# CMSC 471

## Intro to AI

# Search (Cont.)

# General search algorithm

*;; problem describes the start state, operators, goal test, and operator costs*

*;; queueing-function is a comparator function that ranks two states*

*;; general-search returns either a goal node or failure*

```
function general-search (problem, QUEUEING-FUNCTION)

  nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
                problem.OPERATORS))
 end
```

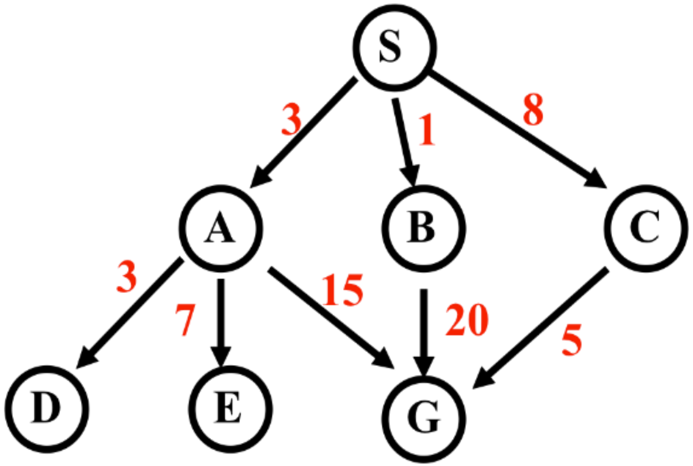
*;; Note: The goal test is NOT done when nodes are generated*

*;; Note: This algorithm does not detect loops*

# Properties of Searching Strategies

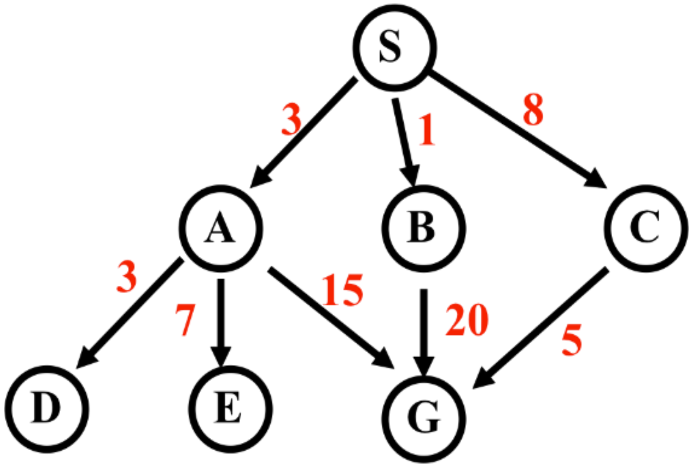
- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
  - Usually measured by maximum size of graph/tree during the search
- **Optimality/Admissibility**
  - If a solution is found, is it **guaranteed** to be an optimal one, i.e., one with minimum cost

# Uninformed Search (Recap)




| Search Strategy | Expanded nodes |
|---|---|
| BFS | |
| DFS | |
| Iterative-Deepening Search | |
| Uniform-Cost Search | |

# Uninformed Search (Recap)




| Search Strategy | Expanded nodes |
|---|---|
| BFS | SABCDEG |
| DFS | SADEG |
| Iterative-Deepening Search | SSABCSADEG |
| Uniform-Cost Search | SADBCEG |

# Comparing Search Strategies

| Search Strategy | Time | Space | Complete? | Optimal? |
|---|---|---|---|---|
| BFS | | | | |
| DFS | | | | |
| Iterative-Deepening Search | | | | |
| Uniform-Cost Search | | | | |

b = Branching Factor ; d = depth

# Comparing Search Strategies

| Search Strategy | Time | Space | Complete? | Optimal? |
|---|---|---|---|---|
| BFS | $b^d$ | | | |
| DFS | $b^d$ | | | |
| Iterative-Deepening Search | $b^d$ | | | |
| Uniform-Cost Search | $b^d$ | | | |

$b$ = Branching Factor ; $d$ = depth

# Comparing Search Strategies

| Search Strategy | Time | Space | Complete? | Optimal? |
|---|---|---|---|---|
| BFS | $b^d$ | $b^d$ | | |
| DFS | $b^d$ | bd | | |
| Iterative-Deepening Search | $b^d$ | bd | | |
| Uniform-Cost Search | $b^d$ | $b^d$ | | |

b = Branching Factor ; d = depth

# Comparing Search Strategies

| Search Strategy | Time | Space | Complete? | Optimal? |
|---|---|---|---|---|
| BFS | $b^d$ | $b^d$ | Yes | |
| DFS | $b^d$ | bd | No | |
| Iterative-Deepening Search | $b^d$ | bd | Yes | |
| Uniform-Cost Search | $b^d$ | $b^d$ | Yes | |

b = Branching Factor ; d = depth

# Comparing Search Strategies

| Search Strategy | Time | Space | Complete? | Optimal? |
|---|---|---|---|---|
| BFS | $b^d$ | $b^d$ | Yes | Yes (Only for unitary cost) |
| DFS | $b^d$ | bd | No | No |
| Iterative-Deepening Search | $b^d$ | bd | Yes | Yes (Only for unitary cost) |
| Uniform-Cost Search | $b^d$ | $b^d$ | Yes | Yes |

b = Branching Factor ; d = depth

# Notes on Uniform-Cost Search

- Greedy Algorithm
  - Algorithms that make locally optimal choice in the hopes of reaching global optima
- Uniform because:
  - Tries to reach uniform cost on the priority queue or node list

# UCS vs BFS vs DFS

- DFS:
  - Goes too far down the depth before backtracking
  - Can find solution in less time
- BFS:
  - Goes too far down the width before backtracking
  - Optimal
- UCS
  - Tries to find a balance between going too far deep or wide

# Informed (Heuristic) Search

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A* Search
- Heuristic functions

# Big idea: heuristic

**Merriam-Webster's Online Dictionary:**

Heuristic (pron. \hyu-’ris-tik\): adj. [from Greek *heuriskein* to discover] involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods

# Heuristics, More Formally

$h(n)$ is a **heuristic function**, that maps a state $n$ to an estimated cost from $n$-to-goal

# Heuristics, More Formally

$h(n)$ is a **heuristic function**, that maps a state $n$ to an estimated cost from $n$-to-goal

$h(n)$ is **admissible** iff $h(n) \leq$ the lowest actual cost from $n$-to-goal

# Heuristics, More Formally

$h(n)$ is a **heuristic function**, that maps a state $n$ to an estimated cost from $n$-to-goal

$h(n)$ is **admissible** iff $h(n) \leq$ the lowest actual cost from $n$-to-goal

$h(n)$ is **consistent** iff
$h(n) \leq \text{lowestcost}(n, n') + h(n')$

# Informed methods add domain-specific information

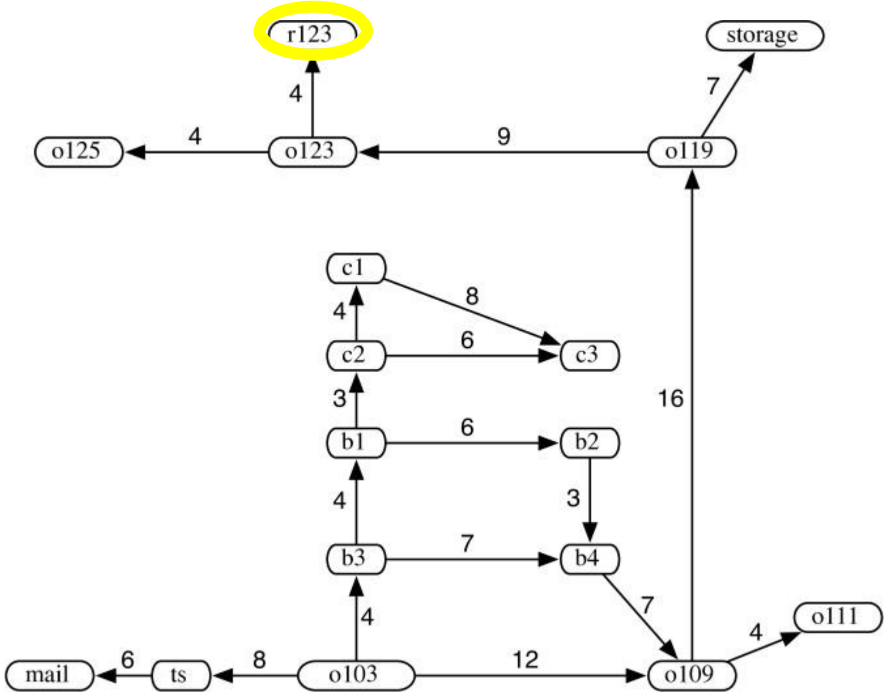
- Select best path along which to continue searching
- h(n): estimates *goodness* of node n
- h(n) = **estimated cost** (or distance) of minimal cost path from n **to a goal state**.
- Based on domain-specific information and computable from current state description that estimates how close we are to a goal

# Heuristics

- **All domain knowledge** used in search is encoded in the **heuristic function, h(<node>)**
- Examples:
  - 8-puzzle: number of tiles out of place
  - 8-puzzle: sum of distances each tile is from its goal

In general

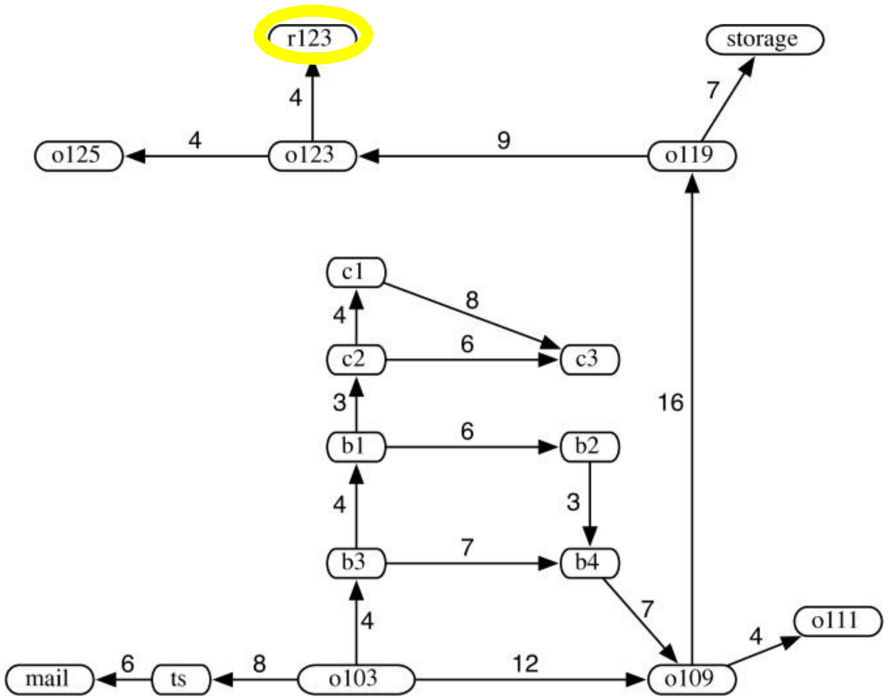
- $h(n) \geq 0$ for all nodes n
- $h(n) = 0$ implies that n is a goal node
- $h(n) = \infty$ implies n is a dead-end that can't lead to goal




# Example 3.5

$h(o123) = 4 \qquad h(o125) = 6 \qquad h(r123) = 0$

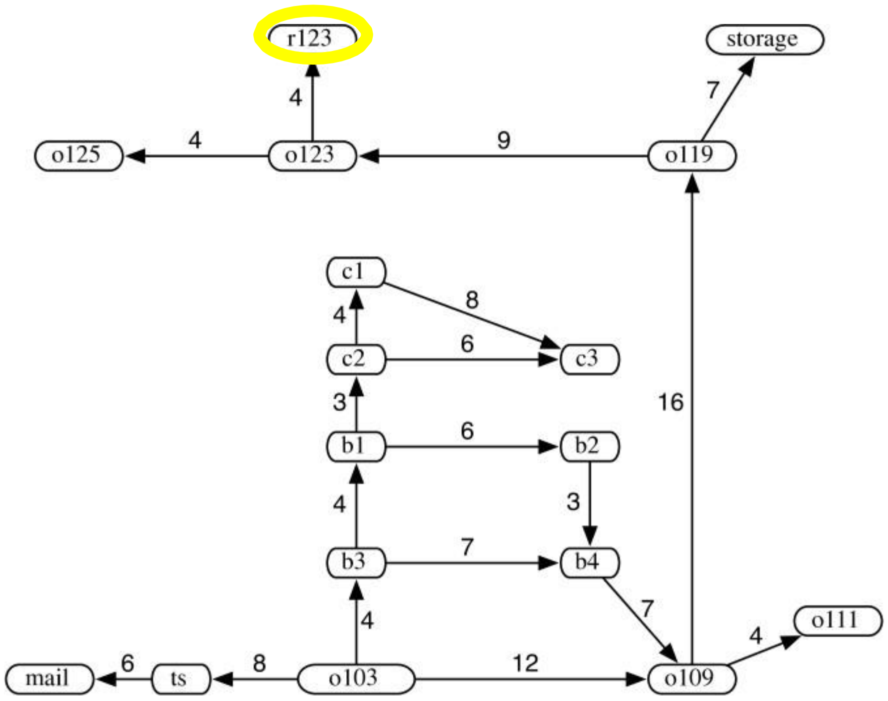
(Partial) Heuristic $h(n)$
for goal r123




# Example 3.5

(Partial) Heuristic $h(n)$
for goal r123

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

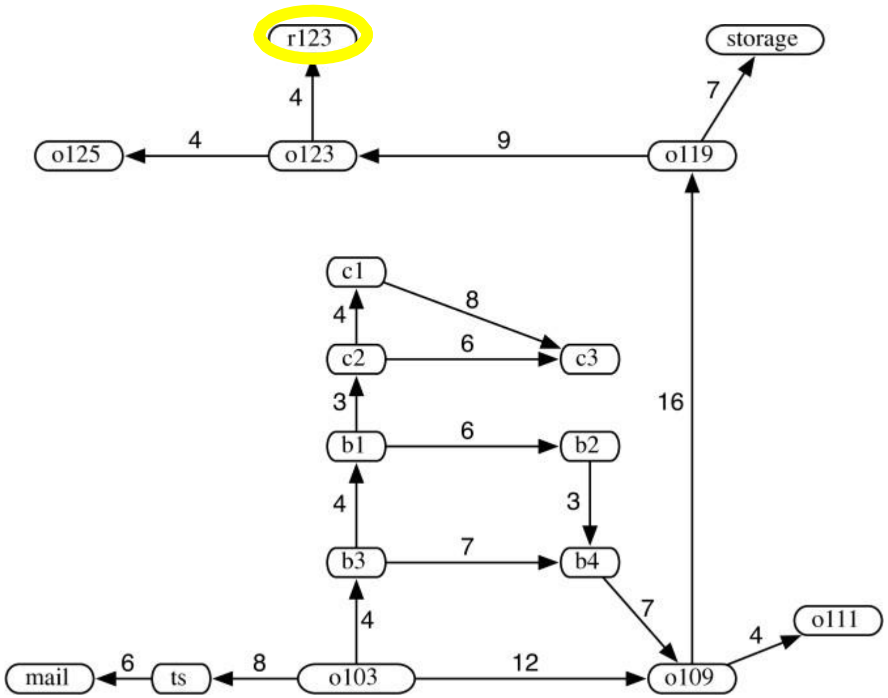



# Example 3.5

**Q: Is this an admissible heuristic?**

(Partial) Heuristic $h(n)$
for goal r123

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




(Partial) Heuristic $h(n)$
for goal r123

# Example 3.5

Q: Is this an **admissible** heuristic?

Q: Is it an accurate heuristic?

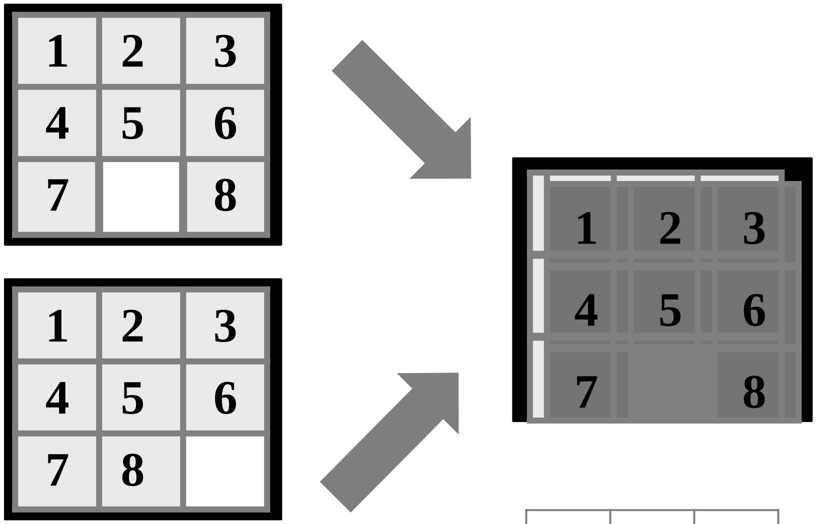
| | | |
|---|---|---|
| $h(mail) = 26$ | $h(ts) = 23$ | $h(o103) = 21$ |
| $h(o109) = 24$ | $h(o111) = 27$ | $h(o119) = 11$ |
| $h(o123) = 4$ | $h(o125) = 6$ | $h(r123) = 0$ |
| $h(b1) = 13$ | $h(b2) = 15$ | $h(b3) = 17$ |
| $h(b4) = 18$ | $h(c1) = 6$ | $h(c2) = 10$ |
| $h(c3) = 12$ | $h(storage) = 12$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

# Heuristics for 8-puzzle

*The number of* ***misplaced tiles*** *(not including the blank)*

Current State

Goal State




| | | |
|---|---|---|
| N | N | N |
| N | N | N |
| N | Y | |

In this case, only “**8**” is misplaced, so heuristic function evaluates to 1

In other words, the heuristic *says* that it *thinks* a solution may be available in just 1 more move

# Heuristics for 8-puzzle

**Manhattan Distance** (not including the blank)

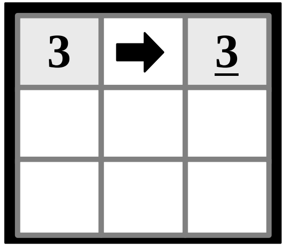
Current State

| | | |
|---|---|---|
| 3 | 2 | 8 |
| 4 | 5 | 6 |
| 7 | 1 | |

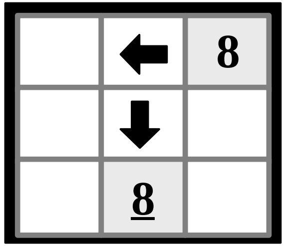
Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

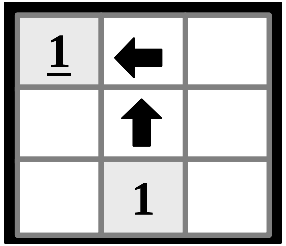
- The **3**, **8** and **1** tiles are misplaced (by 2, 3, and 3 steps) so the heuristic function evaluates to 8
- Heuristic says that it *thinks* a solution may be available in just 8 more moves.
- The misplaced heuristic's value is 3




2 spaces




3 spaces

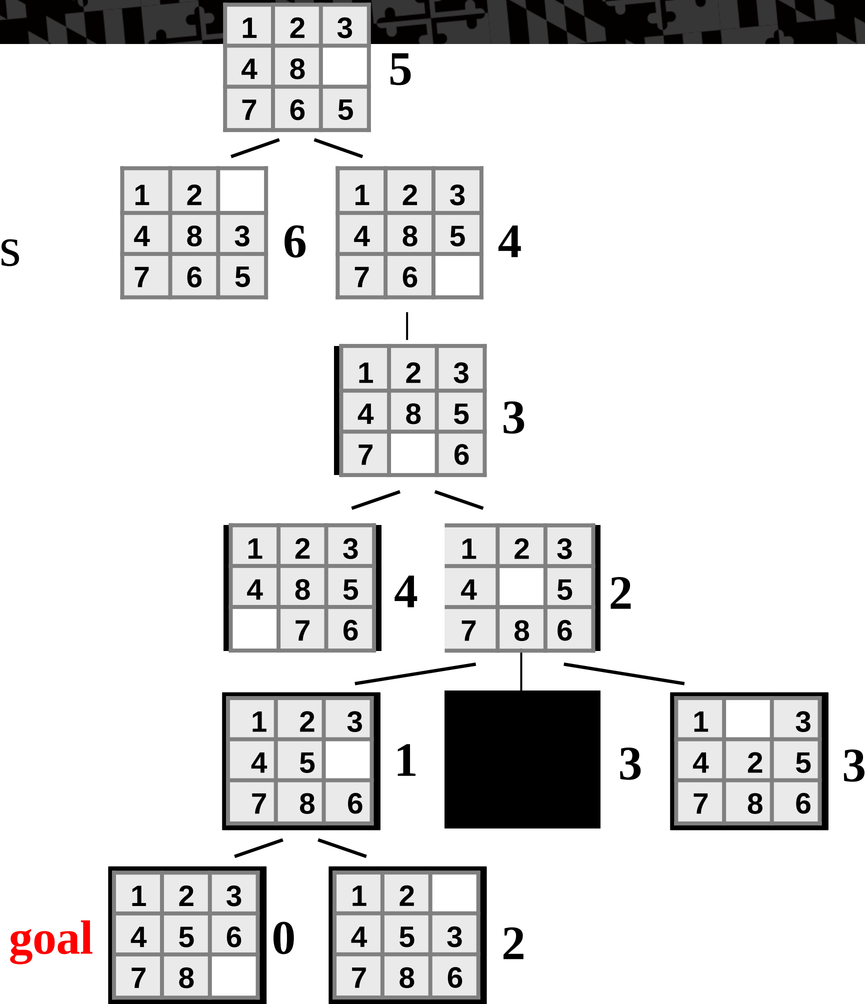



3 spaces

**Total 8**

We can use heuristics to guide search

Manhattan Distance heuristic helps us quickly find a solution to the 8-puzzle



# Best-first search

- Search algorithm that improves **depth- first search** by expanding most promising node chosen according to heuristic rule

- Order nodes on nodes list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information

# Best-first search

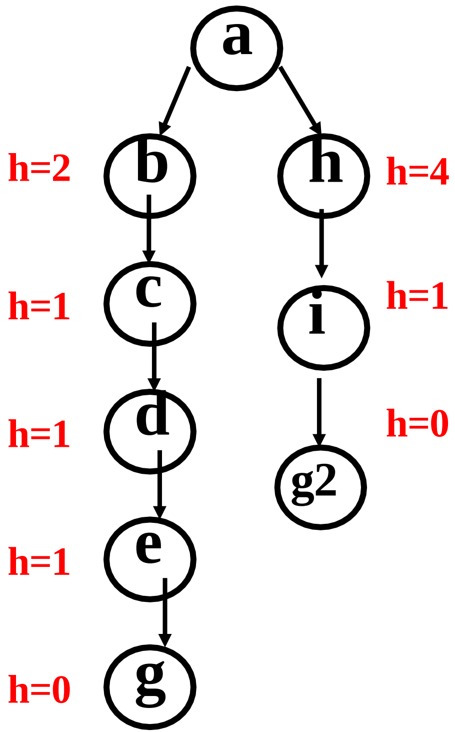
- Search algorithm that improves **depth- first search** by expanding most promising node chosen according to heuristic rule
- Order nodes on nodes list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information
- This is a generic way of referring to the class of informed methods

# Greedy best first search

- A greedy algorithm makes locally optimal choices in hope of finding a global optimum
- Uses evaluation function $f(n) = h(n)$, sorting nodes by increasing values of $f$
- Selects node to expand appearing **closest** to goal (i.e., node with smallest f value)
- Not complete
- Not Admissible

# Greedy best first search example

- Proof of non-admissibility
  - Assume arc costs = 1, greedy search finds goal g, with solution cost of 5
  - Optimal solution is path to goal with cost 3



# Beam search

- Use evaluation function f(n), but maximum size of the nodes list is k, a fixed constant
- Only keep k best nodes as candidates for expansion, discard rest
- k is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As k increases, approaches best first search
- Complete?
- Admissible?

# Beam search

- Use evaluation function f(n), but maximum size of the nodes list is k, a fixed constant
- Only keep k best nodes as candidates for expansion, discard rest
- k is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As k increases, approaches best first search
- **Not Complete**
- **Not Admissible**

We’ve *got* to be able to do better, right?

# A* Search

## Use an evaluation function

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

estimated total cost from start to goal via state n = minimal-cost path from the start state to state n + cost estimate from state n to the goal

# A* Search

- Use an evaluation function

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

estimated total cost from start to goal via state n = minimal-cost path from the start state to state n + cost estimate from state n to the goal

- g(n) term adds “breadth-first” component to evaluation function
- Ranks nodes on search frontier by estimated cost of solution from start node *via given node* to goal

# A*

- Pronounced "*a star*"
- h is **admissible** when h(n) <= h*(n) holds
  - **h*(n)** = *true cost* of *minimal cost path* from n to a goal
- Using an admissible heuristic guarantees that 1st solution found will be an **optimal** one
- A* is **complete** whenever branching factor is finite and every action has fixed, positive cost
- A* is **admissible**

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4* **4** (2): 100–107.

# Implementing A*

Q: Can this be an instance of our general search algorithm?

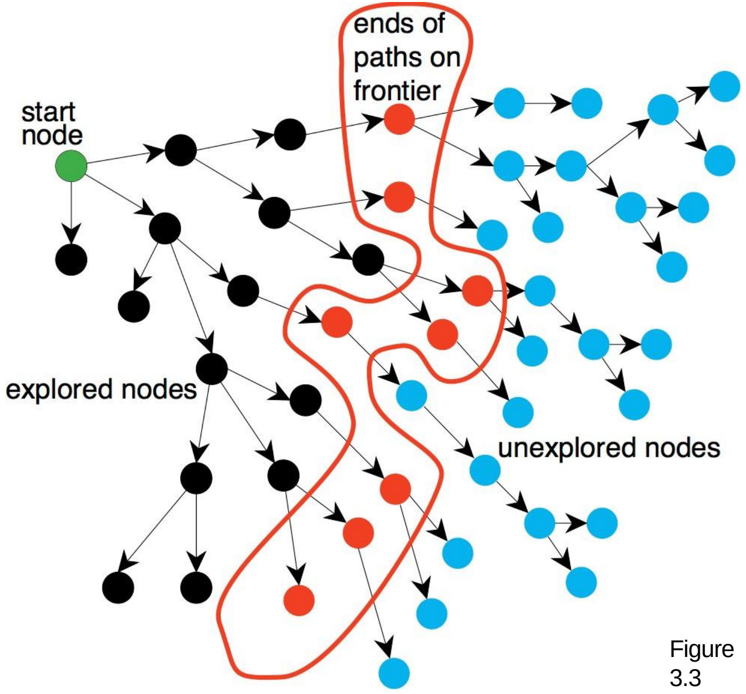



Figure 3.3

# Implementing A*

Q: Can this be an instance of our general search algorithm?

A: Yup! Just make the fringe a priority queue ordered by $f(n)$

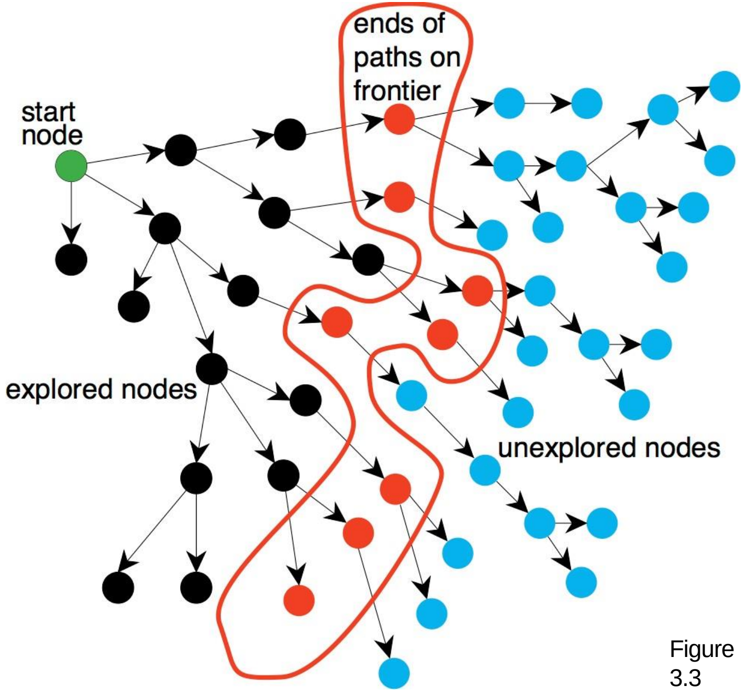



Figure 3.3

# Alternative A* Pseudo-code

1 Put the start node S on the nodes list, called OPEN

2 If OPEN is empty, exit with failure

3 Select node in OPEN with minimal f(n) and place on CLOSED

4 If n is a goal node, collect path back to start and stop

5 Expand n, generating all its successors and attach to them pointers back to n. For each successor n' of n

  1 If n’ not already on OPEN or CLOSED

  - put n' on OPEN
  - compute h(n'), g(n')=g(n)+ c(n,n'), f(n')=g(n')+h(n')

  2 If n’ already on OPEN or CLOSED and if g(n') is lower for new version of n', then:

  - Redirect pointers backward from n’ on path with lower g(n’)
  - Put n' on OPEN

# Observations on A*

- **Perfect heuristic:** If $h(n) = h^*(n)$ for all n, only nodes on an optimal solution path expanded; no extra work is done

# Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search

# Observations on A*

- **Perfect heuristic:** If $h(n) = h^*(n)$ for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If $h(n) = 0$ for all n, then it is an admissible heuristic and A* acts like uniform-cost search
- **Better heuristic:** If $h1(n) < h2(n) <= h^*(n)$ for all non-goal nodes, then h2 is a *better* heuristic than h1

# Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search
- **Better heuristic:** If h1(n) < h2(n) <= h*(n) for all non-goal nodes, then h2 is a *better* heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*

    i.e., A1 expands at least as many nodes as A2*
  - We say that A2* is *better informed* than A1*

# Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search
- **Better heuristic:** If h1(n) < h2(n) <= h*(n) for all non-goal nodes, then h2 is a *better* heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*

    i.e., A1 expands at least as many nodes as A2*
  - We say that A2* is *better informed* than A1*
- The closer h to h*, the fewer extra nodes expanded

# Proof of the optimality of A*

- Assume that A* has selected G2, a goal state with a suboptimal solution, i.e., g(G2) > f*
- Proof by contradiction shows it's impossible

# Proof of the optimality of A*

- Assume that A* has selected G2, a goal state with a suboptimal solution, i.e., g(G2) > f*
- Proof by contradiction shows it's impossible
  - Choose a node n on an optimal path to G
  - Because h(n) is admissible, f* >= f(n)
  - If we choose G2 instead of n for expansion, then f(n) >= f(G2)
  - This implies f* >= f(G2)
  - G2 is a goal state: h(G2) = 0, f(G2) = g(G2).
  - Therefore f* >= g(G2)
  - Contradiction

# How to find good heuristics

*Some* options (mix-and-match):

- If $h1(n) < h2(n) <= h^*(n)$ for all n, h2 is better than (**dominates**) h1
- **Relaxing problem:** remove constraints for easier problem; use its solution cost as heuristic function
- Max of two admissible heuristics is a **Combining heuristics**: admissible heuristic, and it's better!
- Use statistical estimates to compute h; may lose admissibility
- Identify good features, then use **machine learning** to find heuristic function; also may lose admissibility