浙江大学伊利诺伊大学厄巴纳香槟校区联合学院
Zhejiang University-University of Illinois at Urbana Champaign Institute

ECE 448: Artificial Intelligence

Lecture 5: Search informed by lookahead heuristics: Greedy, Admissible A*, Consistent A*

Prof. Hongwei Wang hongweiwang@intl.zju.edu.cn

Prof. Mark Hasegawa-Johnson jhasegaw@illinois.edu

Spring 2023

Outline

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

Review: DFS and BFS

- **Depth-first search**
  - LIFO: expand the deepest node (farthest from START)
  - Pro: reach the end of the path as quickly as possible (space is $O\{bm\}$). Good if there are many paths to goal.
  - Con: not optimal, or even complete. Time is $O\{b^m\}$.
- **Breadth-first search**
  - FIFO: expand the shallowest node (closest to START)
  - Pro: complete and optimal. Time is $O\{b^d\}$
  - Con: no path is found until the best path is found. Space is $O\{b^d\}$.

Outline

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

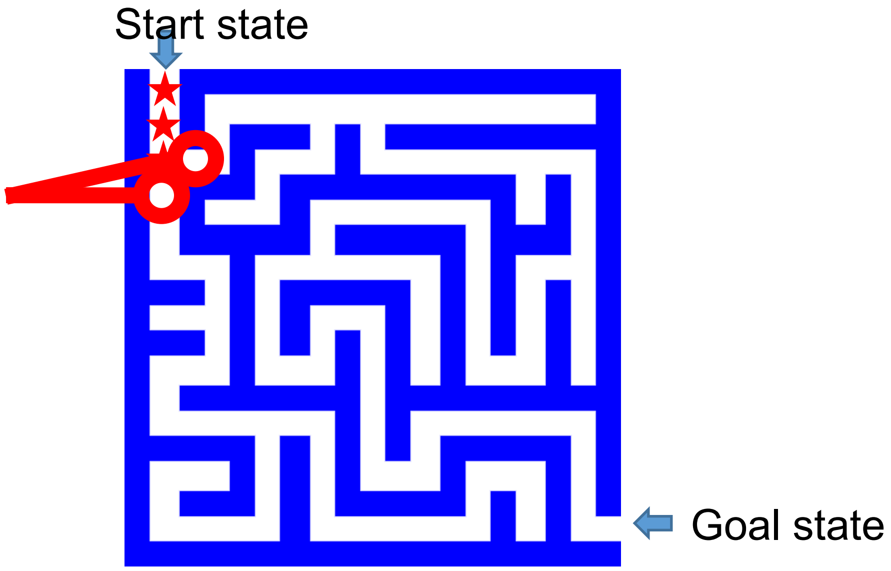
Why don’t we just measure…

Instead of FARTHEST FROM START (DFS):

why not choose the node that’s CLOSEST TO GOAL?

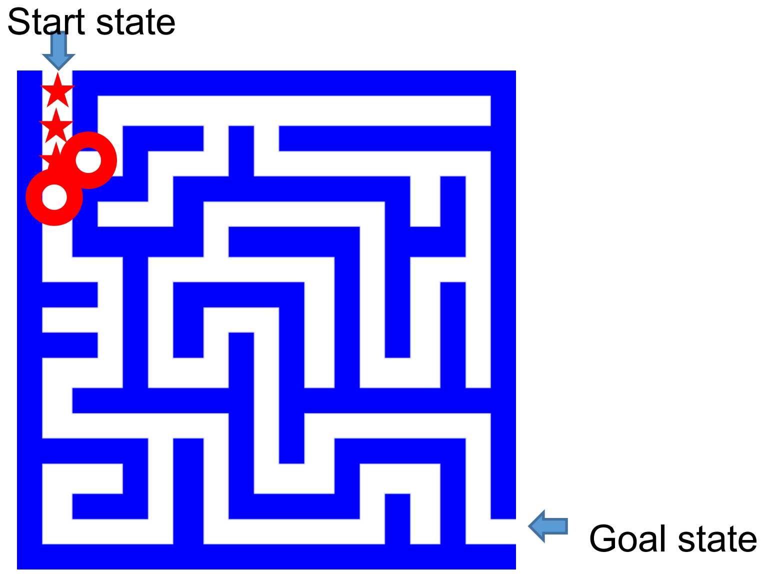
Why not choose the node CLOSEST TO GOAL?

- Answer: because we don't know which node that is!!
- Example: which of these two is closest to goal?



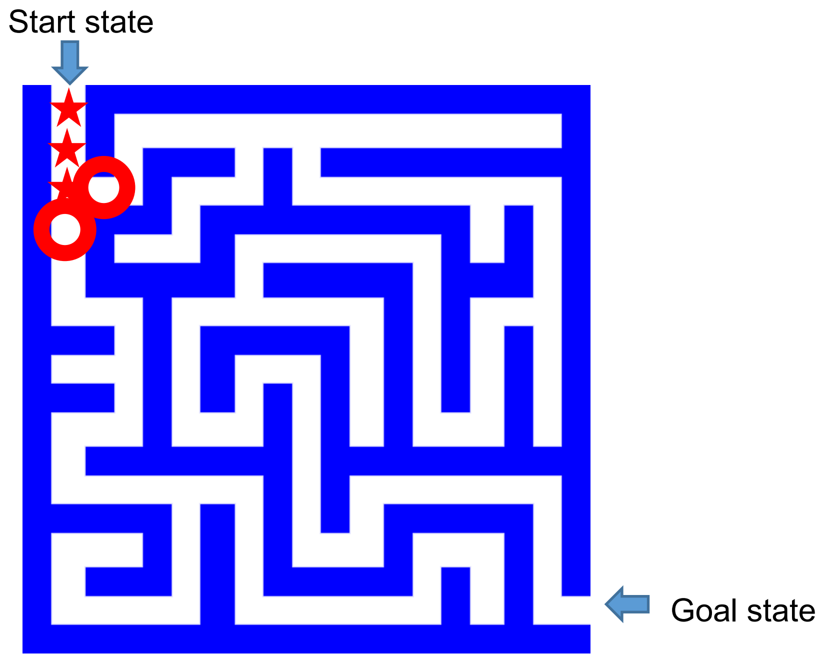
We don’t know which state is closest to goal

- Finding the shortest path is the whole point of the search
- If we already knew which state was closest to goal, there would be no reason to do the search
- Figuring out which one is closest, in general, is a complexity $O\{b^d\}$ problem.



Search heuristics: estimates of distance-to-goal

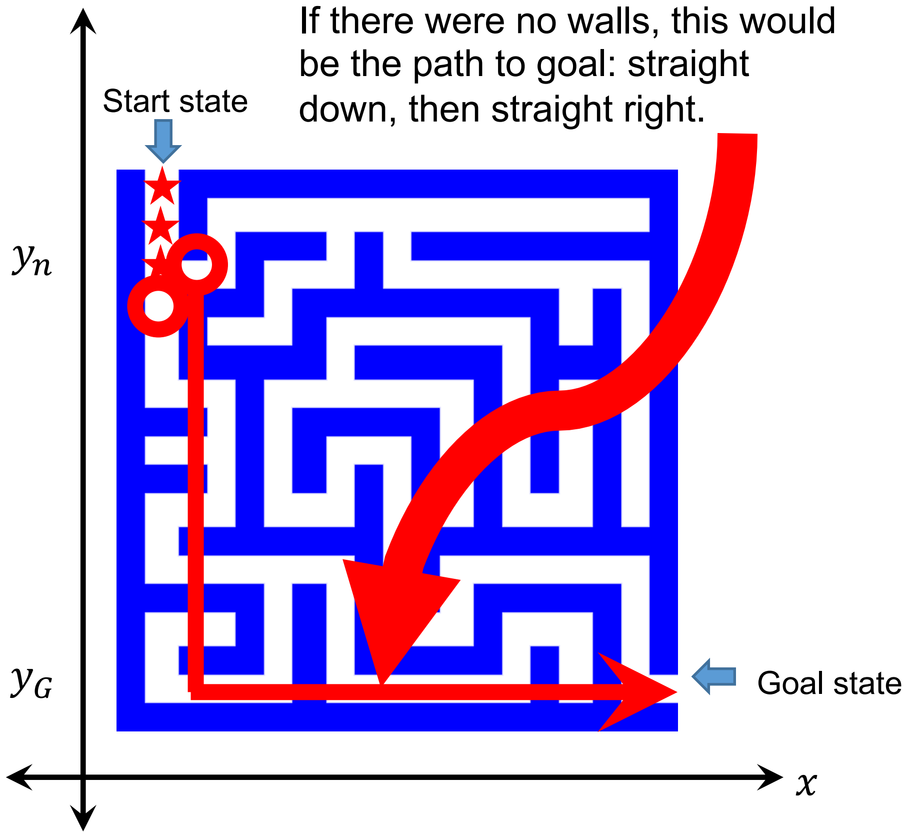
- Often, even if we don't know the distance to the goal, we can estimate it.
- This estimate is called a heuristic.
- A heuristic is useful if:
  1. **Accurate**: $h(n) \approx d(n)$, where $h(n)$ is the heuristic estimate, and $d(n)$ is the true distance to the goal
  2. **Cheap:** It can be computed in complexity less than $O\{b^d\}$



Example heuristic: Manhattan distance

If there were no walls in the maze, then the number of steps from position (x_n, y_n) to the goal position (x_G, y_G) would be

$$h(n) = |x_n - x_G| + |y_n - y_G|$$


If there were no walls, this would be the path to goal: straight down, then straight right.
Start state
y_n
y_G
Goal state
x

Outline

1. Search heuristics
2. <u>Greedy best-first search: minimum h(n)</u>
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

Greedy Best-First Search

Instead of FARTHEST FROM START (DFS):

why not choose the node whose

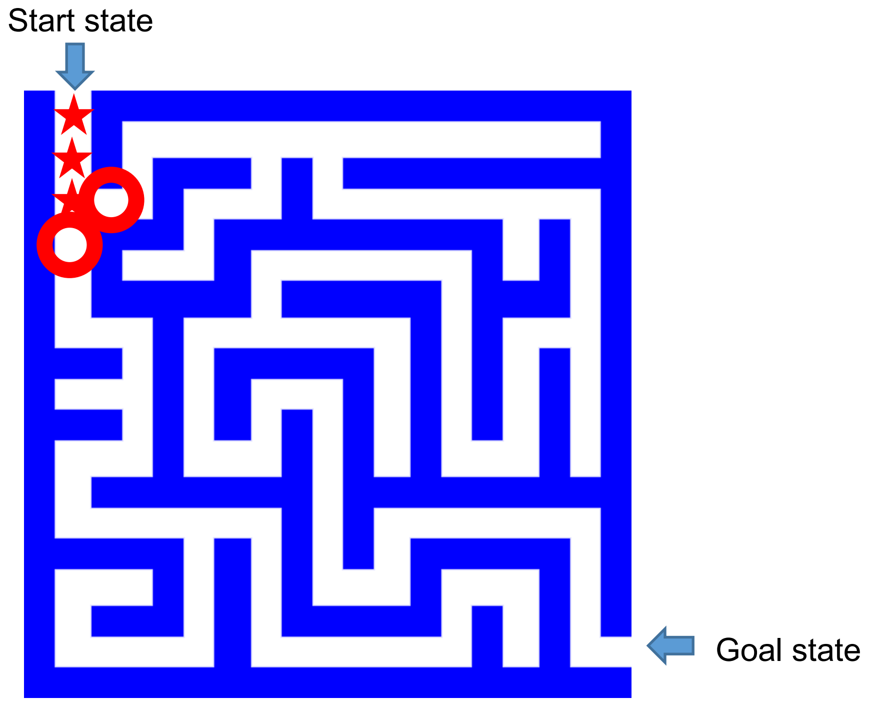
HEURISTIC ESTIMATE

indicates that it might be

CLOSEST TO GOAL?

Greedy Search Example

According to the Manhattan distance heuristic, these two nodes are equally far from the goal, so we have to choose one at random.

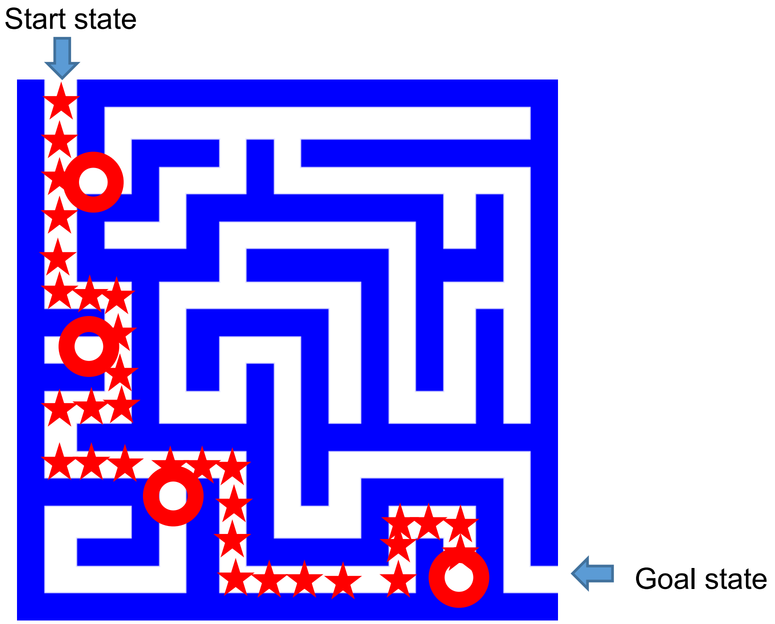


Greedy Search Example

If our random choice goes badly, we might end up very far from the goal.

★ = states in the explored set

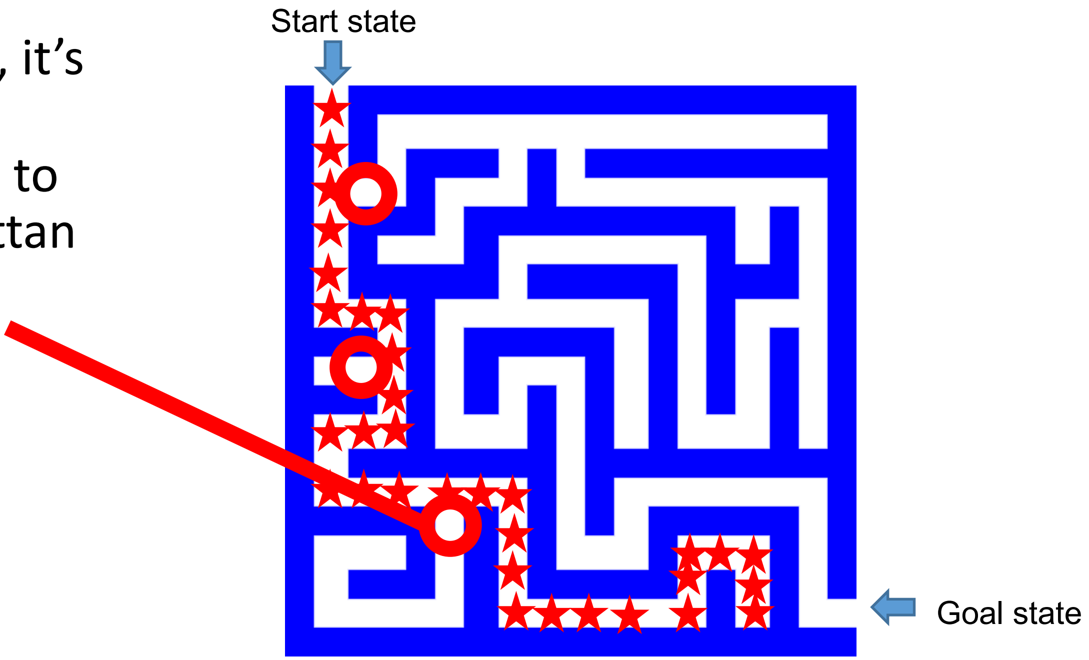
● = states on the frontier



The problem with Greedy Search

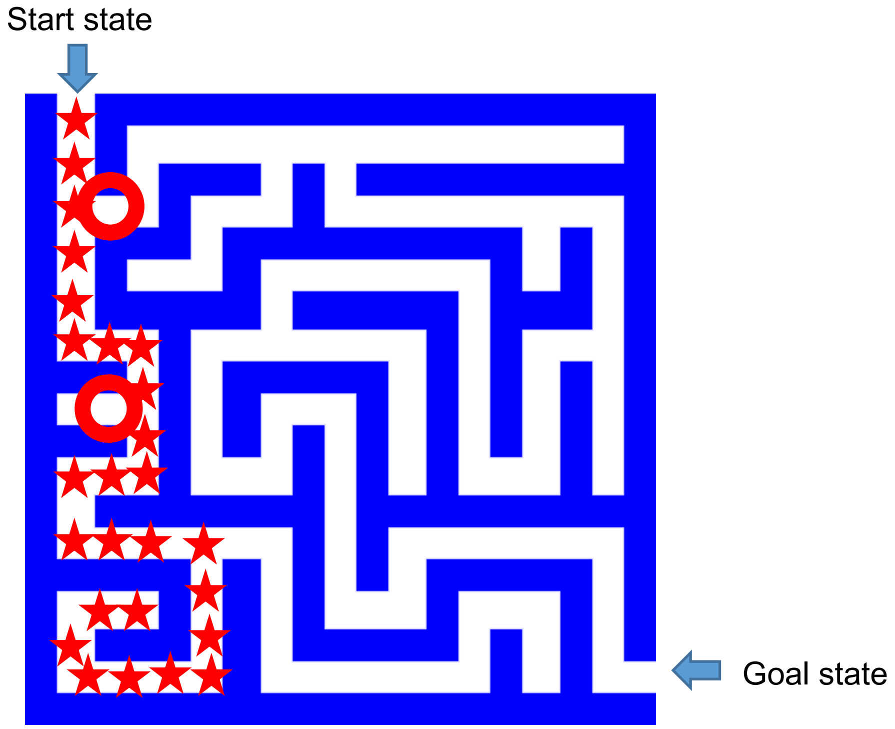
ZJUI

Having gone down a bad path, it's very hard to recover, because now, the frontier node closest to goal (according to the Manhattan distance heuristic) is this one:



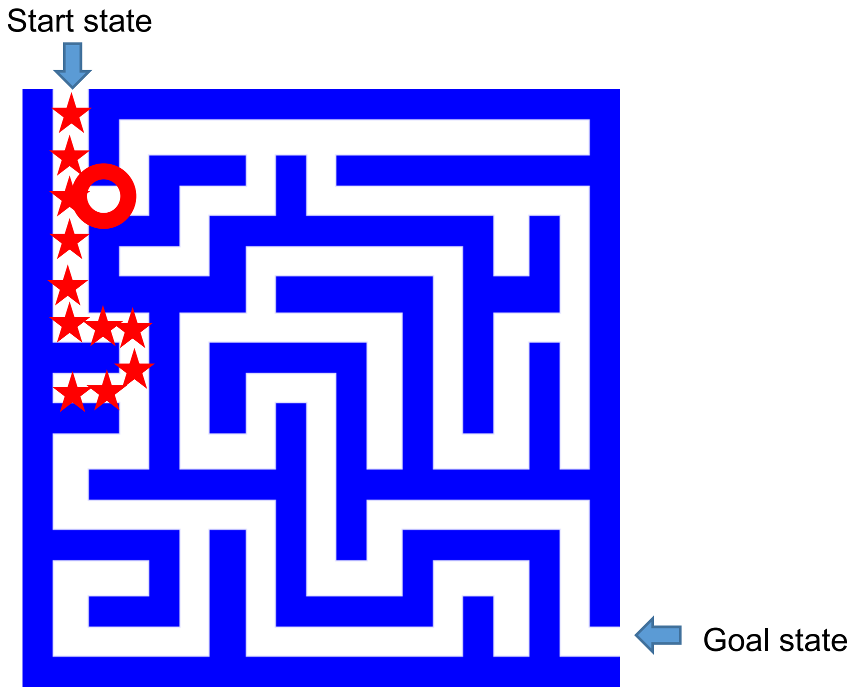
The problem with Greedy Search

That's not a useful path…



The problem with Greedy Search

Neither is that one…



Greedy Best-First Search

What went wrong?

Outline

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

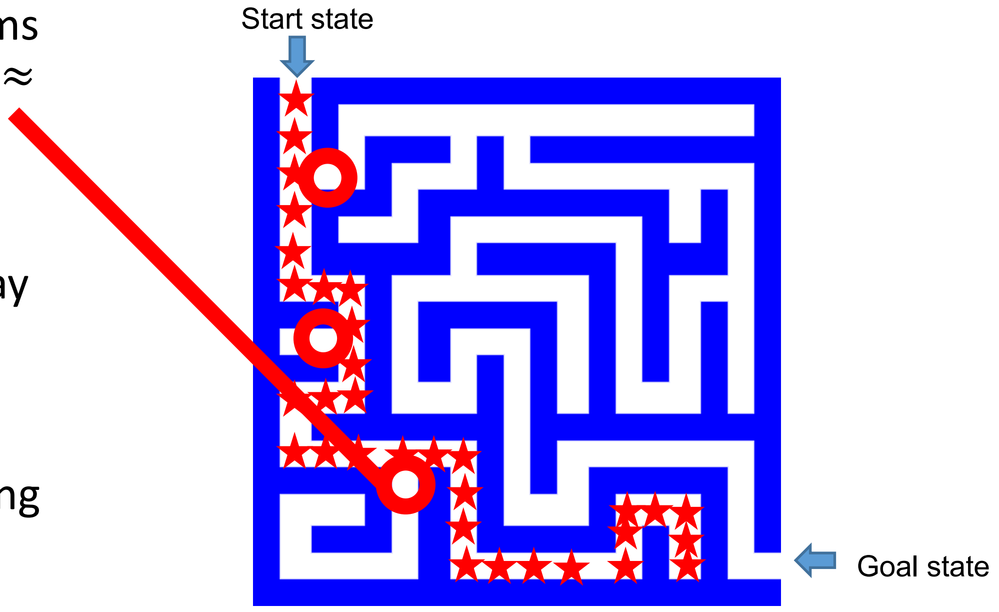
The problem with Greedy Search

Among nodes on the frontier, this one seems closest to goal (smallest $h(n)$, where $h(n) \approx d(n)$).

But it's also farthest from the start. Let's say $g(n)$ = total path cost so far.

So the total distance from start to goal, going through node n, is

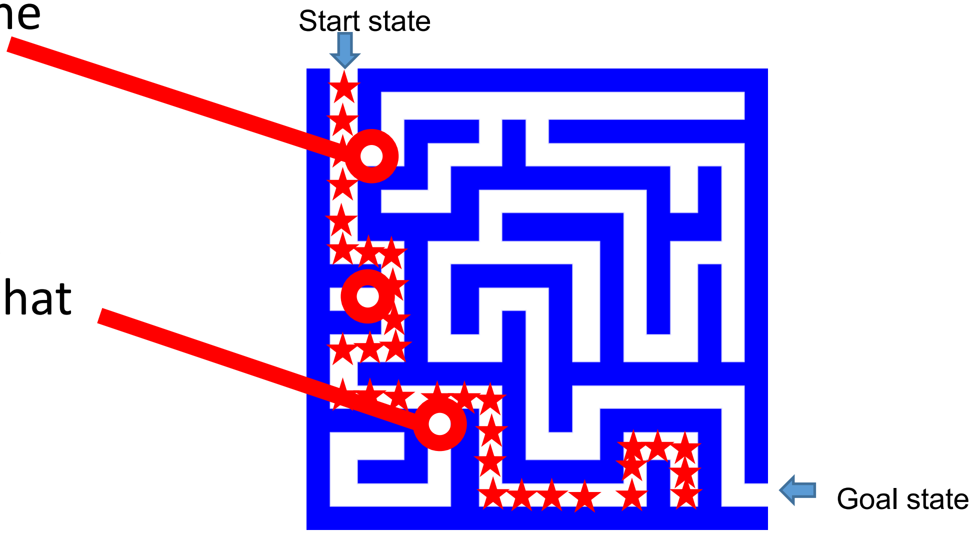
$$c(n) = g(n) + d(n) \approx g(n) + h(n)$$



The problem with Greedy Search

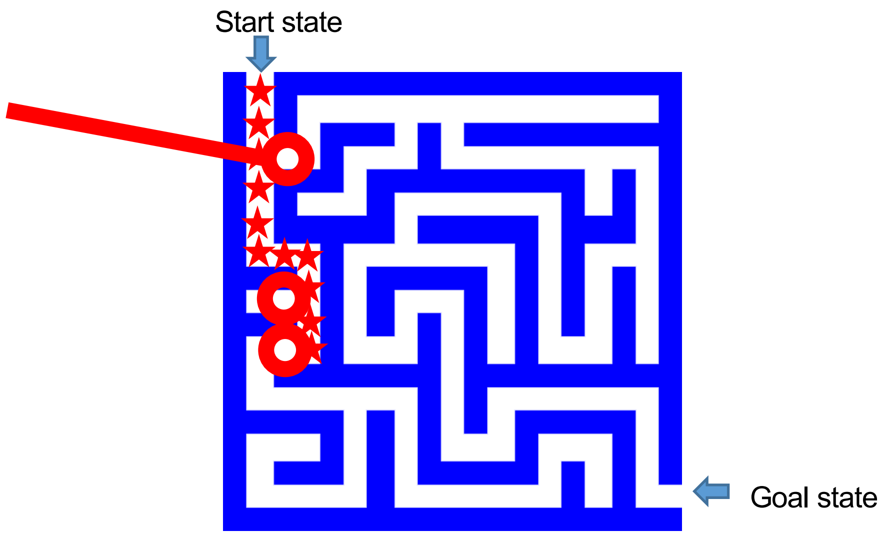
Of these three nodes, this one has the smallest $g(n) + h(n)$.

So if we want to find the lowest-cost path, then it would be better to try that node, instead of this one.



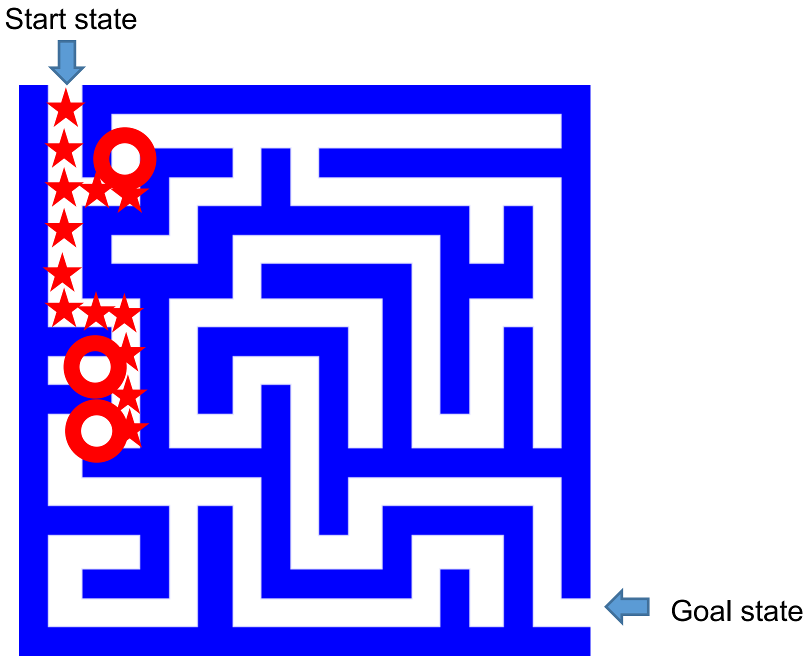
Smart Greedy Search

In fact, let's back up. Already, at this point in the search, this node has the smallest $g(n) + h(n)$.



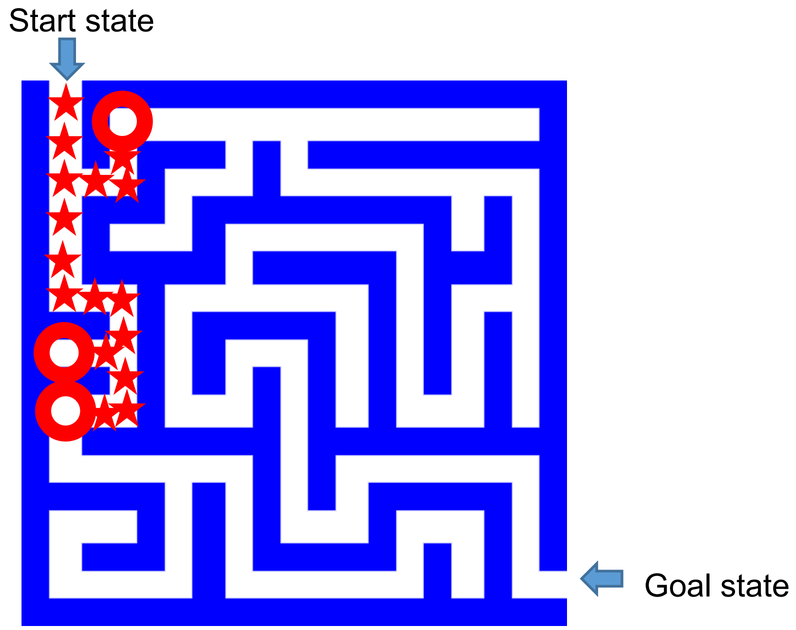
Smart Greedy Search

So we move forward along THAT path instead, until we reach this point, where all three nodes have the same $g(n) + h(n)$.



Smart Greedy Search

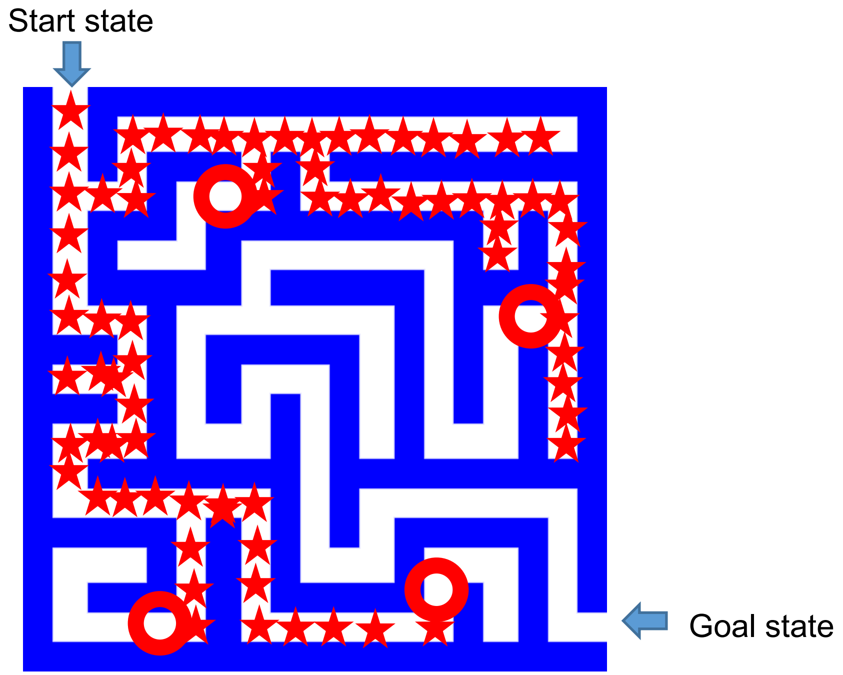
Moving forward on all three paths…



Smart Greedy Search

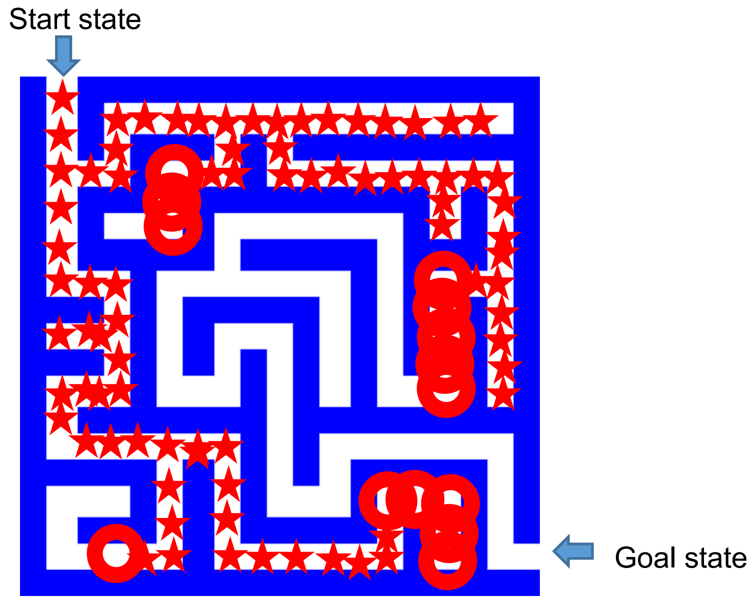
All of the new star nodes here had EXACTLY THE SAME value of $g(n) + h(n) = 34$.

Now these four circles, shown here, are the new frontier, the set of nodes with $g(n) + h(n) = 35$



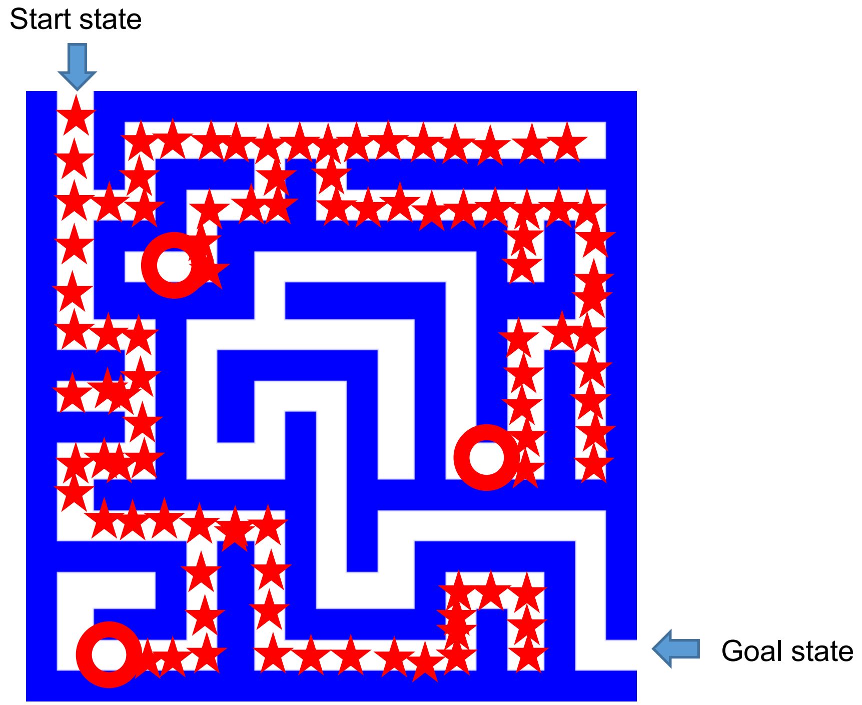
Smart Greedy Search

$$g(n) + h(n) = 36$$



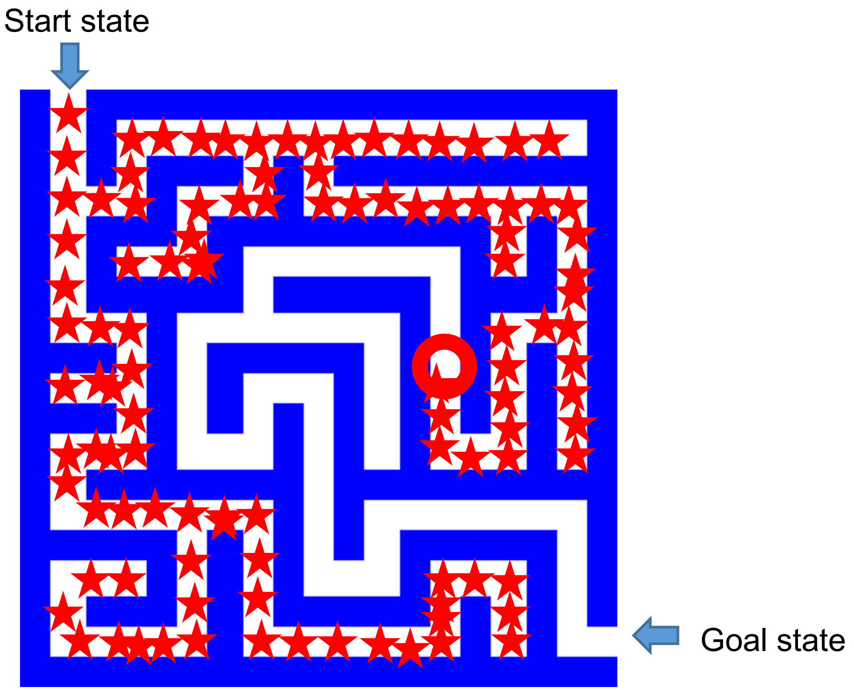
Smart Greedy Search

$$g(n) + h(n) = 37$$



Smart Greedy Search

$$g(n) + h(n) = 41$$



Smart Greedy Search

And so on…

I’m going to stop using this maze, at this point, because this maze was designed (by an author on Wikipedia) to be uniquely bad for A* search. A* search, on this maze, is just as bad as BFS.

Usually, A* search is much better than BFS. But not always.

"Almost-A* Search"

- Idea: avoid expanding paths that are already expensive
- The **evaluation function** $f(n)$ is the estimated total cost of the path through node n to the goal:

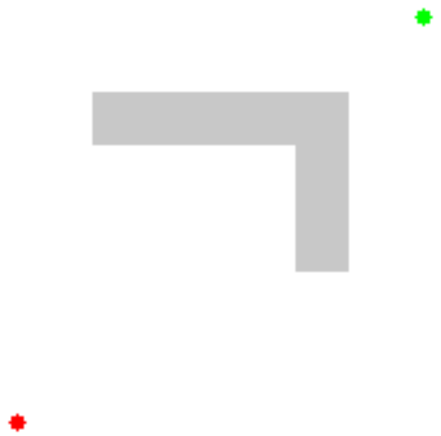
$$f(n) = g(n) + h(n)$$

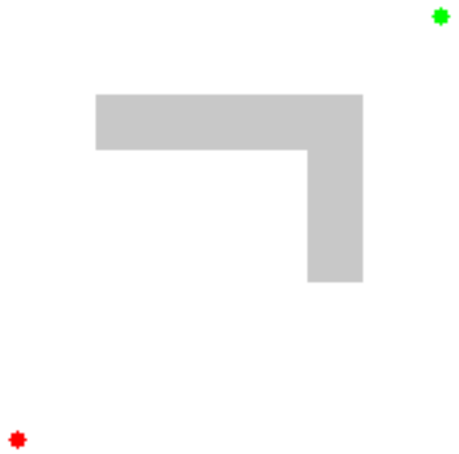
$g(n)$: cost so far to reach n (path cost)
$h(n)$: estimated cost from n to goal (heuristic)

- This is called A* search if and only if the heuristic, h(n), is admissible. That's a term I'll define a few slides from now. But first, let's look at an example where A* is much better than BFS.

BFS vs. A* Search

- The heuristic h(n)=Manhattan distance favors nodes on the main diagonal. Those nodes all have the same g(n)+h(n), so A* evaluates them first.





Source: Wikipedia

Outline

ZJUI

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

Problems with "Almost-A*"

- "Almost-A*" search looks pretty good! So are we done?
- There's one more problem. What, exactly, do we mean by the squiggly lines in these two equations:

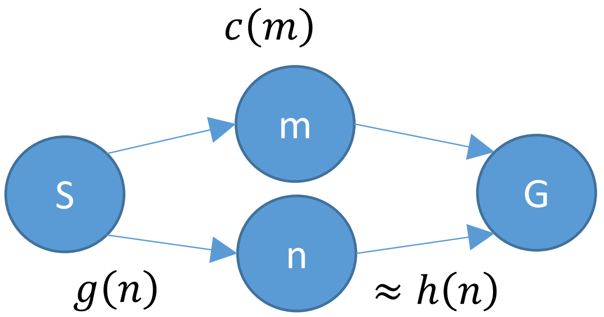
Distance from n to Goal is "**approximately**" h(n):

$$d(n) \approx h(n)$$

Total cost of the path through n is "**approximately**" g(n)+h(n)

$$c(n) \approx g(n) + h(n)$$

Problems with "Almost A*"



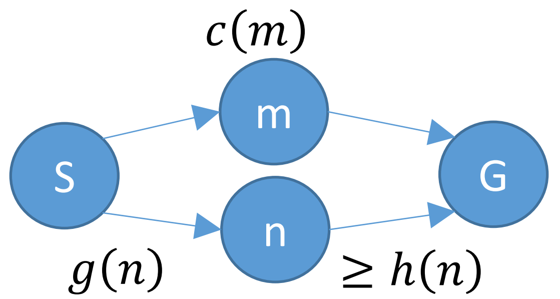
- Suppose we've found one path to G; the path goes through node m. Since we've calculated the whole path, we know its total path cost to be $c(m)$.
- Suppose that, for every other node on the frontier (n), we have

$$h(n) + g(n) > c(m)$$

Does that mean that $c(m)$ is really the best path?

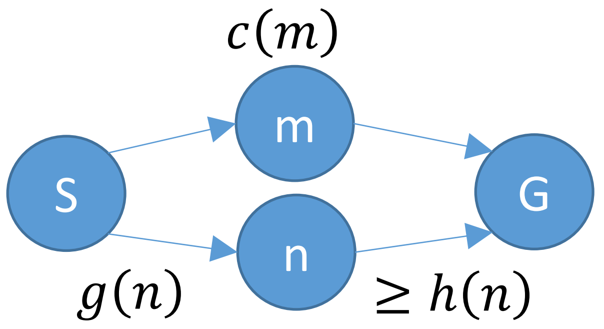
- No!! Because all we know is that $c(n) \approx g(n) + h(n)$.
- "Approximately" allows the possibility that $c(n) < g(n) + h(n)$.
- Therefore it's possible that $c(n) < c(m)$.

Admissible heuristic



- We want to guarantee that
$$c(n) \geq g(n) + h(n)$$
- Then if we can find a best path, m, such that for every node n left on the frontier,
$$h(n) + g(n) \geq c(m)$$
- Then we are guaranteed that there is no better node. We are guaranteed that for every node n that is not on the path m,
$$c(n) \geq h(n) + g(n) \geq c(m)$$

Admissible heuristic



- Remember that the total path cost is $c(n) = g(n) + d(n)$. So in order to guarantee that

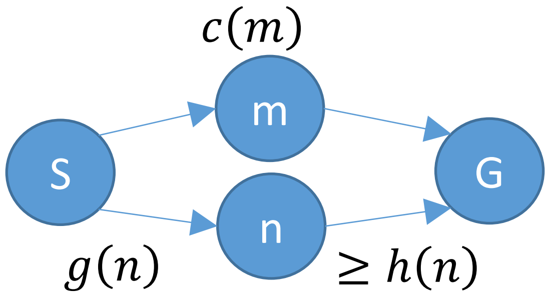
$$c(n) \geq g(n) + h(n)$$

we just need

$$d(n) \geq h(n)$$

Definition: A heuristic $h(n)$ is **admissible** if $d(n) \geq h(n)$, i.e., if the heuristic is guaranteed to be less than or equal to the remaining path cost from node n to the goal state.

A* Search



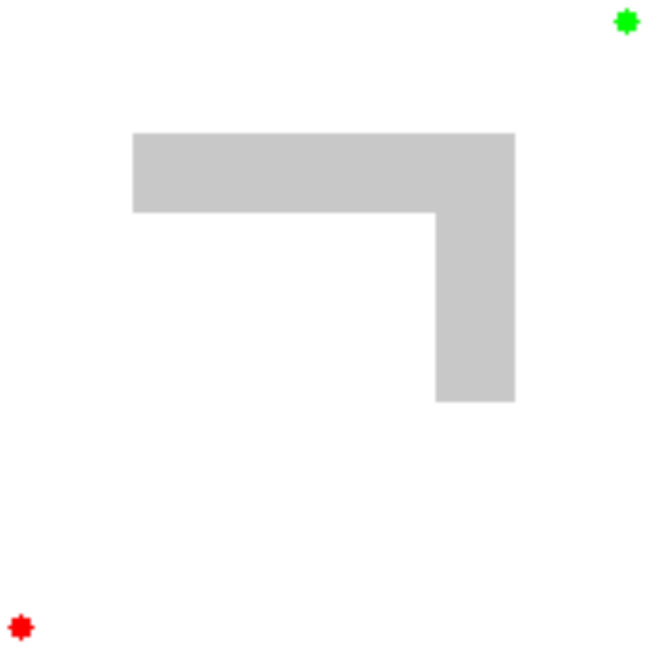
Definition: A* SEARCH

- If $h(n)$ is **admissible** $(d(n) \geq h(n))$, and
- if the frontier is a priority queue sorted according to $g(n) + h(n)$, then
- the FIRST path to goal uncovered by the tree search, path m, is guaranteed to be the SHORTEST path to goal

$(h(n) + g(n) \geq c(m)$ for every node n that is not on path $m)$

Example A* Search: Manhattan Distance

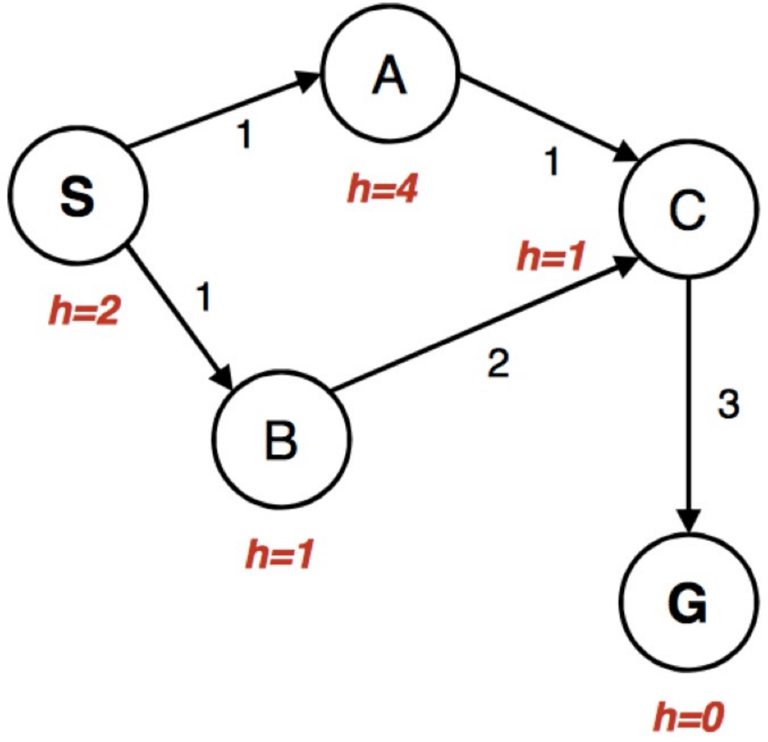
- Manhattan distance is guaranteed to be less than or equal to the true path to goal
- Therefore, “smart greedy” search with Manhattan distance heuristic = A* Search



Outline

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

Bad interaction between A* and the explored set



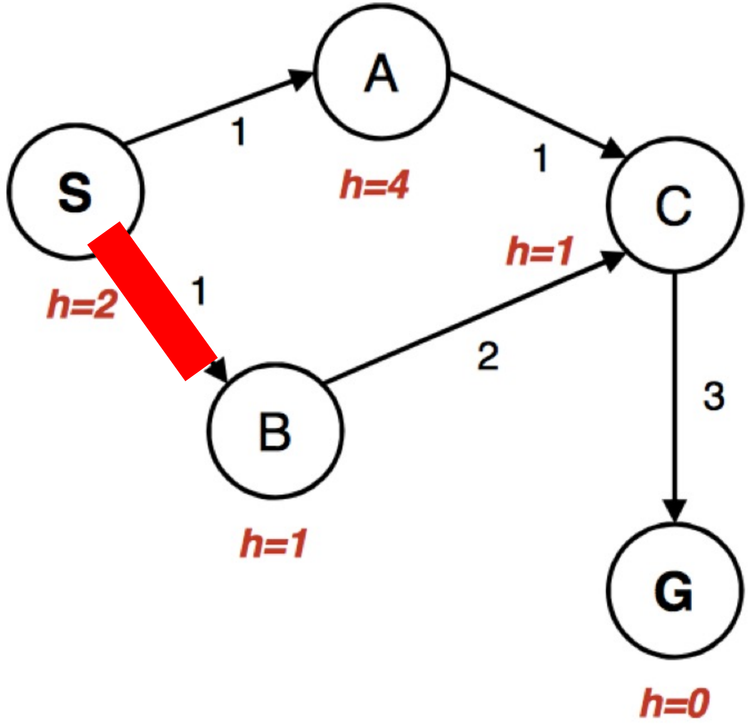
Frontier

S: g(n)+h(n)=2, parent=none

Explored Set

Select from the frontier: S

Bad interaction between A* and the explored set



Frontier

A: g(n)+h(n)=5, parent=S

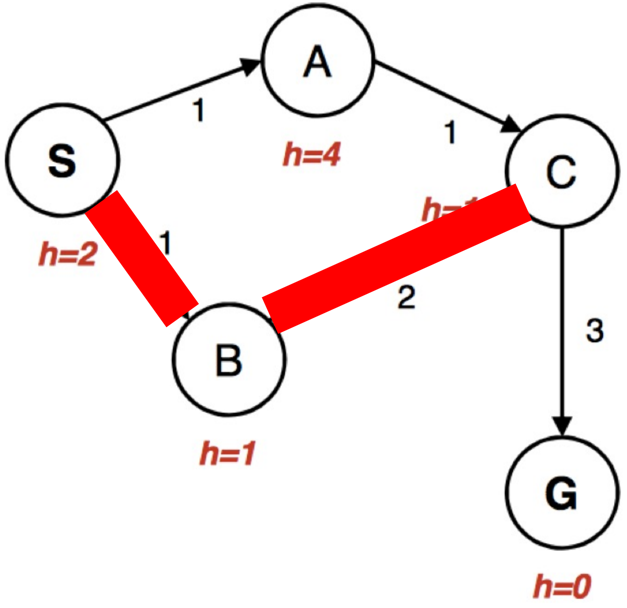
B: g(n)+h(n)=2, parent=S

Explored Set

S

Select from the frontier: B

Bad interaction between A* and the explored set



Frontier

A: g(n)+h(n)=5, parent=S

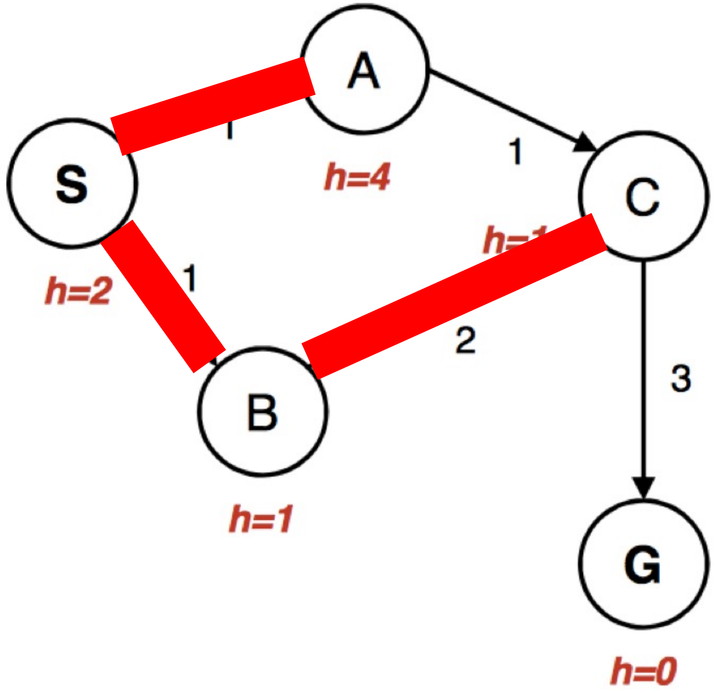
C: g(n)+h(n)=4, parent=B

Explored Set

S, B

Select from the frontier: C

Bad interaction between A* and the explored set



Frontier

A: g(n)+h(n)=5, parent=S

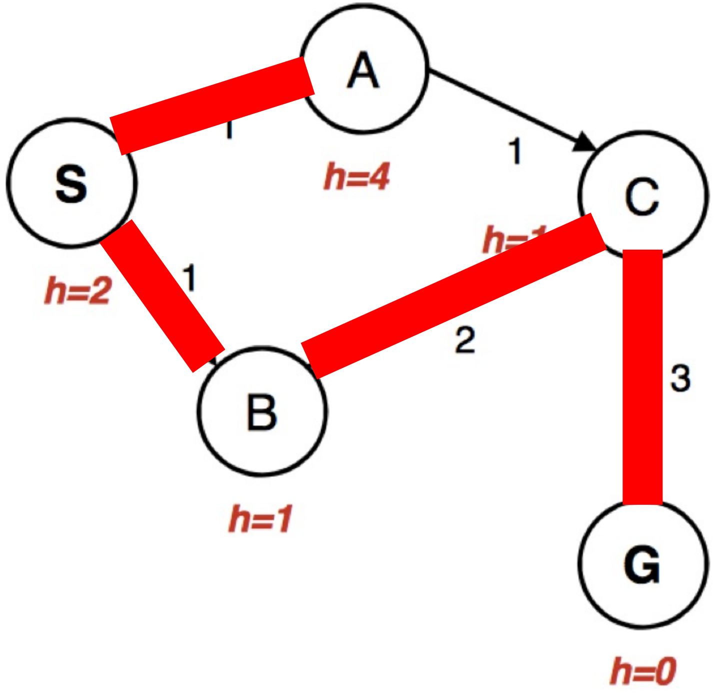
G: g(n)+h(n)=6, parent=C

Explored Set

S, B, C

Select from the frontier: A

Bad interaction between A* and the explored set



Frontier

G: g(n)+h(n)=6, parent=C

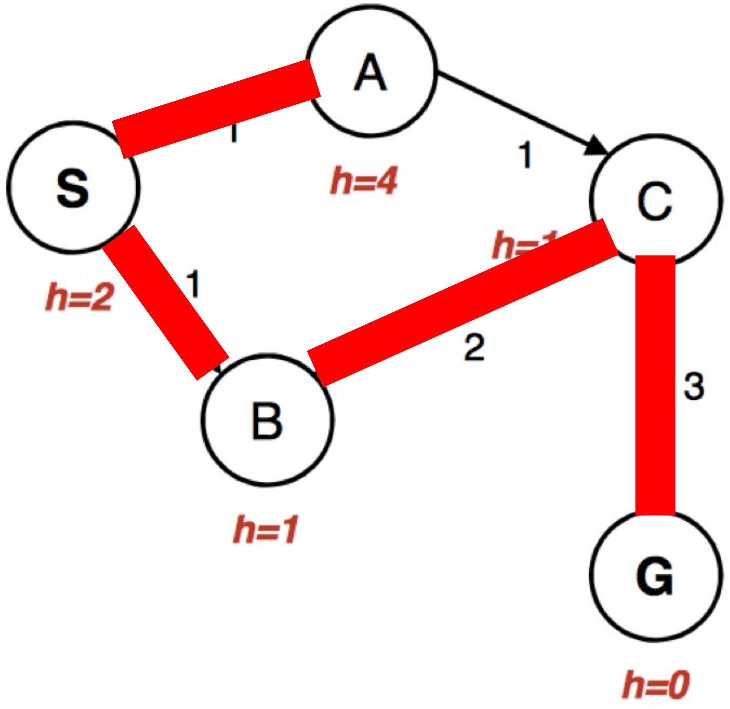
- **Now we would place C in the frontier, with parent=A and h(n)+g(n)=3, except that C was already in the explored set!**

Explored Set

S, B, C

Select from the frontier: **Would be C**, but instead it’s G

Bad interaction between A* and the explored set



Return the path S,B,C,G

Path cost = 6

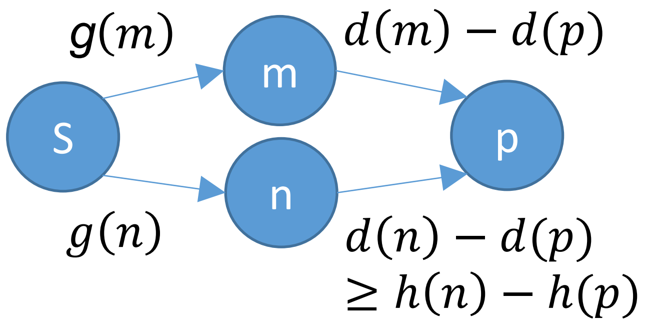
OOPS

ZJUI

Three possible solutions

1. Don’t use an explored set
   - This option is OK for any finite state space, as long as you check for loops.
2. Nodes on the explored set are tagged by their h(n)+g(n). If you find a node that’s <u>already in the explored set</u>, test to see if the <u>new h(n)+g(n) is smaller than the old one</u>.
   - If so, put the node back on the frontier
   - If not, leave the node off the frontier
3. Use a heuristic that’s not only admissible, but also consistent.

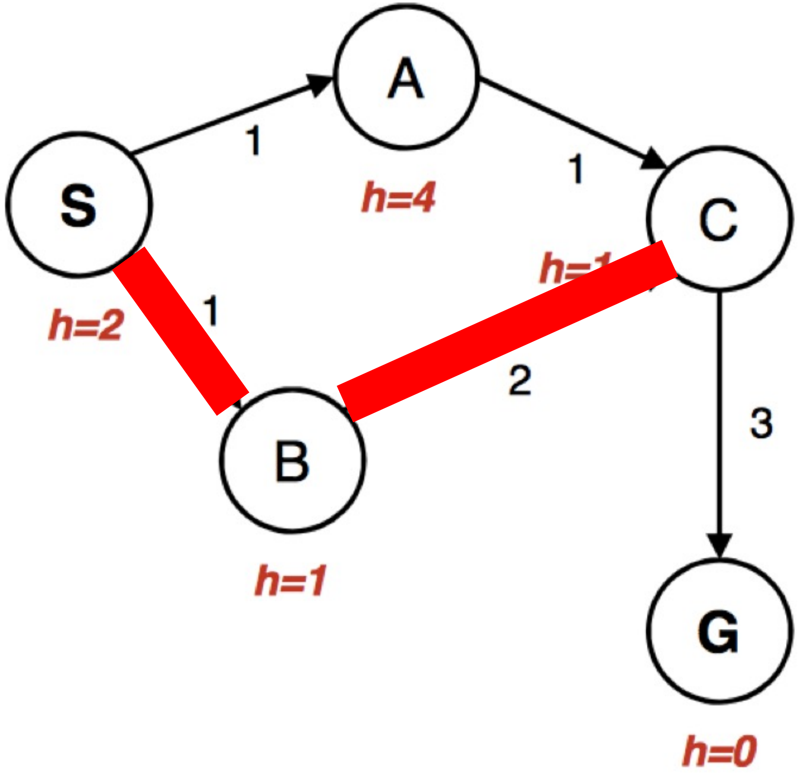
Consistent (monotonic) heuristic



Definition: A **consistent heuristic** is one for which, for every pair of nodes in the graph, $d(n) - d(p) \geq h(n) - h(p)$.

In words: the distance between any pair of nodes is greater than or equal to the difference in their heuristics.

A* with an inconsistent heuristic



Frontier

A: g(n)+h(n)=5, parent=S

C: g(n)+h(n)=4, parent=B

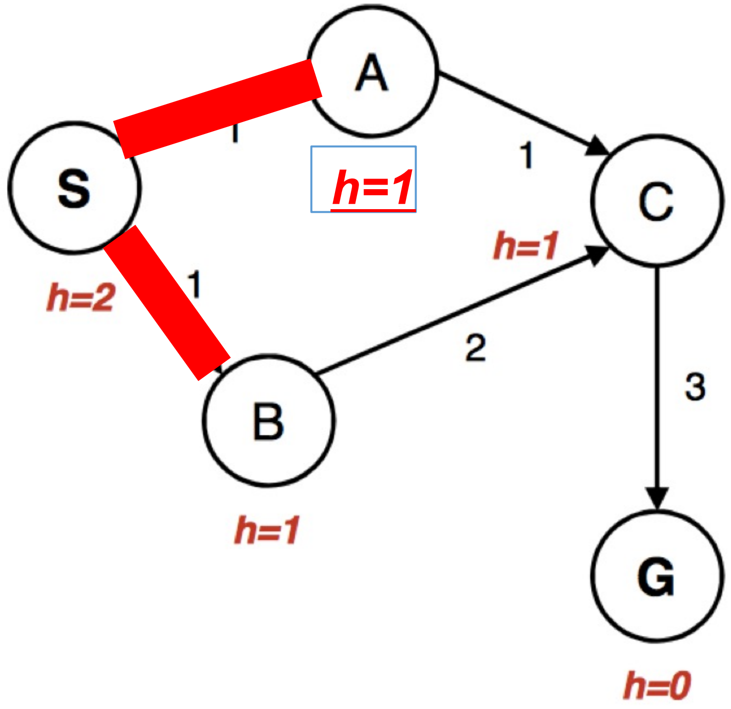
Explored Set

S, B

Select from the frontier: C

A* with a consistent heuristic

ZJUI



Frontier

A: g(n)+h(n)=**2**, parent=S

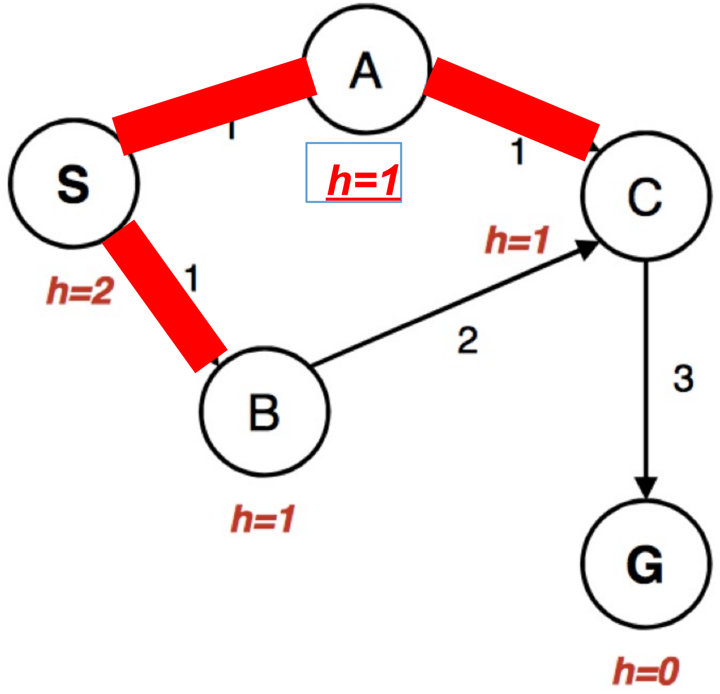
C: g(n)+h(n)=4, parent=B

Explored Set

S, B

Select from the frontier: **A**

A* with a consistent heuristic



Frontier

.

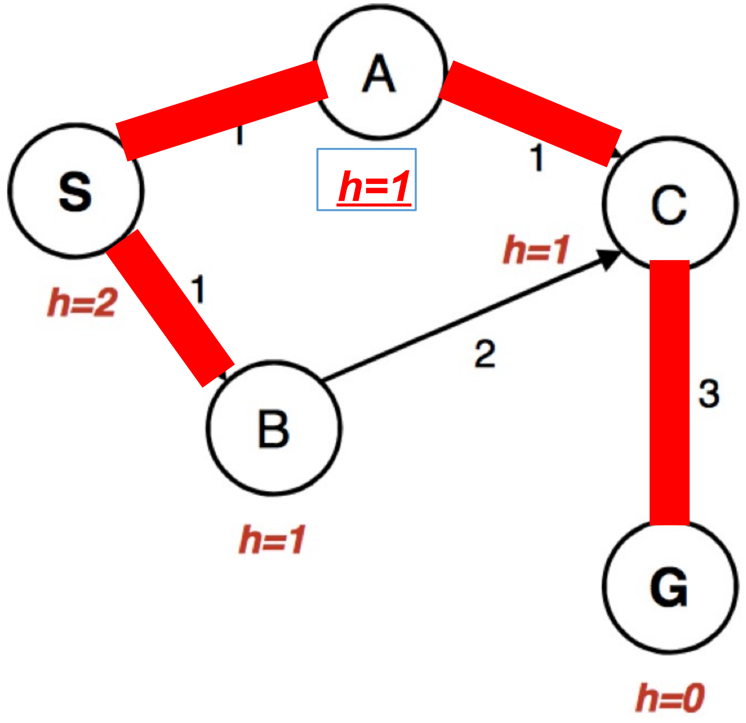
C: g(n)+h(n)=**2**, parent=**A**

Explored Set

S, B, **A**

Select from the frontier: **C**

A* with a consistent heuristic



Frontier

.

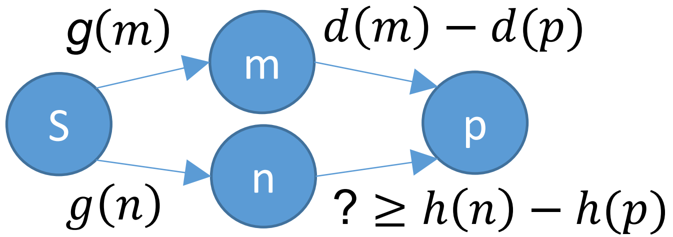
G: g(n)+h(n)=<u>**5**</u>, parent=C

Explored Set

S, B, <u>**A,**</u> C

Select from the frontier: G

How consistency works



Suppose that, on the best path from start to node p, node m is p's parent, and say that $d(m) - d(p)$ is the distance between them. Then the distance from start to node p is

$$g(p) = g(m) + (d(m) - d(p)) \leq g(n) + (d(n) - d(p))$$

Definition: A **consistent heuristic** is one for which, for every pair of nodes in the graph, $d(n) - d(p) \geq h(n) - h(p)$.

Implication:

$$g(p) \geq g(m) + \big(h(m) - h(p)\big)$$
$$g(p) + h(p) \geq g(m) + h(m)$$

- $g(p) + h(p)$ is monotonically non-decreasing along the path!! So it is guaranteed that node m is expanded before node p. (We have no such guarantees about node n).
- By the time node p is popped from the frontier, it might have been inserted onto the frontier by many different paths. Each path uses the same h(p), but computes a different g(p). The shortest one (through node m) is guaranteed to already be on the frontier by that time, and is guaranteed to have inserted the best g(p).

Three possible solutions

1. Don’t use an explored set.

This works for the MP!

2. If you find a node that’s already in the explored set, test to see if the new h(n)+g(n) is smaller than the old one.

Most students find that this is the most computationally efficient solution to the multi-dots problem.

3. Use a consistent heuristic.

This works for the single-dot problem, because Manhattan distance is a consistent heuristic.

Outline

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. **Dijkstra = A* with h(n)=0**
7. Designing heuristics: Relaxed problem, Sub-problem, Dominance

The trivial case: h(n)=0

- A heuristic is **admissible** if and only if $d(n) \geq h(n)$ for every n.
- A heuristic is **consistent** if and only if $d(n, p) \geq h(n) - h(p)$ for every n and p.

- Both criteria are satisfied by $h(n) = 0$.

Dijkstra = A* with h(n)=0

- Suppose we choose $h(n) = 0$
- Then the frontier is a priority queue sorted by
$$g(n) + h(n) = g(n)$$
- In other words, the first node we pull from the queue is the one that's closest to START!! (The one with minimum $g(n)$).
- So this is just Dijkstra's algorithm!

Outline

ZJUI

1. Search heuristics
2. Greedy best-first search: minimum h(n)
3. Nearly-A*: f(n)=h(n)+g(n)
4. A*: Optimal search
5. Bad interaction between A* and the explored set
6. Dijkstra = A* with h(n)=0
7. **Designing heuristics: Relaxed problem, Sub-problem, Dominance**

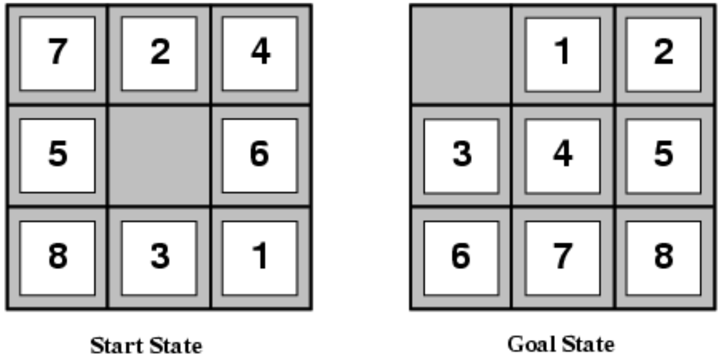
Designing heuristic functions

Now we start to see things that actually resemble the multi-dot problem…

- Heuristics for the 8-puzzle

  $h_1(n)$ = number of misplaced tiles

  $h_2(n)$ = total Manhattan distance (number of squares from desired location of each tile)



$h_1(\text{start}) = 8$

$h_2(\text{start}) = 3+1+2+2+2+3+3+2 = 18$

- Are h_1 and h_2 admissible?

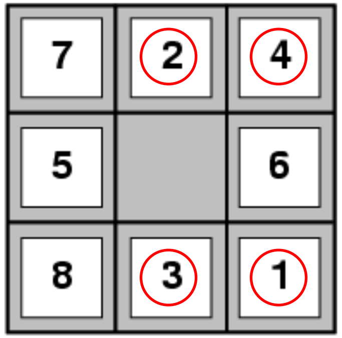
Heuristics from relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

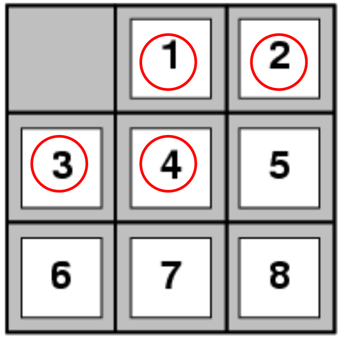
Heuristics from subproblems

This is also a trick that many students find useful for the multi-dot problem.

- Let $h_3(n)$ be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions
- Can precompute and save the exact solution cost for every possible subproblem instance – *pattern database*
- If the subproblem is O{9^4}, and the full problem is O{9^9}, then you can solve as many as 9^5 subproblems without increasing the complexity of the problem!!



Start State



Goal State

Dominance

- If h_1 and h_2 are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all n, (both admissible) then h_2 dominates h_1
- Which one is better for search?
  - A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$
  - Therefore, A* search with h_1 will expand more nodes = h_1 is more computationally expensive.

Dominance

- Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths):

- d=12 BFS expands 3,644,035 nodes
  A*(h_1) expands 227 nodes
  A*(h_2) expands 73 nodes

- d=24 BFS expands 54,000,000,000 nodes
  A*(h_1) expands 39,135 nodes
  A*(h_2) expands 1,641 nodes

Combining heuristics

- Suppose we have a collection of admissible heuristics $h_1(n)$, $h_2(n)$, …, $h_m(n)$, but none of them dominates the others
- How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

All search strategies. C*=cost of best path.

ZJUI

Algorithm	Complete?	Optimal?	Time complexity	Space complexity	Implement the Frontier as a…
BFS	Yes	If all step costs are equal	O(b^d)	O(b^d)	Queue
DFS	No	No	O(b^m)	O(bm)	Stack
IDS	Yes	If all step costs are equal	O(b^d)	O(bd)	Stack
UCS	Yes	Yes	Number of nodes w/ g(n) ≤ C*	Number of nodes w/ g(n) ≤ C*	Priority Queue sorted by g(n)
Greedy	No	No	Worst case: O(b^m) Best case: O(bd)	Worse case: O(b^m) Best case: O(bd)	Priority Queue sorted by h(n)
A*	Yes	Yes	Number of nodes w/ g(n)+h(n) ≤ C*	Number of nodes w/ g(n)+h(n) ≤ C*	Priority Queue sorted by h(n)+g(n)