Refinements to BFS/DBS and informed (heuristic) search strategies

CS B551
Fall 2022

Announcements

- Assignment 0 (almost) released
  - Please make sure to login to your IU GitHub account https://github.iu.edu/ before the end of the day (how about now?)
- Don’t forget the Lecture 3 Review Quiz due Sep 5 at 11:59pm
- Are you getting Canvas notifications?
  - If not, you can change them; go to Profile -> Settings -> Notifications
  - Might consider notifications for Q&A Community and slack
- No class on September 5 (Labor day)

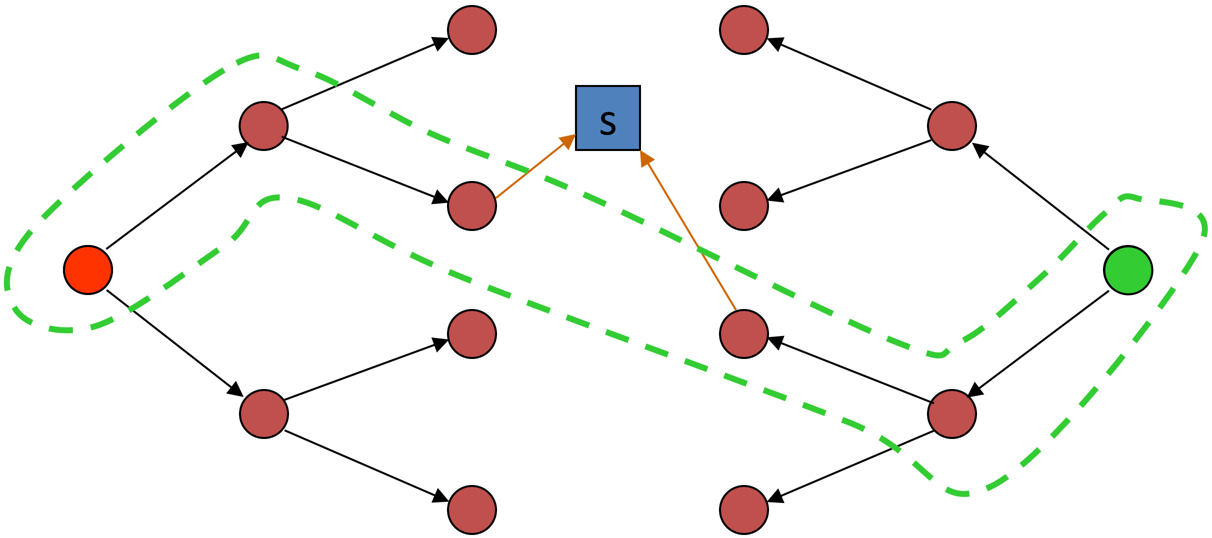
Refinements to BFS & DFS

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2



Time and space complexity is $O(b^{d/2}) << O(b^d)$
if both trees have the same branching factor b

Depth-Limited Search

For k = 0, 1, 2, … do:

Perform depth-first search with depth cutoff k

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)

Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
  - Store all states associated with **generated (expanded)** nodes in VISITED
  - If the state of a new node is in VISITED, then discard the node

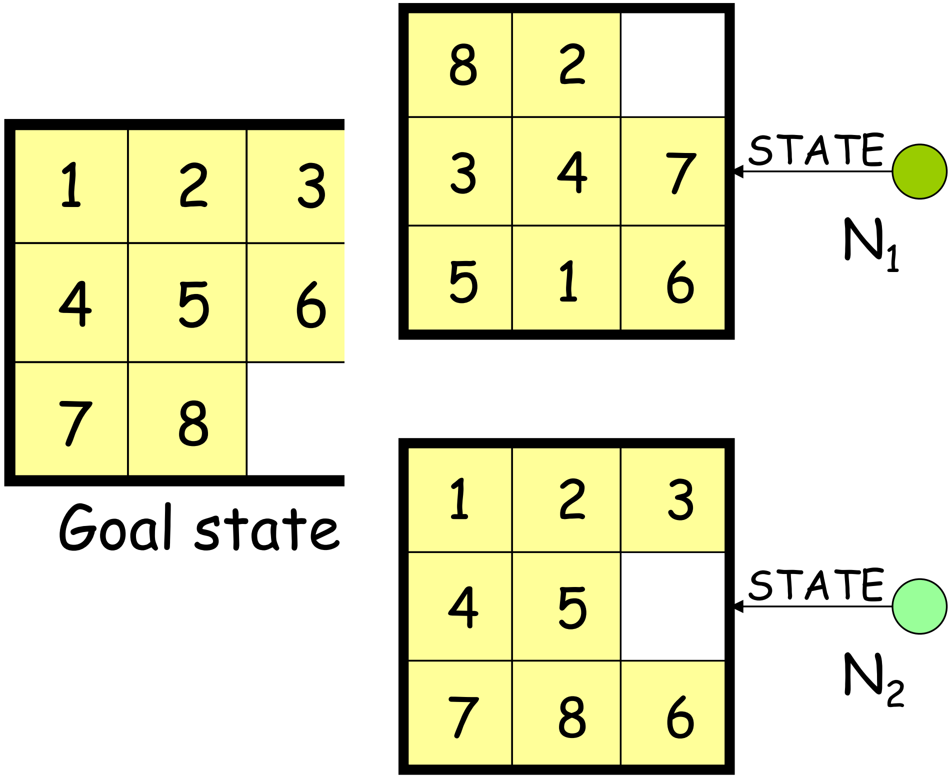
Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
  - Store all states associated with **generated (expanded)** nodes in VISITED
  - If the state of a new node is in VISITED, then discard the node

Implemented as hash-table
or as explicit data structure with flags

Heuristic search

Heuristic vs blind search



For a blind strategy, N_1 and N_2 are just two nodes (at some position in the search tree)

For a heuristic strategy, N_2 seems more promising than N_1 (fewer misplaced tiles)

Best-First Search

- Explores most "promising" state from FRINGE first
  - Typically implemented with a priority queue
  - Requires *evaluation function* $f(s) \geq 0$ that estimates the **"cost" from initial state, through s, to a goal state**
- Typical choices of f(s):
  - $f(s) = h(s)$, where h estimates the cost of reaching **a goal from s** (greedy best first search)
  - $f(s) = g(s) + h(s)$, where g is **cost of path from start state to s** and h estimates the **cost of reaching a goal from s**

Adding to our abstraction

1. Set of states S
2. Initial state s_0
3. A function SUCC: $S \rightarrow 2^S$ that encodes possible transitions of the system
4. Set of goal states
5. A cost function c() that calculates how "expensive" a given set of moves is
6. A *heuristic function* h() that estimates how "promising" a given state is

Heuristic function h(s) $\geq$ 0

- Estimates the cost to go from state s to a goal state
  - The better the estimate, the more efficient the search
  - BUT we want to be able to compute it efficiently
  - Typically there are many possibilities, each with trade-offs

- How would you define h(s) for the 8-puzzle?

5		8
4	2	1
7	3	6

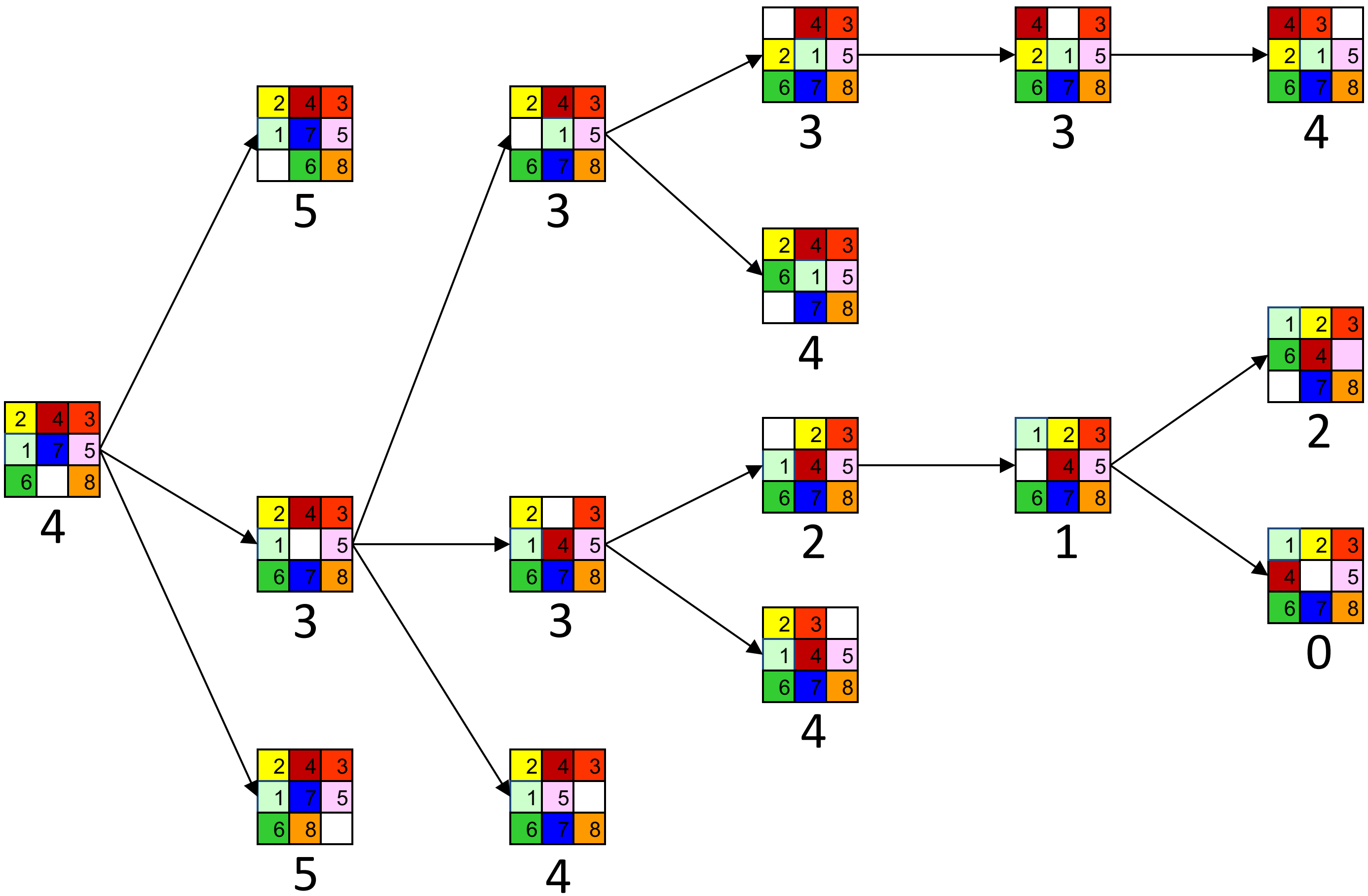
STATE(s)

1	2	3
4	5	6
7	8	

Goal state

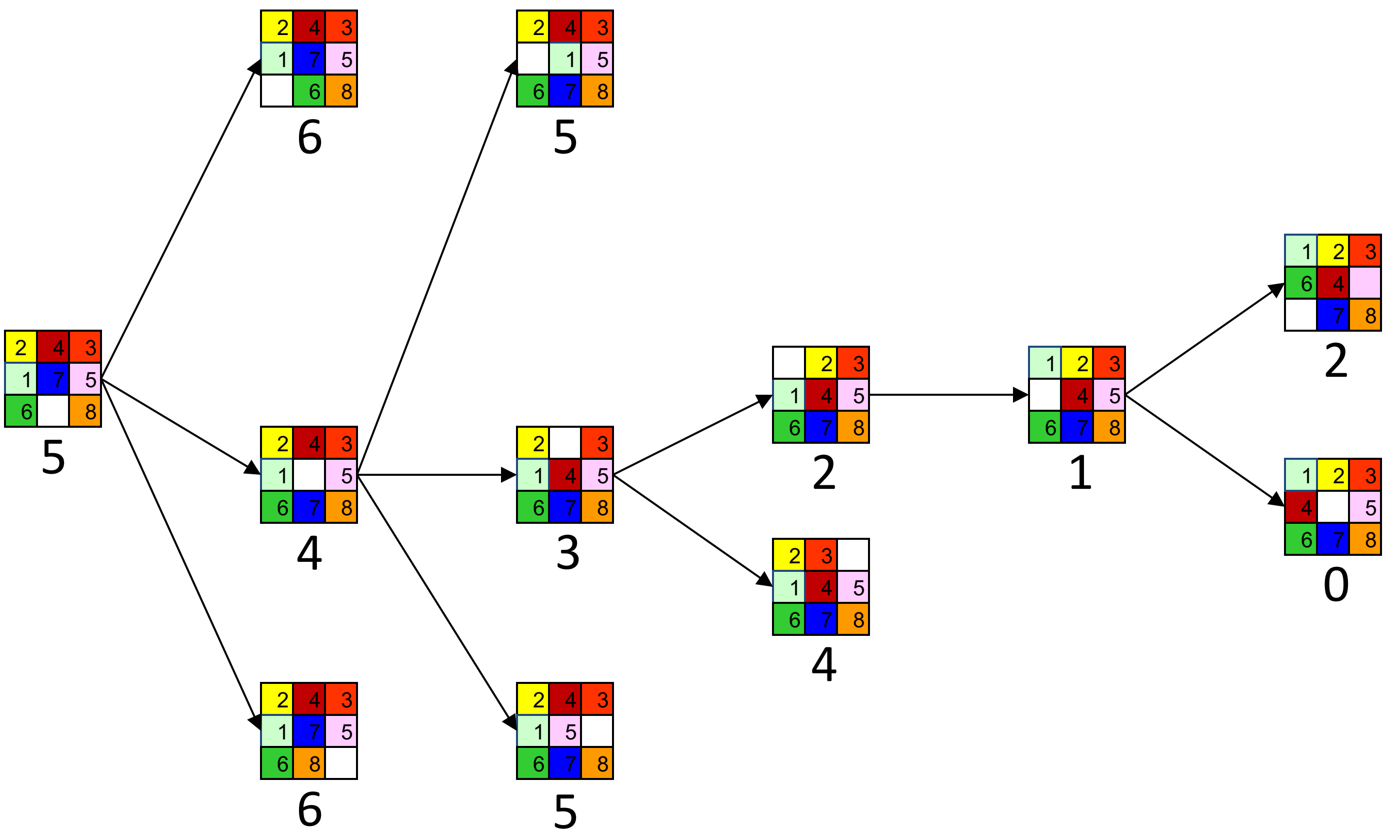
8-Puzzle

f(s) = h(s) = number of misplaced numbered tiles

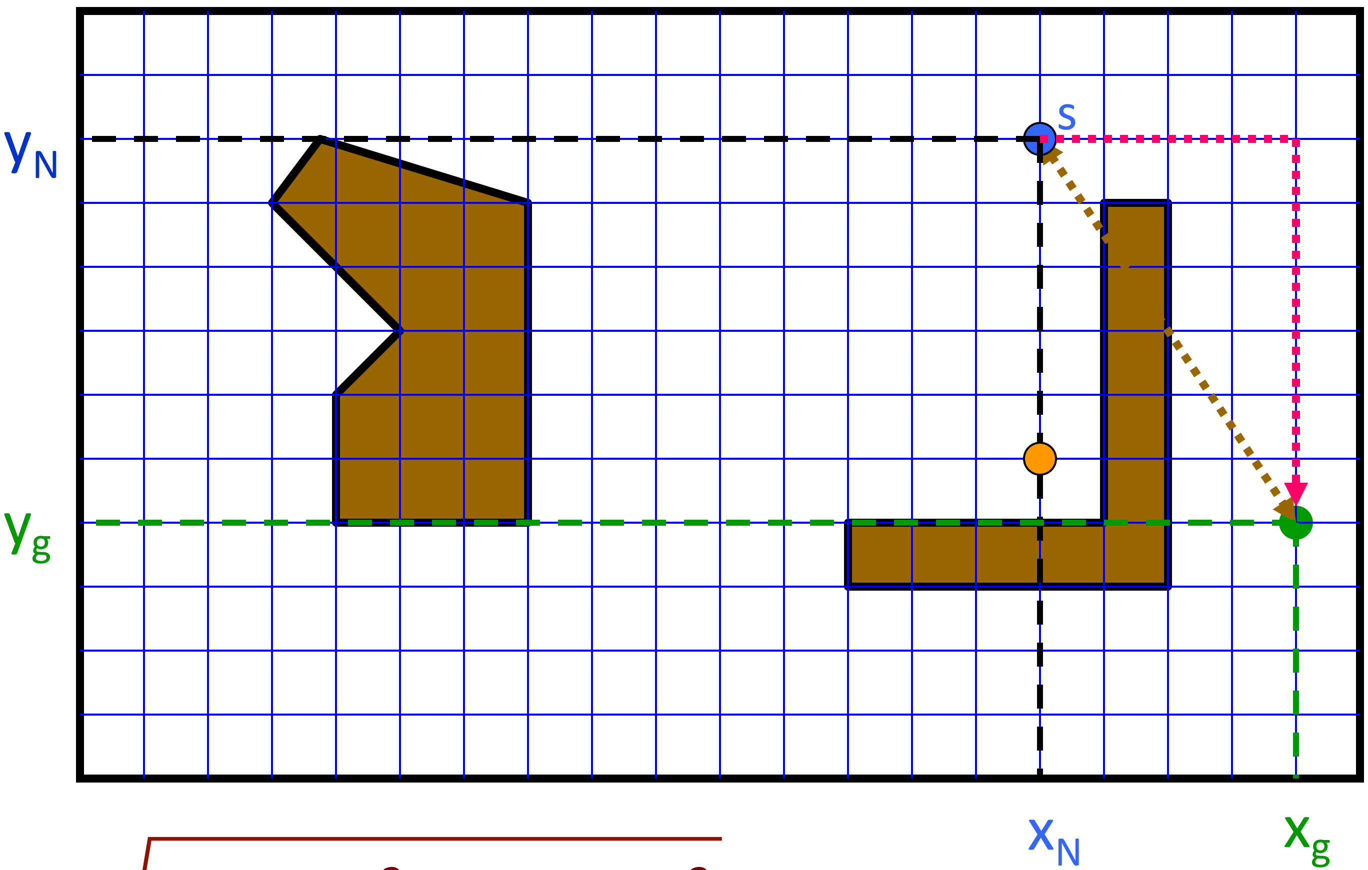


8-Puzzle

f(N) = h(N) = Σ distances of numbered tiles to their goals



Robot Navigation



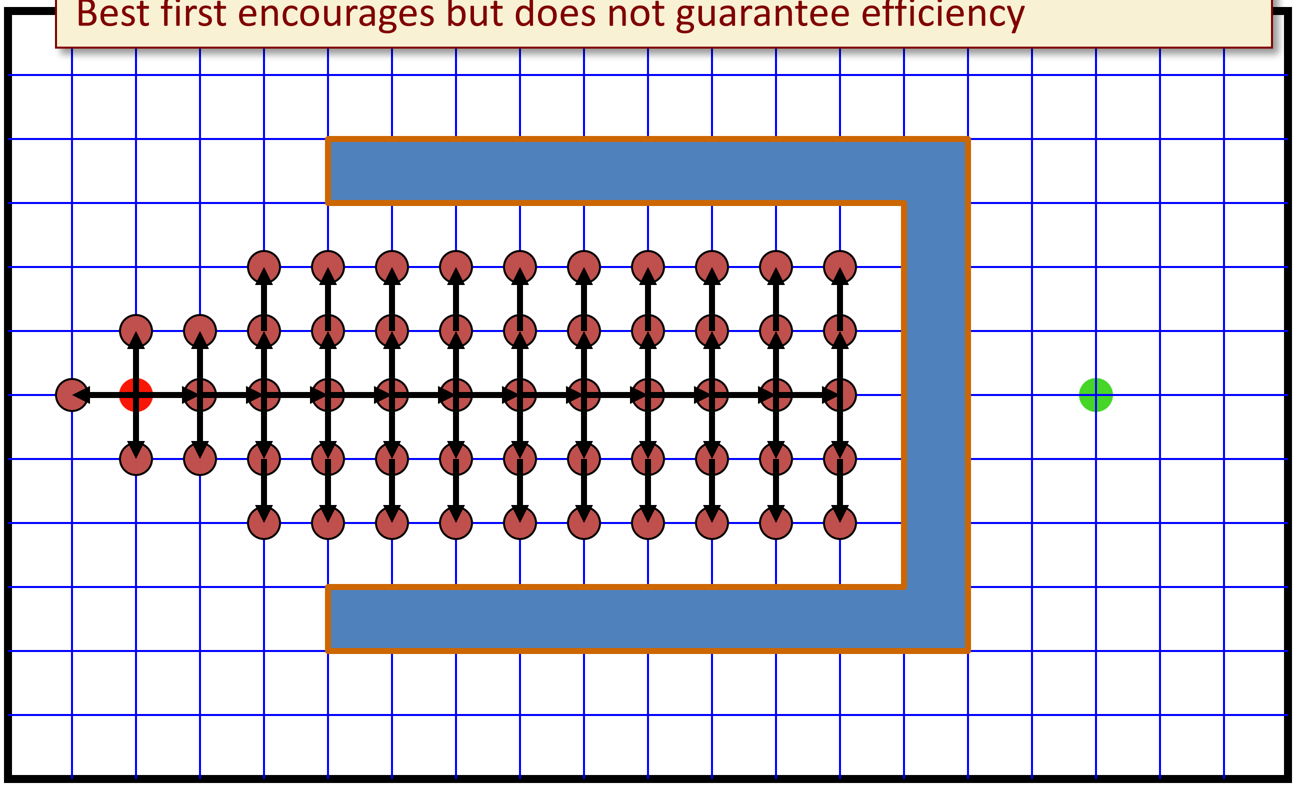
$$h_1(s) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$ (L$_2$ or Euclidean distance)

$$h_2(s) = |x_N - x_g| + |y_N - y_g|$$ (L$_1$ or Manhattan distance)

What can go wrong?

Local-minimum problem:

Best first encourages but does not guarantee efficiency

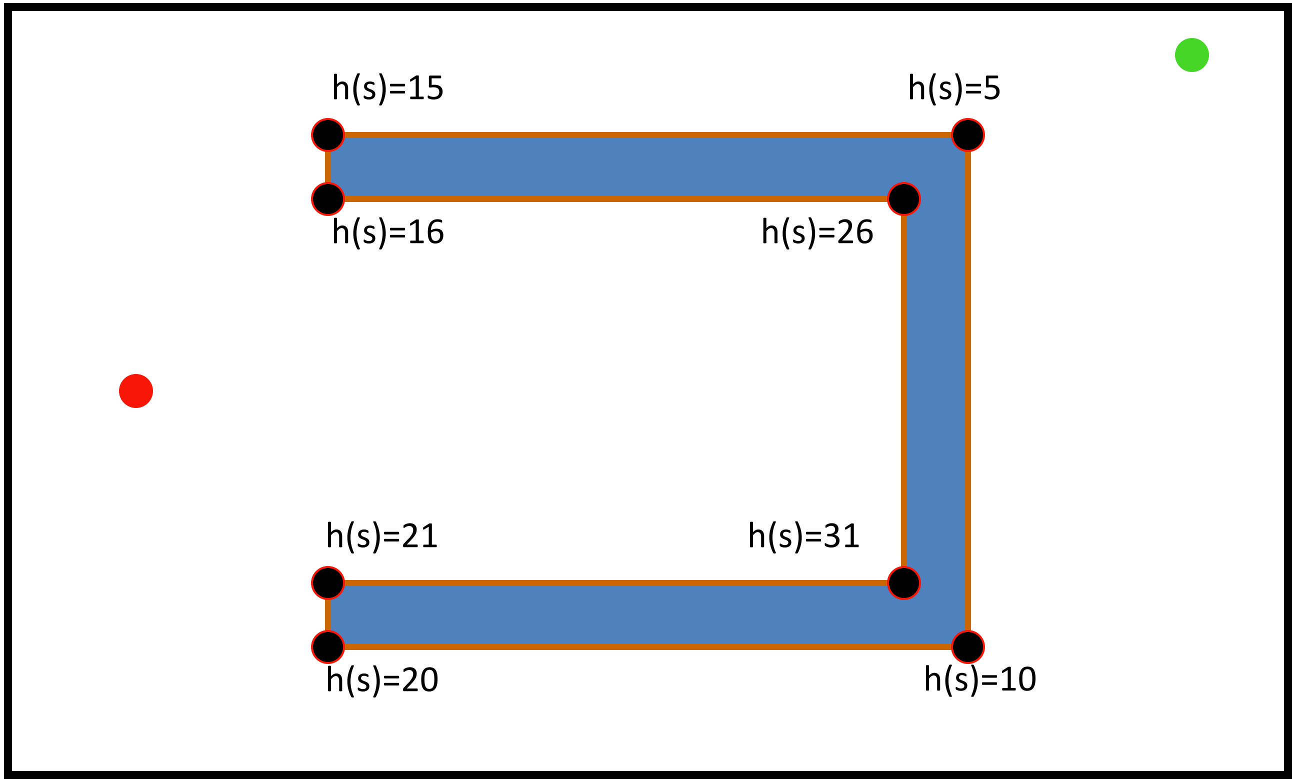


h(s) = straight distance to the goal

Another idea: Pre-computation

E.g. assume environment is mostly static, use past best routes from some waypoints to estimate distances to goal

h(s) = pre-computed distance from waypoint to goal



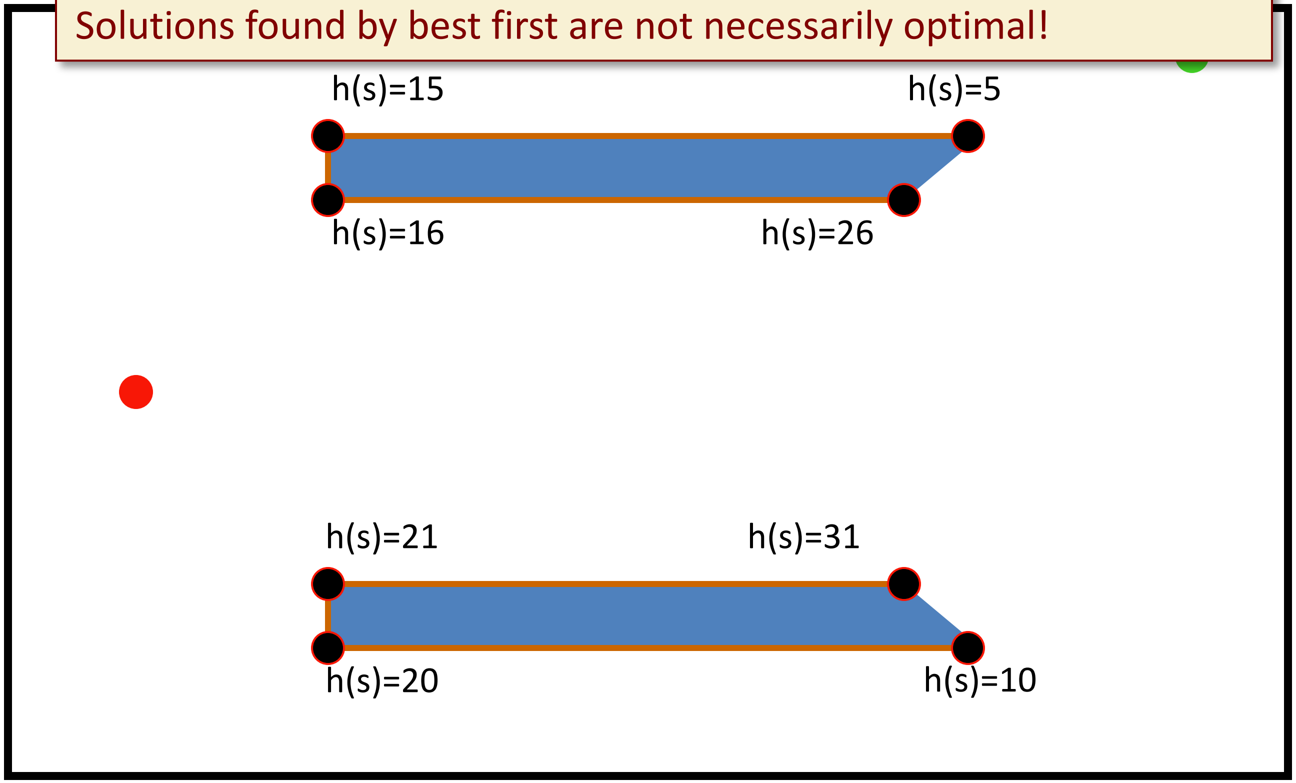
Another idea: Pre-computation

E.g. assume environment is mostly static, use past best routes from some waypoints to estimate distances to goal

h(s) = pre-computed distance from waypoint to goal

Optimality problem:

Solutions found by best first are not necessarily optimal!



Admissible Heuristic

- An heuristic h(s) is **admissible** if for any state s,

$$\mathbf{0 \leq h(s) \leq h^{*}(s)}$$

  where h*(s) is the optimal cost from s to a goal.

- In other words, an admissible heuristic **never overestimates the cost to the goal**
  - We'll need to design h(s) so that it's always less than h*(s), even though we don't know h*(s)!

Which of these are admissible?

5		8
4	2	1
7	3	6

STATE(s)

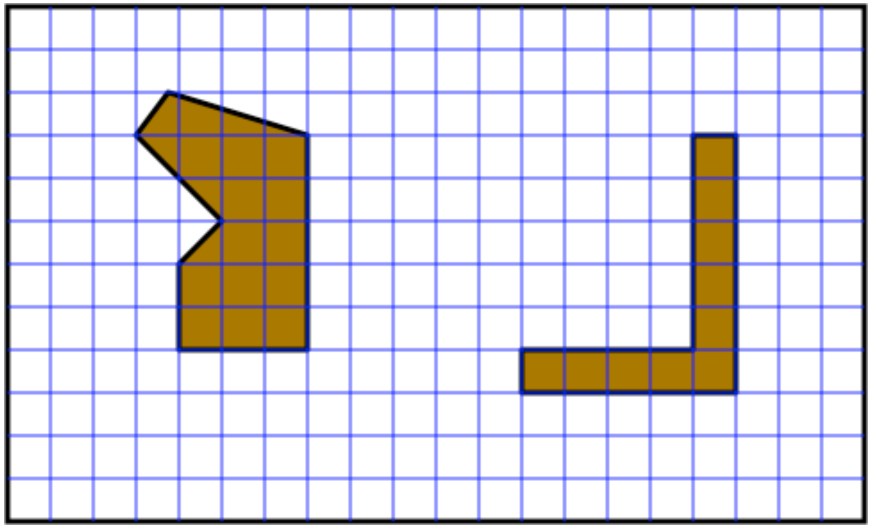
1	2	3
4	5	6
7	8	

Goal state

- $h_1(N)$ = number of misplaced numbered tiles = 6
- $h_2(N)$ = sum of Manhattan distances of tiles to goal positions

  $= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$

Which of these are admissible?



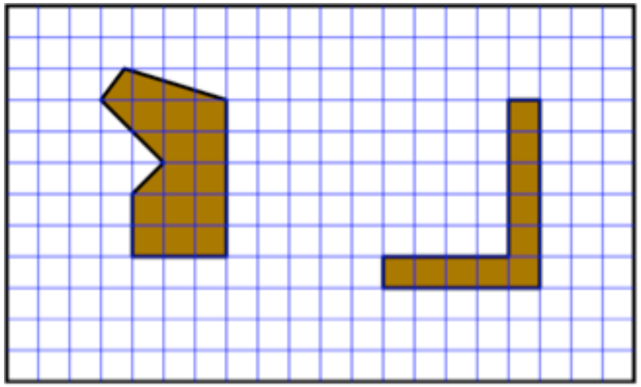
$$h_1(s) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$

$$h_2(s) = |x_N - x_g| + |y_N - y_g|$$

Assume that the agent can move diagonally

How to create an admissible h?

- Often a challenge! A good h is admissible, fast to compute, and still a good estimate
- Admissible heuristics are often optimal solutions to simplified (relaxed) problems (with constraints ignored). E.g. for robot navigation:
  - Manhattan distance ignores obstacles
  - Euclidean distance ignores obstacles and constraint that robot moves on a grid
- Much more on this later!



$$h_1(s) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$$

$$h_2(s) = |x_N - x_g| + |y_N - y_g|$$

A* Search

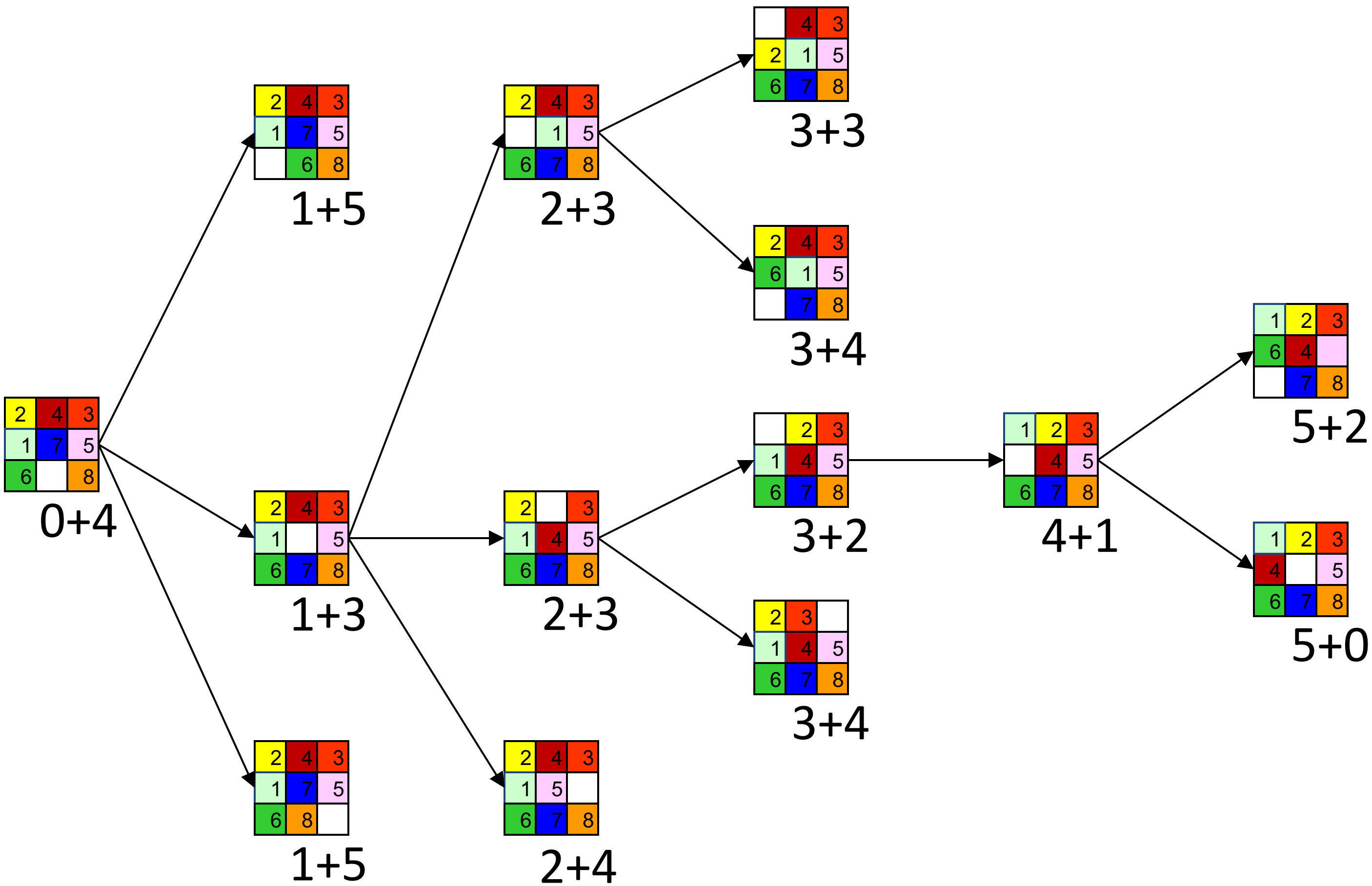
- Best First Search with f(s) = g(s) + h(s), where:
  - g(s) = cost of best path found so far to s
  - h(s) = **admissible** heuristic function

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node, FRINGE)
3. Repeat:
4. If empty(FRINGE) then return failure
5. s ← REMOVE(FRINGE)
6. If GOAL?(s) then return s and/or path
7. For every state s' in SUCC(s):
8. INSERT(s', FRINGE)

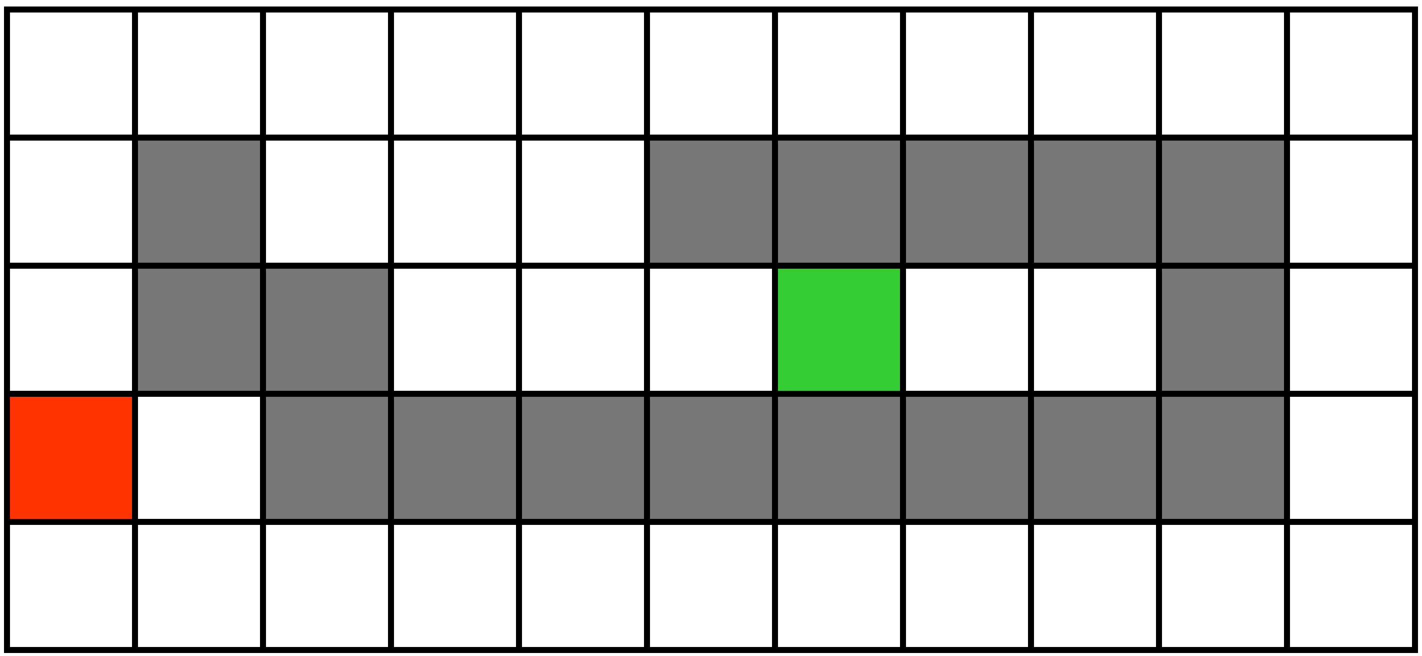
8-Puzzle

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

with $h(s)$ = number of misplaced numbered tiles



Robot Navigation



Robot Navigation

f(s) = h(s), with h(s) = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

f(s) = h(s), with h(s) = Manhattan distance to the goal
(not A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Robot Navigation

$f(s) = g(s)+h(s)$, with $h(s)$ = Manhattan distance to goal (A*)

8+3	7+4	6+5	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

1. Is A* complete?

2. Is A* optimal?

3. What is the running time of A*?

4. What are the memory requirements of A*?

Answer in the next class
(and in the book, ch. 3.5)

Next class

- Finish heuristic (informed) search
- Local search