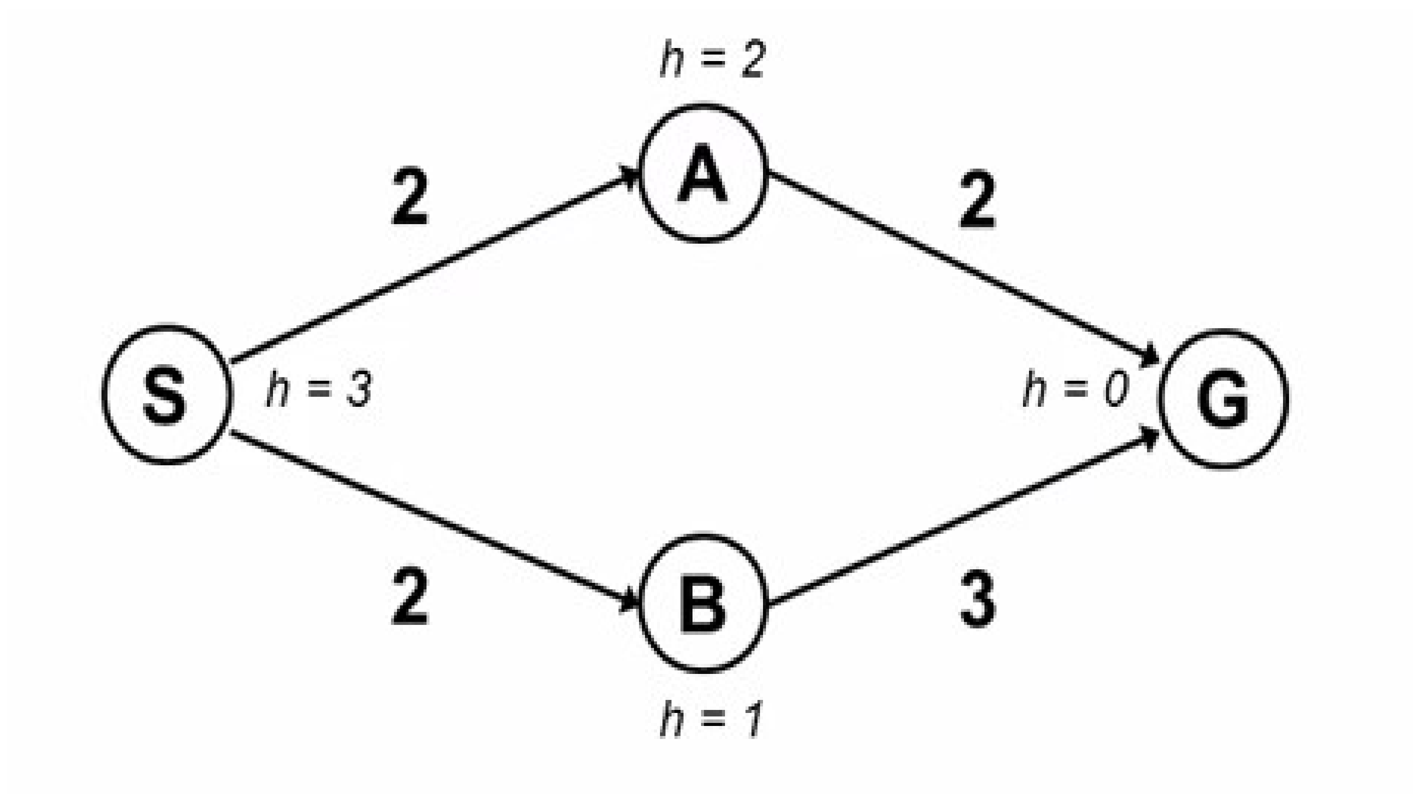


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



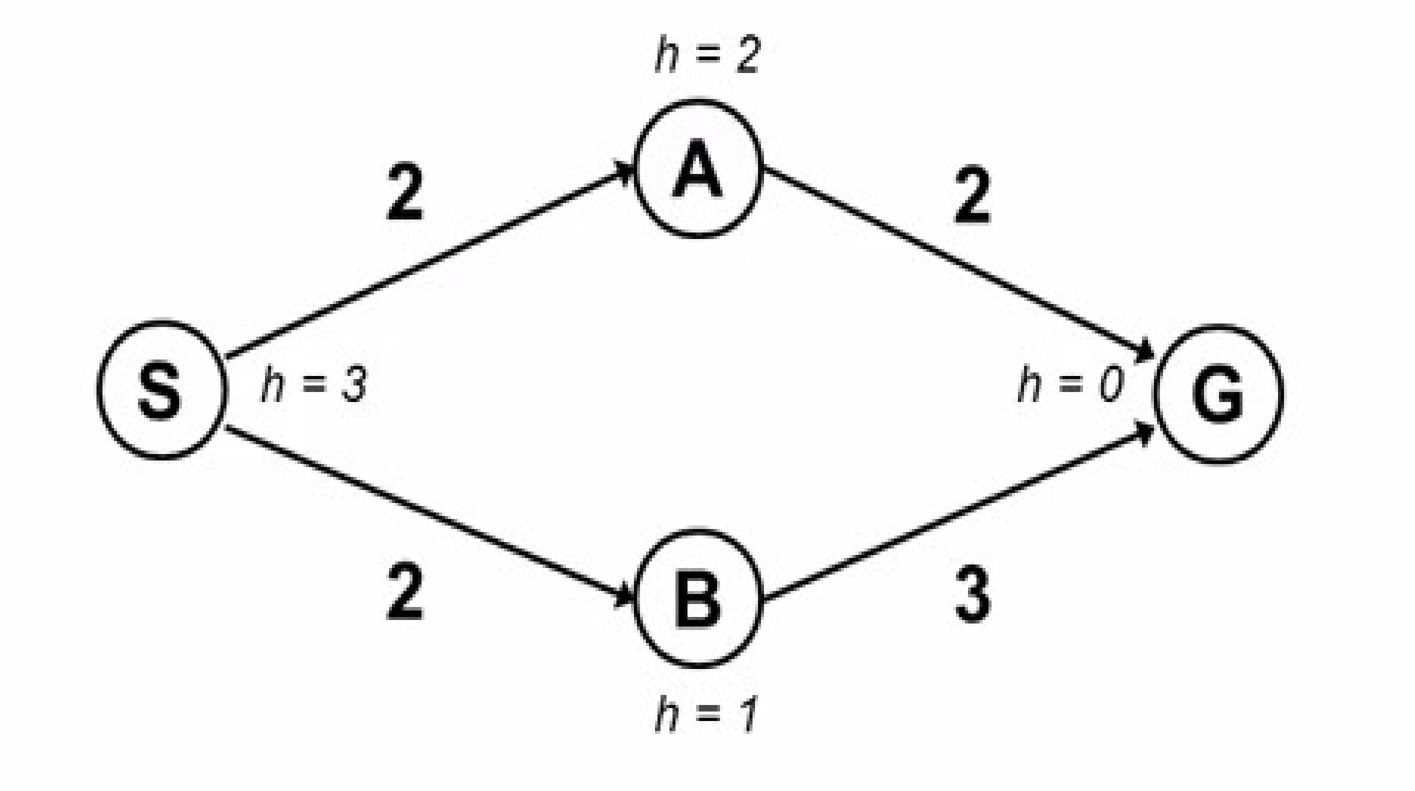
When should A* terminate?

- Should we stop when we enqueue a goal?



When should A* terminate?

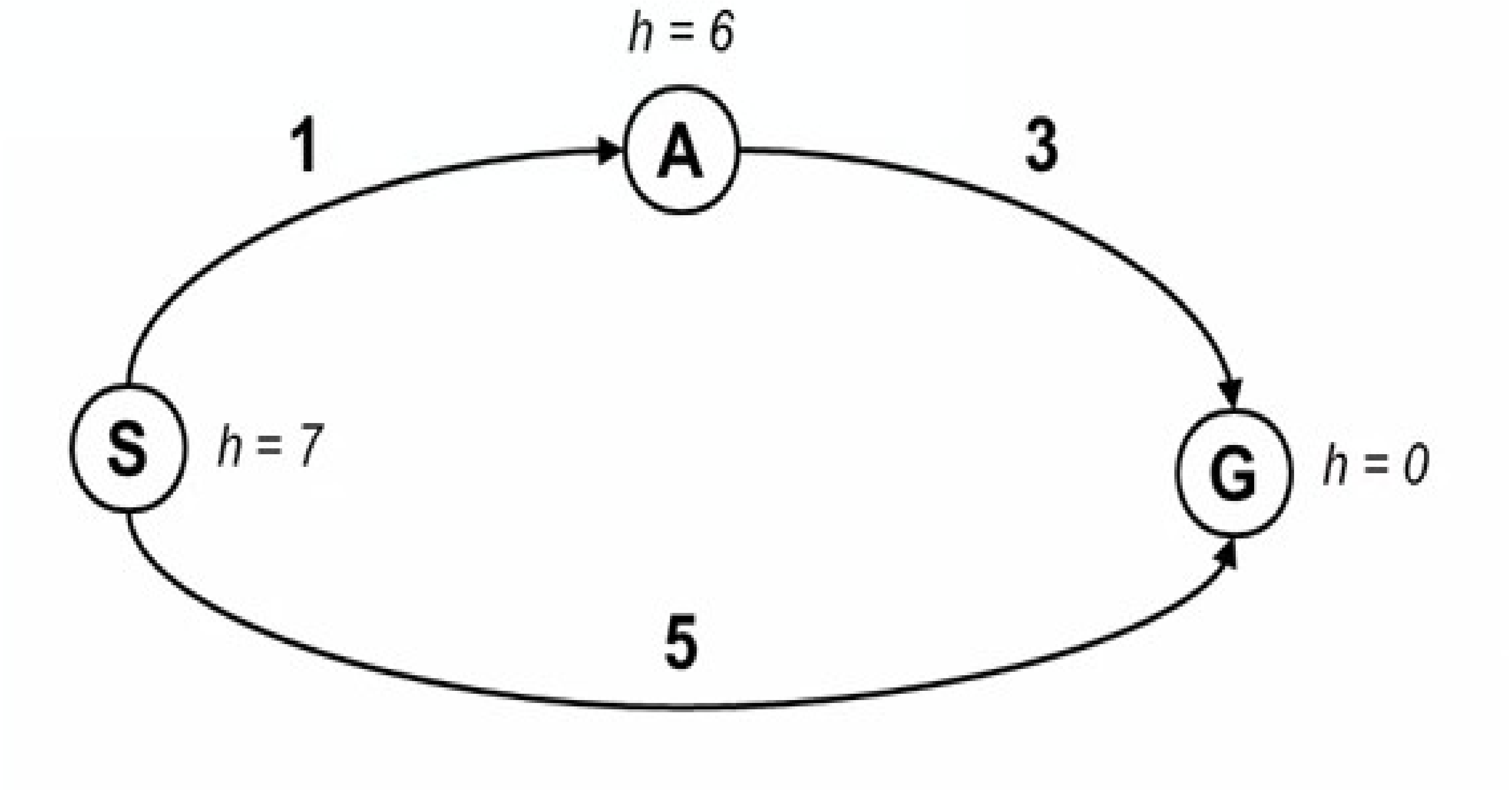
- Should we stop when we enqueue a goal?



- No, only stop when we dequeue a goal.

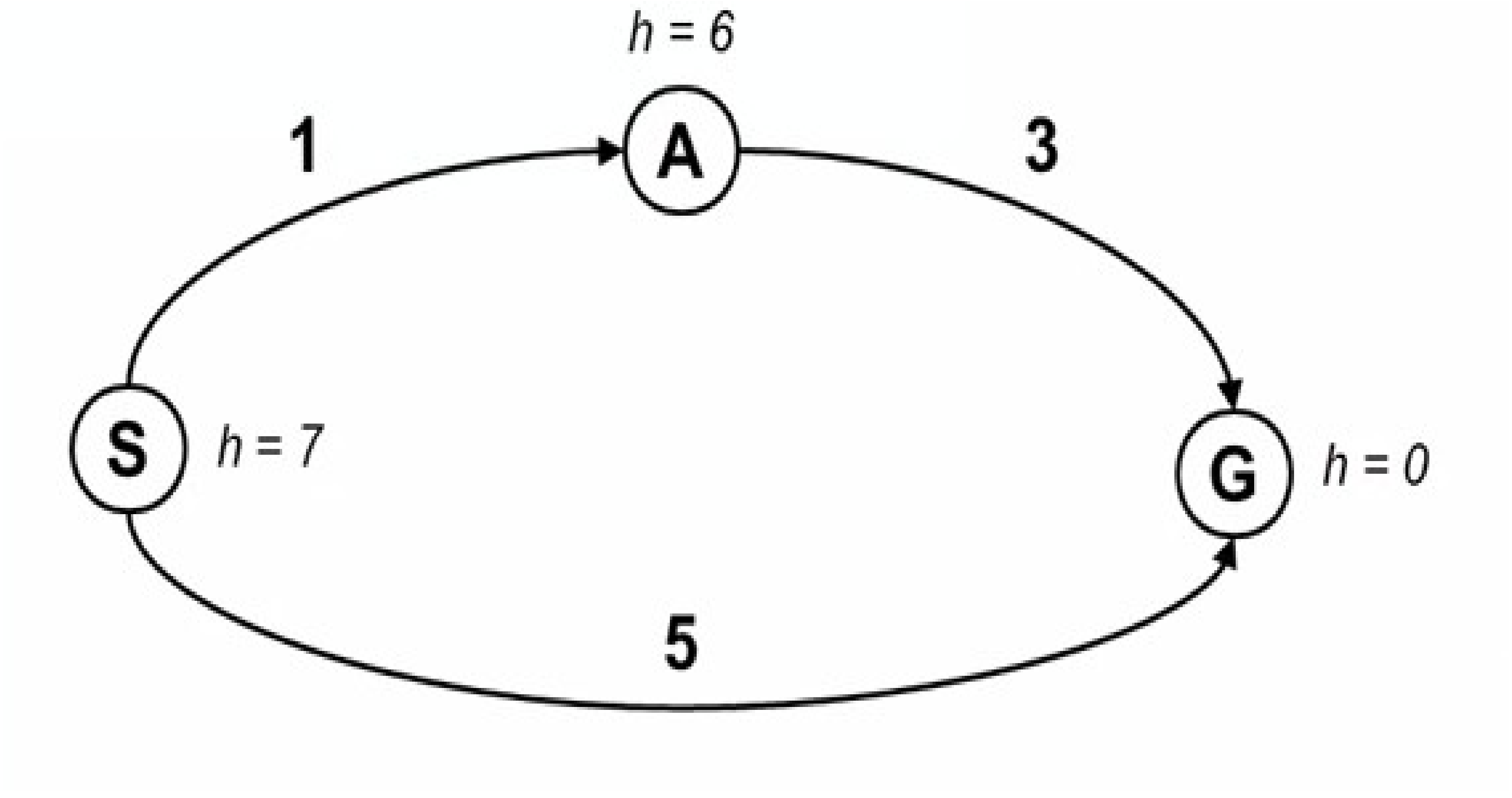
Is A* optimal?

- What went wrong?



Is A* optimal?

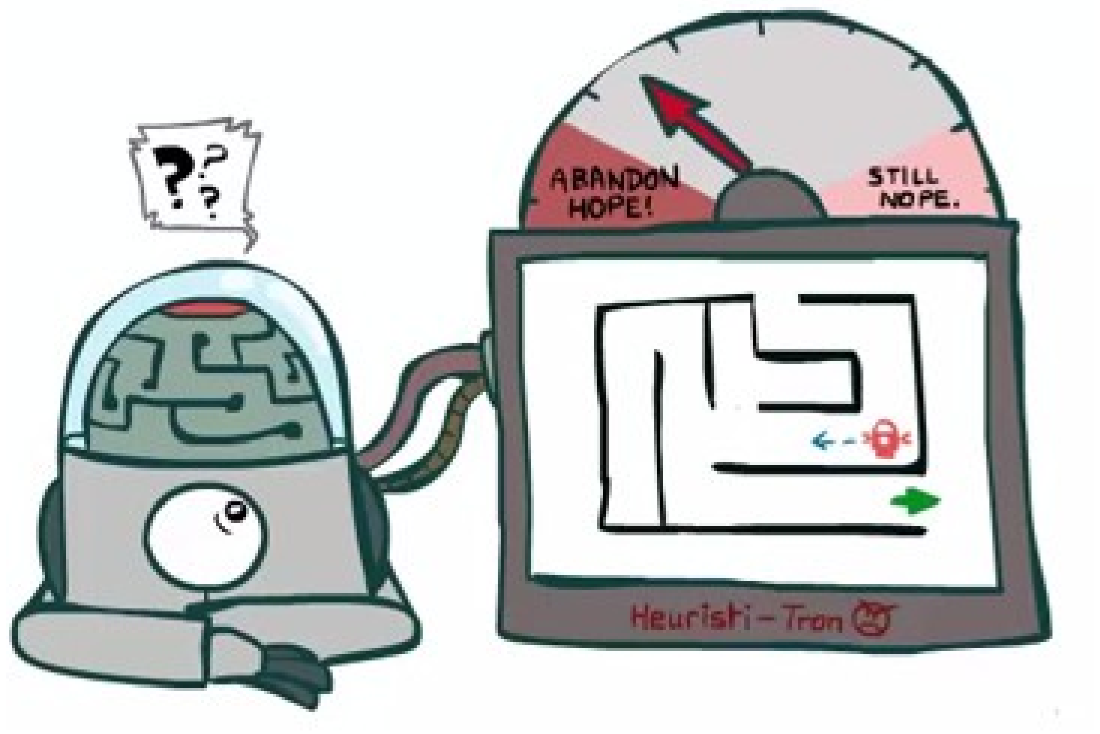
- What went wrong?



- Actual bad goal cost < Estimated good goal cost
- We need estimates to be less than actual costs!!

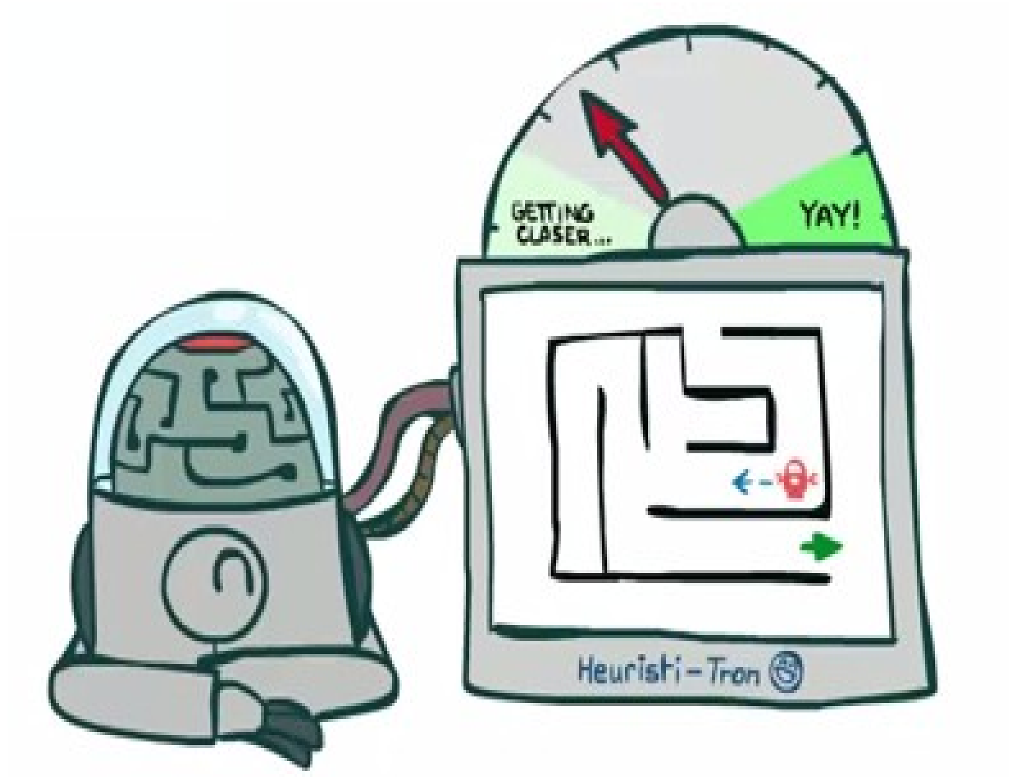
Idea: Admissibility

- Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe.
-



Idea: Admissibility

- Admissible (Optimistic) heuristics slow down bad plans but never outweigh true costs



Admissible Heuristics

- A heuristic h is admissible (optimistic) if
  - $0 <= h(n) <= h^*(n)$

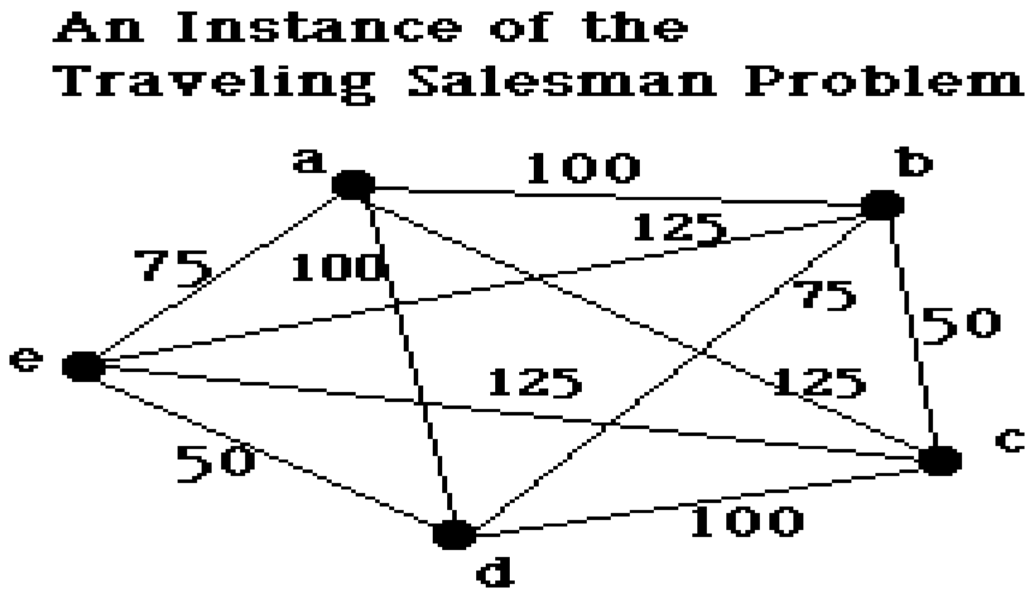
    where $h^*(n)$ is the true cost to a nearest goal.

A* Applications

- Video games
- Routing problems
- Resource planning problems
- Robot motion planning
- Language Analysis
- Machine Translation
- Speech Recognition

Travelling Sales Man Problem

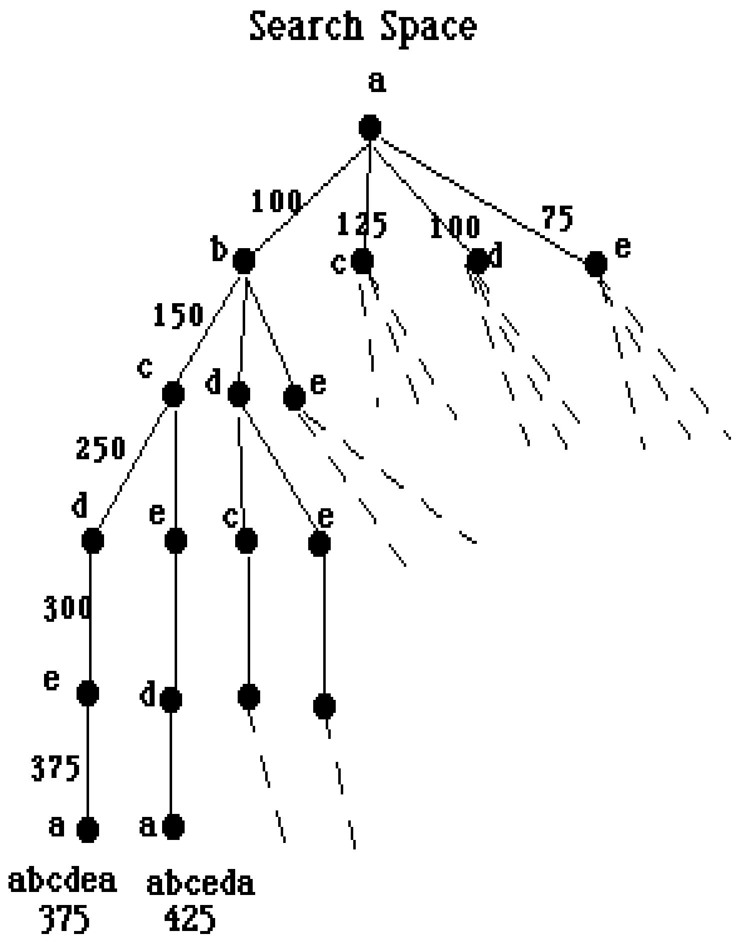
- Start at A, visit all cities, return to A.
- Links show cost of each trip (distance, money).
- Find trip with minimum cost.
- Solution is a path. e.g. [A,D,C,B,E,A]
- What is the shortest path?



Travelling Sales Man Problem

- Representing Travelling salesman problem as state-space problem.
- Start at A, 4 choices for first step, 3 choices for next, 2 choices, 1 choice.
- 4! paths = 24 paths
- What assumption did we make?
  - Given n cities, (n-1) choices for 1st stop

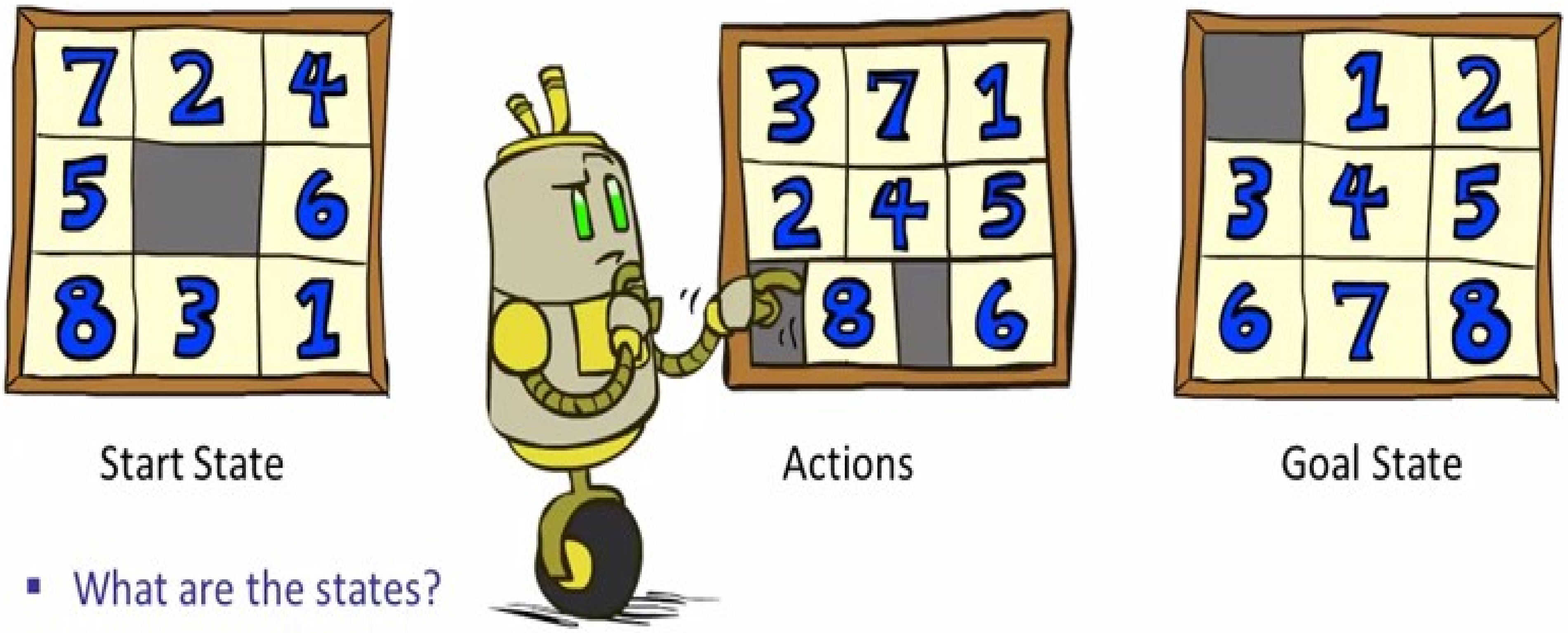
    (n-2) choices for next stop, etc.
  - (n-1)! paths
  - With many cities or nodes this soon becomes intractable.



Travelling Sales Man Problem

- Solve the problem Breadth First Search, Depth First Search, Best First Search,
- Uniform Cost Search and A* search by self defining heuristic function.

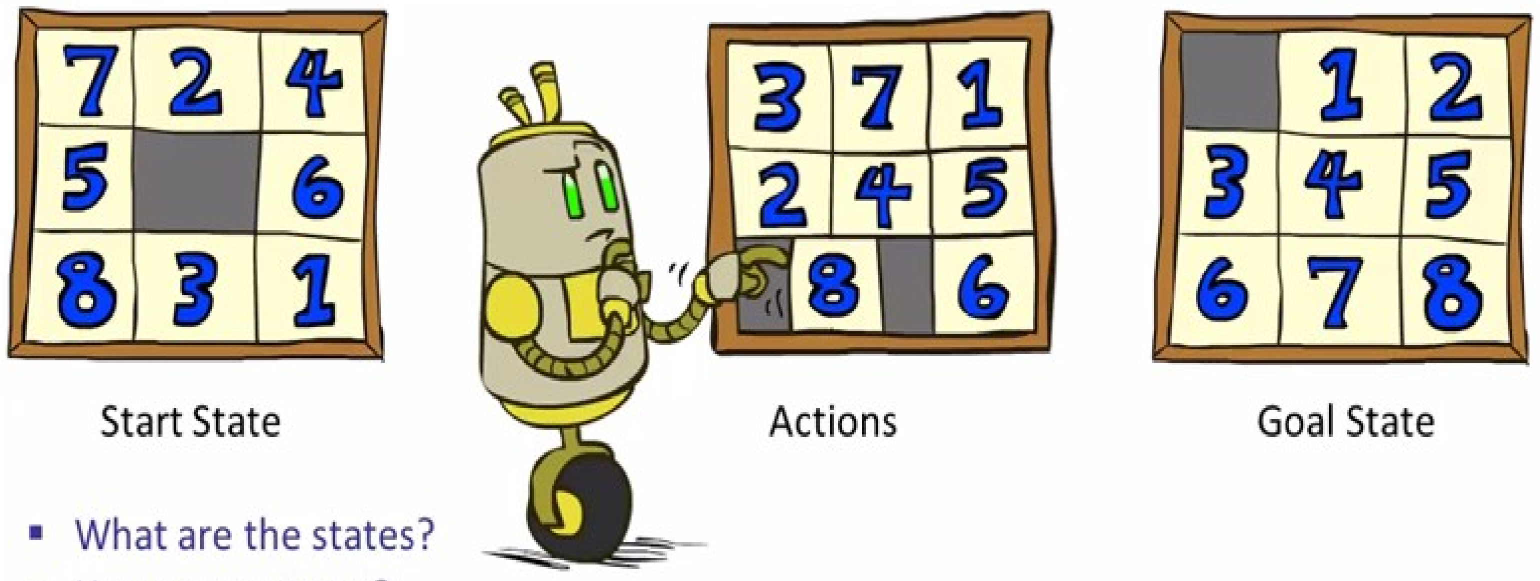
8 Puzzle Problem



- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle Problem

- Hints:
  - Heuristic: Number of tiles misplaced
  - Why is it admissible?
  - h(start) = 8



- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?