CMSC 471
Artificial Intelligence

Search

KMA Solaiman – ksolaima@umbc.edu

Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Many slides courtesy Tim Finin and Frank Ferraro

A General Searching Algorithm

Core ideas:

1. Maintain a list of frontier (fringe) nodes
   1. Nodes coming *into* the frontier have been explored
   2. Nodes going out of the frontier have not been explored
2. Iteratively select nodes from the frontier and explore unexplored nodes from the frontier
3. Stop when you reach your **goal**

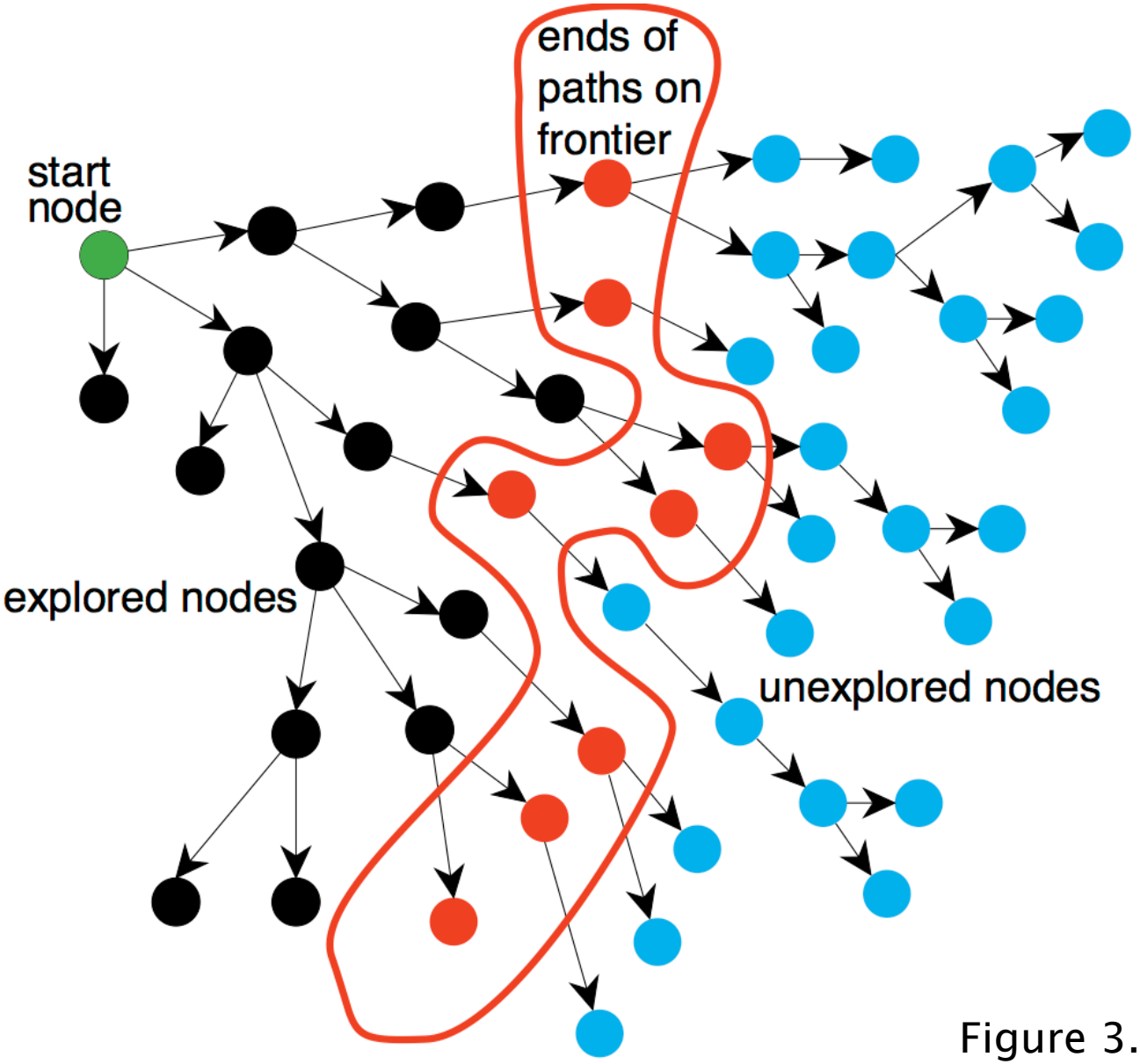


Figure 3.3

State-space 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
```

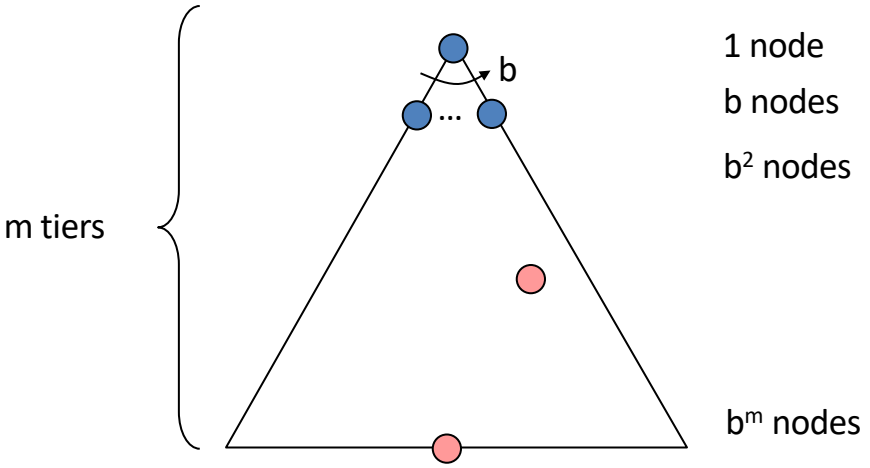
;; Note: The goal test is NOT done when nodes are generated
;; Note: This algorithm does not detect loops

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

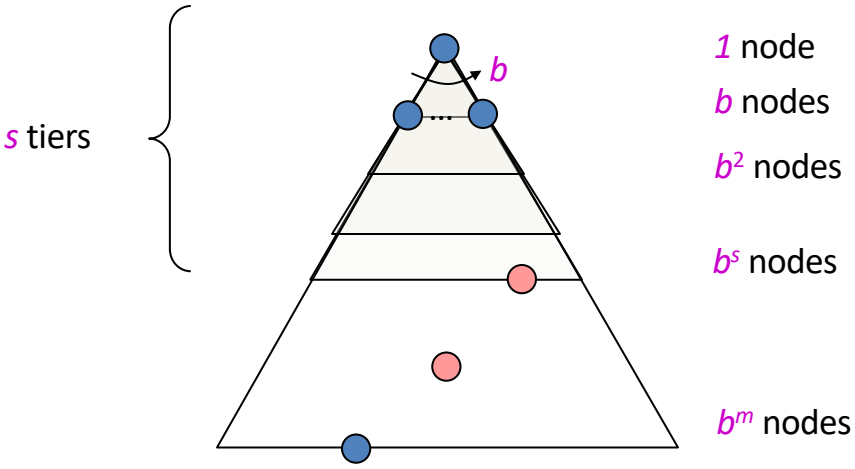
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$



Breadth-First Search (BFS) Properties

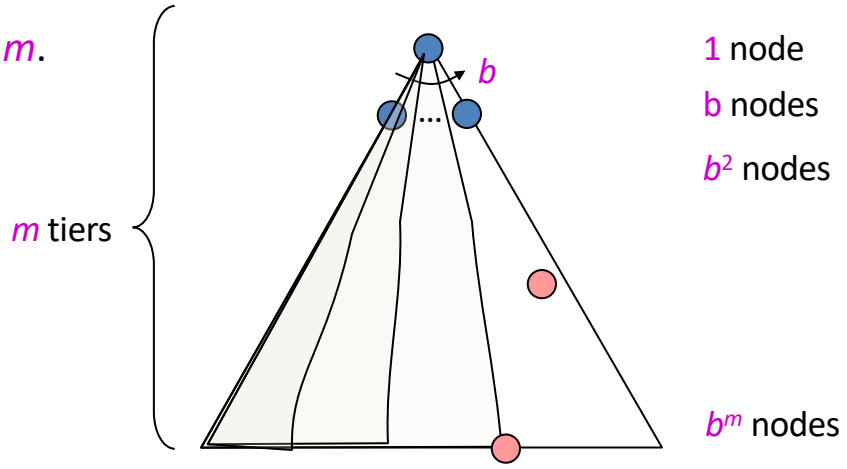
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be *s*
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - *s* must be finite if a solution exists, so yes!
- Is it optimal?
  - If costs are equal for each operator (e.g., 1)



Potential issues??

Depth-First Search (DFS) Properties

- What nodes does DFS expand?
  - Some left prefix of the tree down to depth m.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite
  - preventing cycles may help
  - **May not terminate** w/o depth bound, i.e., ending search below fixed depth D (depth-limited search)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



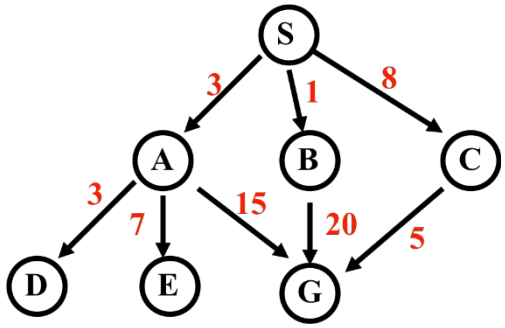
Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Iterative Deepening (DFID)

- Do DFS to depth 0, then (if no solution) DFS to depth 1, etc.
- Usually used with a tree search
- **Complete**
- **Optimal/Admissible** if all operators have unit cost, else finds shortest solution (like BFS)
- Time complexity a bit worse than BFS or DFS

  Nodes near top of search tree generated many times, but since almost all nodes are near tree bottom, worst case time complexity still exponential, $O(b^d)$

How they perform



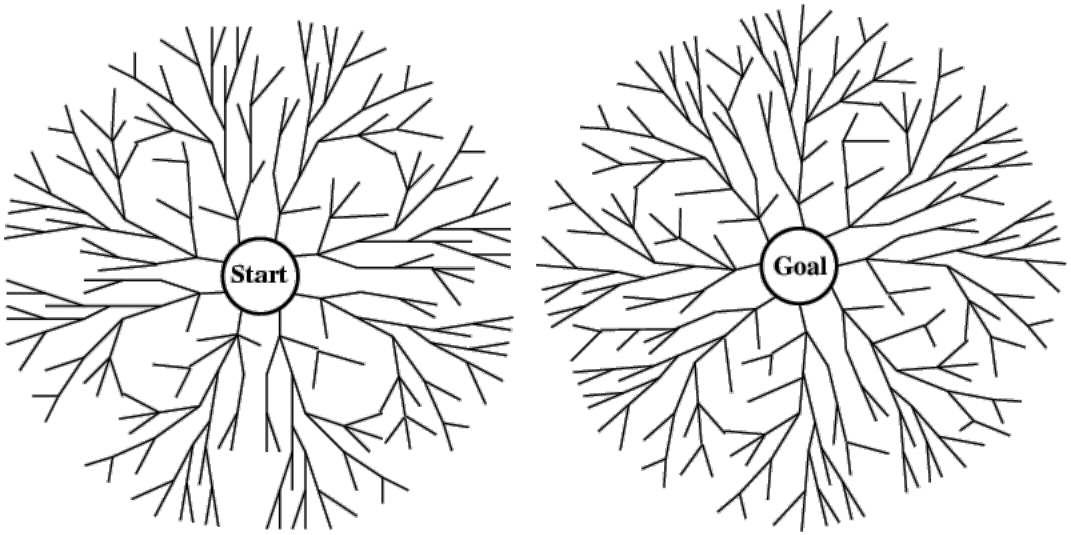
- **Depth-First Search:**
  - 4 Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search**:
  - 7 Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search**:
  - 7 Expanded nodes: S A D B C E G
  - Solution found: S C G (cost 13)

  Only uninformed search that worries about costs
- **Iterative-Deepening Search**:
  - 10 nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)

Searching Backward from Goal

- Usually a successor function is reversible
  - i.e., can generate a node's predecessors in graph
- If we know a single goal (rather than a goal's properties), we could search backward to the initial state
- It might be more efficient
  - Depends on whether the graph fans in or out

Bi-directional search



- Alternate searching from the start state toward the goal and from the goal state toward the start
- Stop when the frontiers intersect
- Works well only when there are unique start & goal states
- Requires ability to generate “predecessor” states
- Can (sometimes) lead to finding a solution more quickly

Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Informed (Heuristic) Search

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A* Search
- Memory-conserving variations of A*
- Heuristic functions

Big idea: heuristic

Merriam-Webster's Online Dictionary

Heuristic (pron. \hy*u*-ˊris-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

The Free On-line Dictionary of Computing (15Feb98)

heuristic 1. <programming> A **rule of thumb**, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Unlike algorithms, heuristics do not guarantee feasible solutions and are often used with no theoretical guarantee. 2. <algorithm> **approximation algorithm**.

From WordNet (r) 1.6

heuristic adj 1: (CS) relating to or using a heuristic rule 2: of or relating to a general formulation that serves to guide investigation [ant: algorithmic] n : a **commonsense rule** (or set of rules) intended to increase the probability of solving some problem [syn: heuristic rule, heuristic program]

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
$h(n) \leq \text{lowestcost}(n, n') + h(n')$

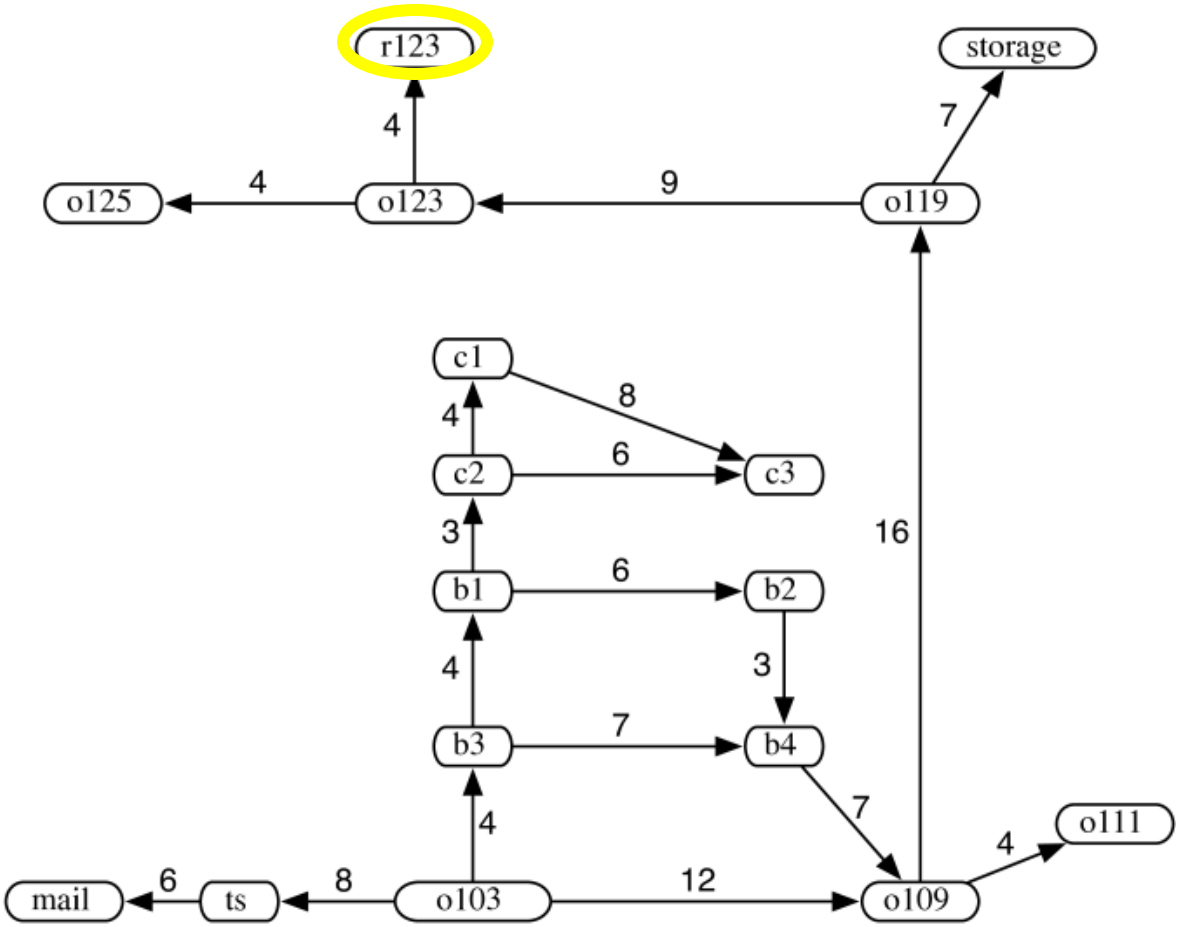
Informed methods add domain-specific information

- 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
  - Missionaries & Cannibals: # people on starting river bank
- 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

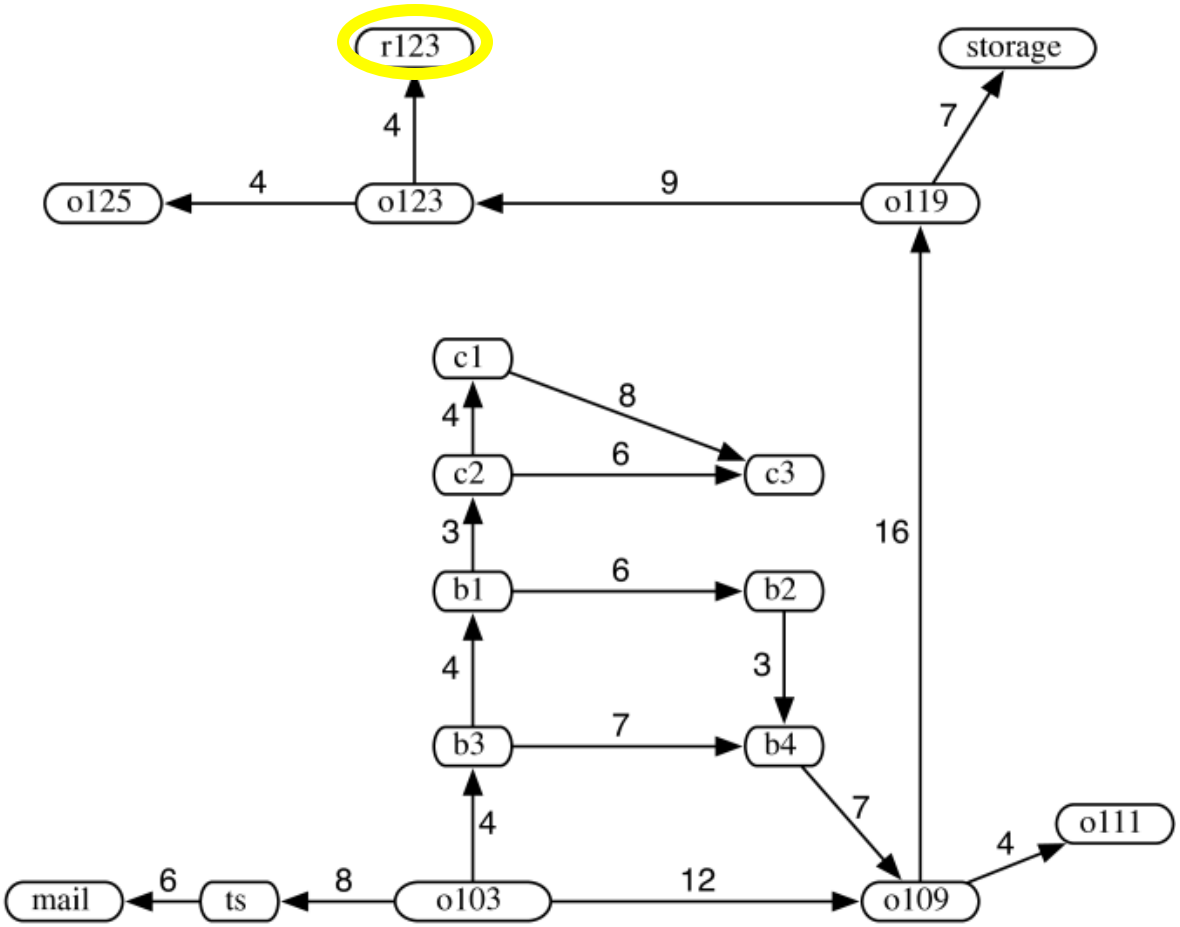
Example 3.13



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

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

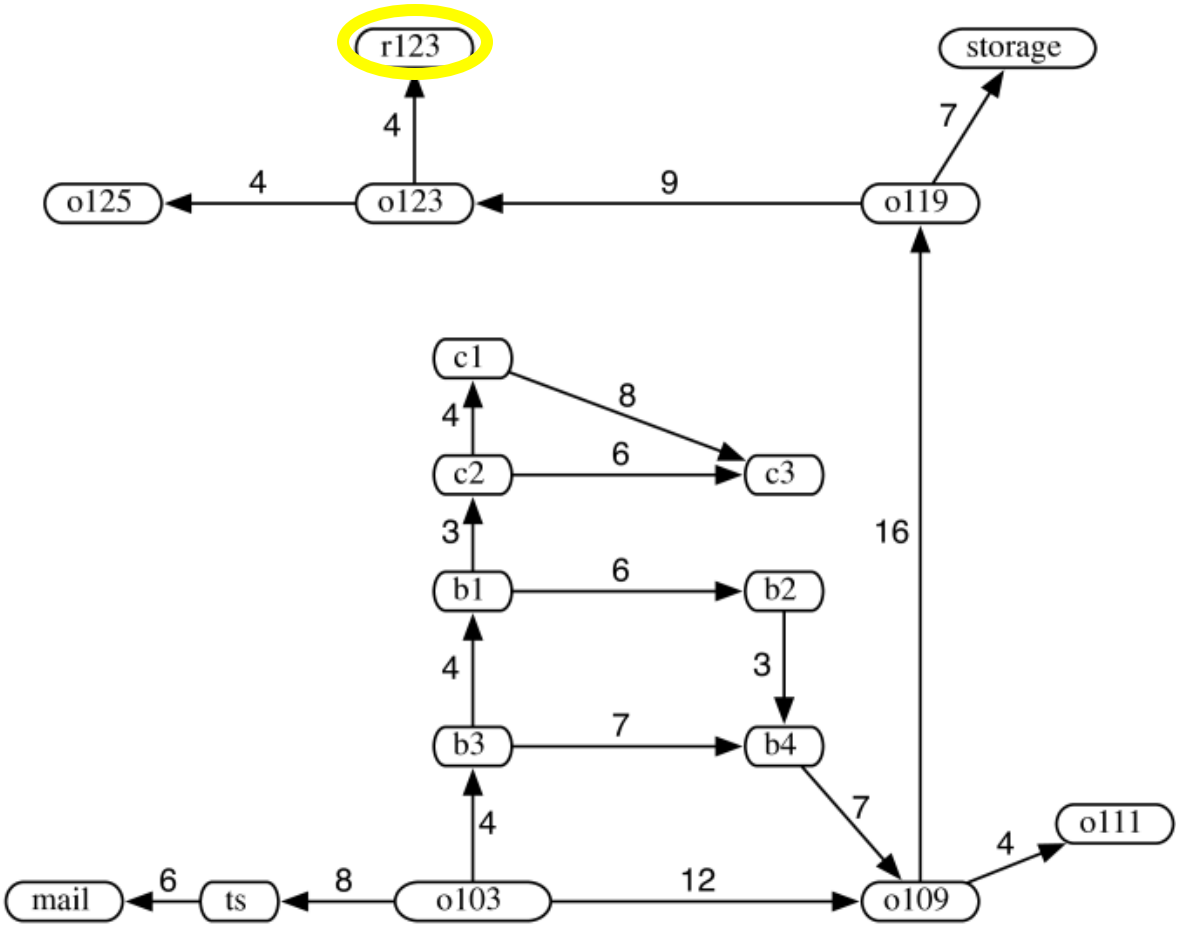
Example 3.13



Heuristic $h(n)$ for goal r123

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

Example 3.13



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

Heuristic $h(n)$ for goal r123

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

Heuristics for 8-puzzle

Misplaced Tiles Heuristic

(not including the blank)

Current State

3	2	8
4	5	6
7	1	

Goal State

1	2	3
4	5	6
7	8	

3	2	8
4	5	6
7	1	

3 tiles are not where they need to be

- Three tiles are misplaced (the 3, 8, and 1) so heuristic function evaluates to 3
- Heuristic says that it *thinks* a solution may be available in **3 or more** moves
- Very rough estimate, but easy to calculate

h = 3

Heuristics for 8-puzzle

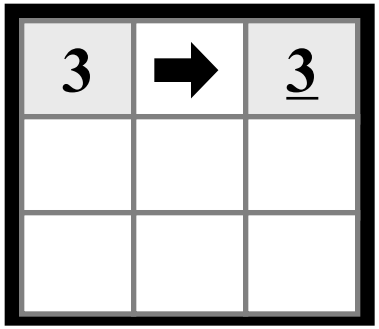
Manhattan Distance (not including the blank)

Current State

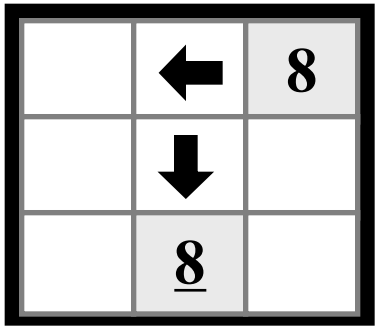
3	2	8
4	5	6
7	1	

Goal State

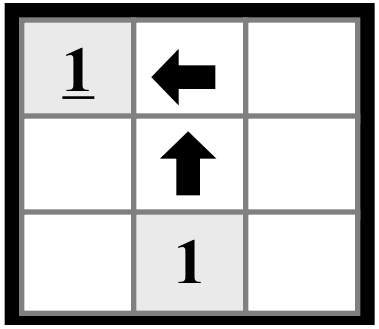
1	2	3
4	5	6
7	8	



2 spaces



3 spaces



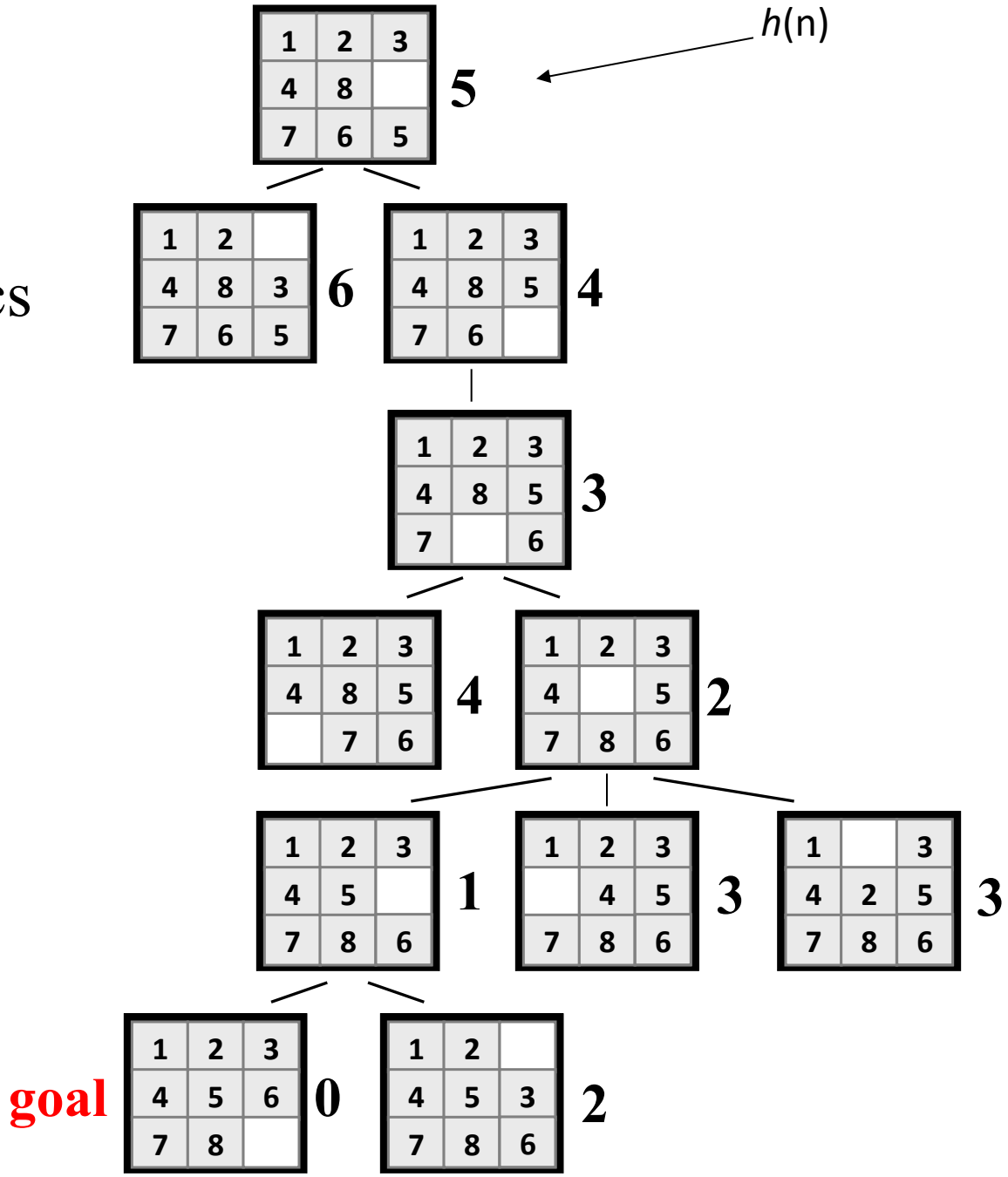
3 spaces

Total 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

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

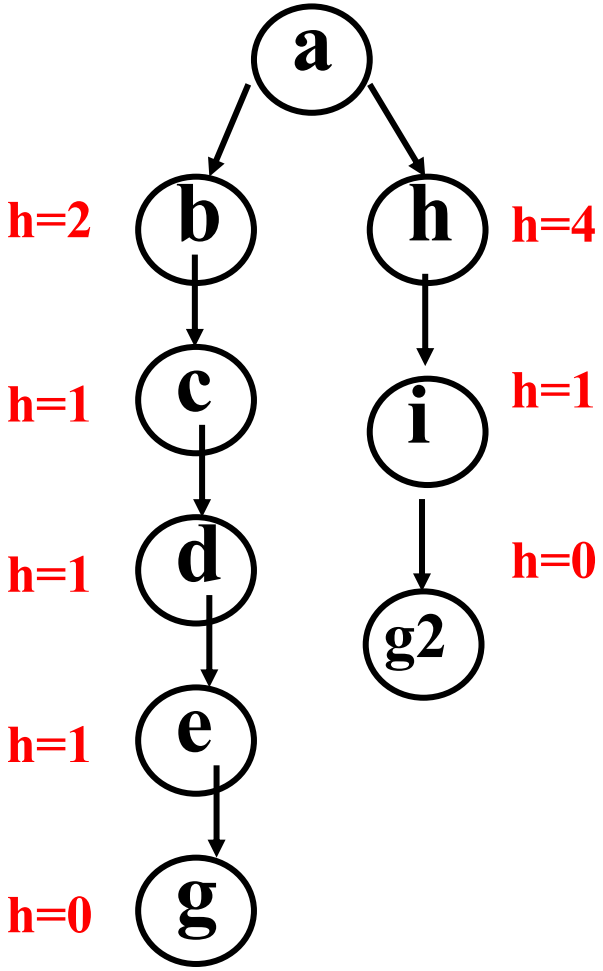
- Search algorithm that improves **depth-first search** by expanding most promising node chosen according to heuristic rule
- Order nodes on nodes list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information
- This is a generic way of referring to the class 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, as in example
  - Assume arc costs = 1, greedy search finds goal g, with solution cost of 5
  - Optimal solution is path to goal with cost 3

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?

Let's think about car trips…

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