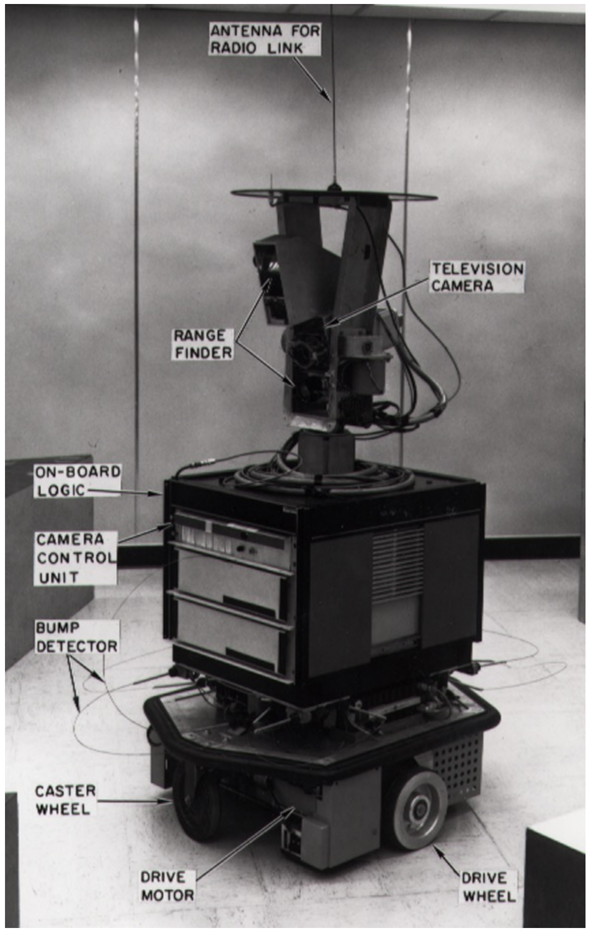
Lecture 15: A* Search

Mark Hasegawa-Johnson

2/2023



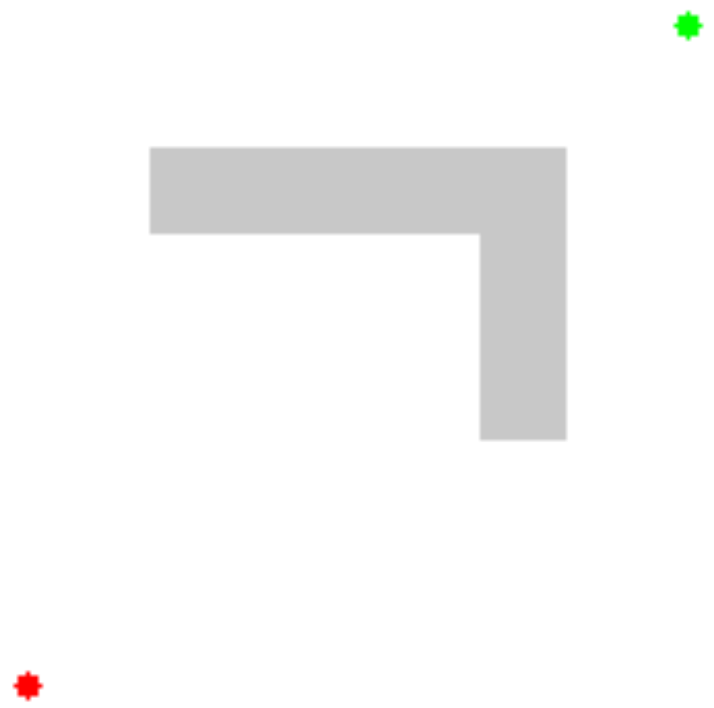
By SRI International - SRI International, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17294520

Contents

- A* search: Using a heuristic to help choose which node to expand
- Proof that Dijkstra’s algorithm is optimal
- Heuristics that allow A* to be optimal:
  - Consistent: $h(p) \leq d(p,r) + h(r)$
  - Admissible: $h(p) \leq d(p, Goal)$
- Designing a consistent heuristic by relaxing constraints

Why is BFS slow?

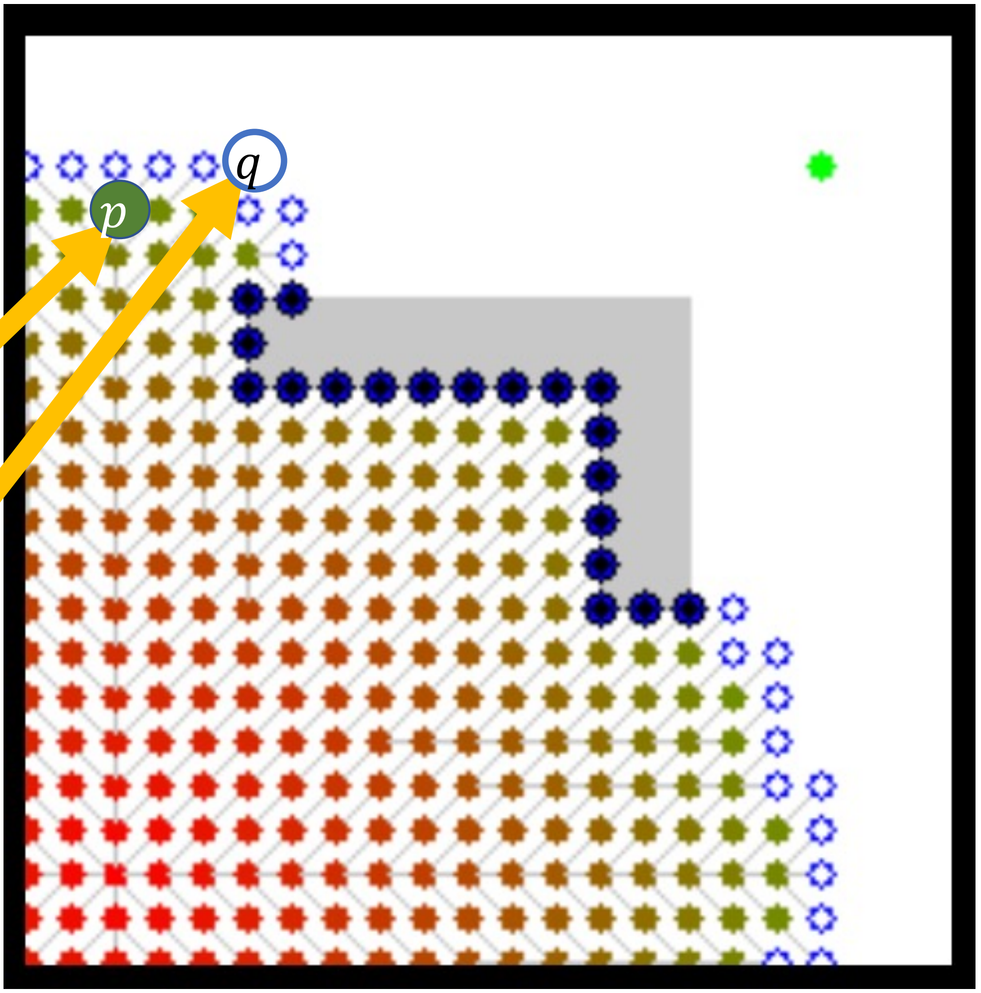
- Before we expand a node that is d steps from the start,
- … we must expand all nodes that are $d-1$ steps from the start.
- Result: complexity is $O\{b^d\}$



CC-SA 3.0,
https://commons.wikimedia.org/wiki/File:Dijkstras_progress_animation.gif

Speeding up BFS and Dijsktra's algorithm (the intuition)

- Intuitively, this node, which is farther from the goal,
- …should not have been expanded before this node, because this one is closer to the goal.



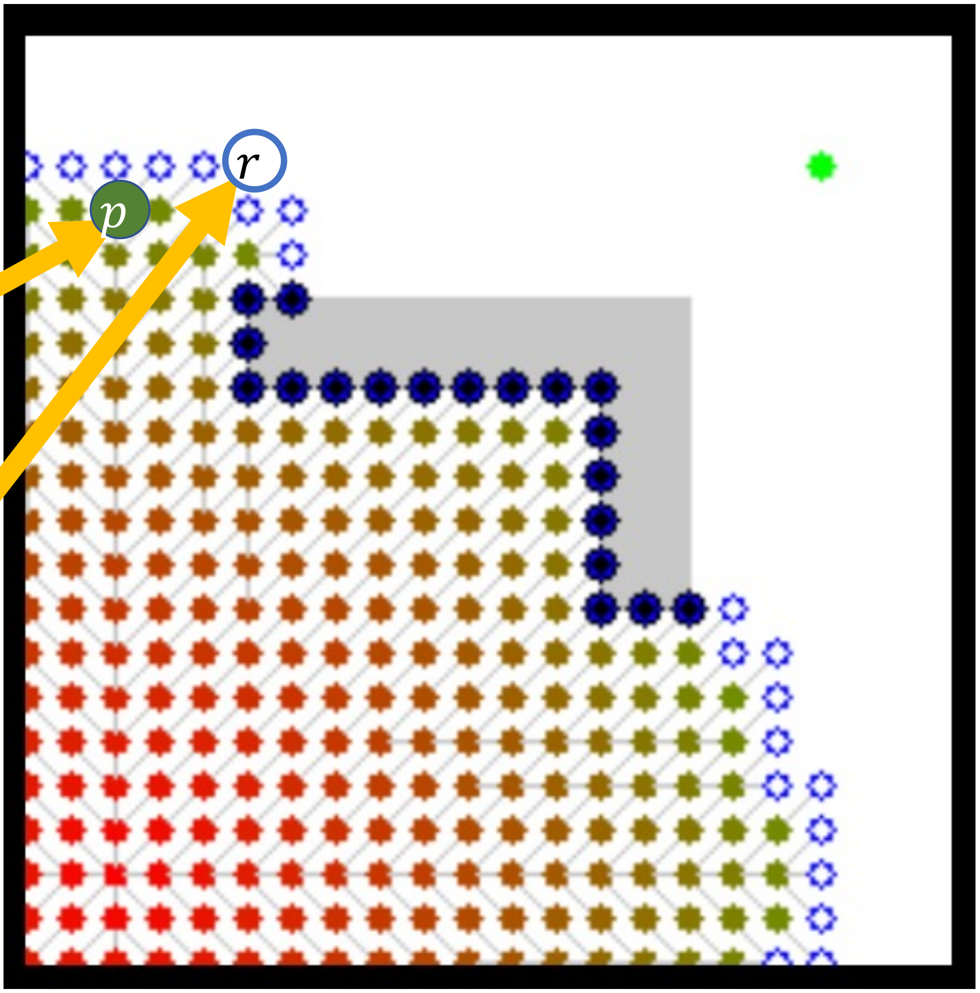
Why was Dijkstra slow?

- Dijkstra's algorithm expanded this node first because its distance from the START node is only

$$g(p) = 15$$

- This node is expanded second, because its distance from the START node is

$$g(r) = 16$$



Fixing Dijkstra's algorithm

Instead of sorting nodes by how far they are from Start, can we sort nodes based on the total length of the best path that goes through that node?

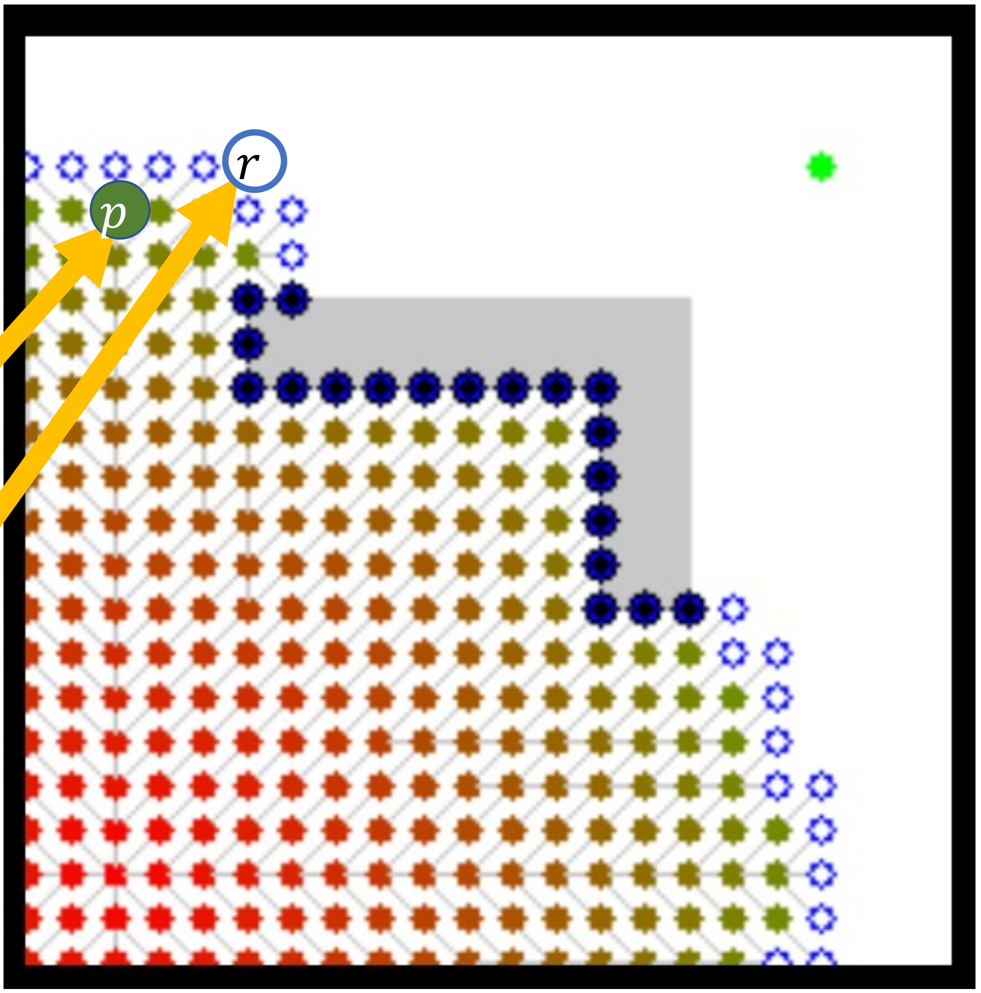
- This node has

$$g(p) = 15$$
$$h(p) = 16$$
$$g(p) + h(p) = 31$$

- This node has

$$g(r) = 16$$
$$h(r) = 13$$
$$g(r) + h(r) = 29$$

…so this node should be expanded first.



A* search: Estimate $f(p)$, the total cost of the best path that goes through node $\boldsymbol{p}$

- DEFINE: $g(p)$ = cost of the best path from the START node to node p,
$$g(p) = d(Start, p)$$
- DEFINE: $h(p)$ = heuristic (approximate) estimate of the distance from p to $Goal$. Finding $h(p)$ must be less expensive than finding the true distance $d(p, Goal)$! So it's not exactly equal, only approximately:
$$h(p) \approx d(p, Goal)$$
- RESULT: estimate of the **<u>total length of the path through node $\boldsymbol{p}$</u>** is

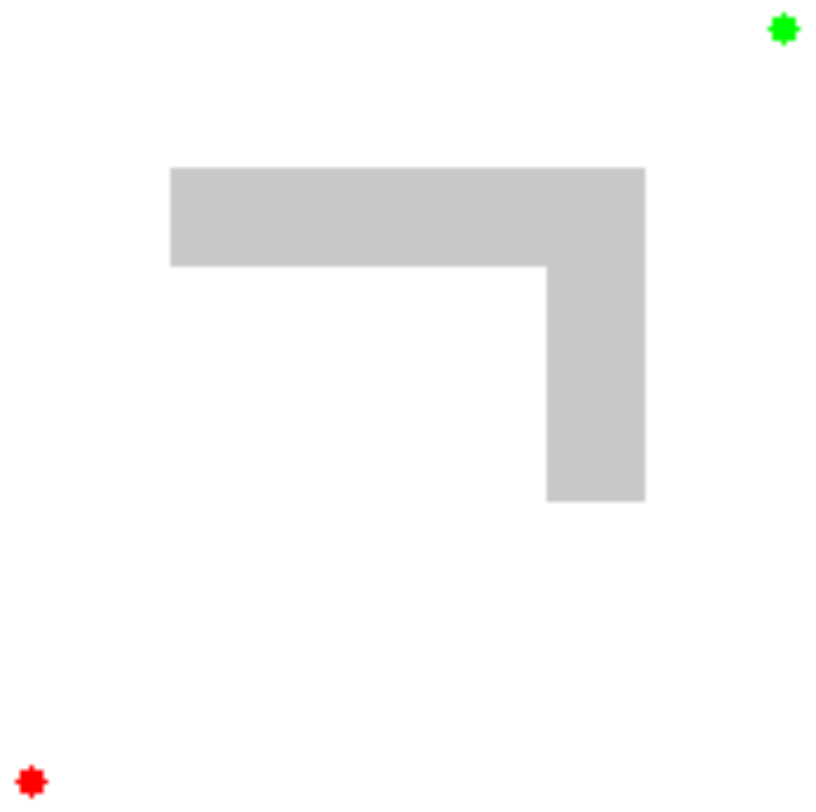
$$f(p) \ = \ g(p) + h(p) \approx d(Start, p) + d(p, Goal)$$

A* Search

The A* algorithm is just like Dijkstra's algorithm, except that,

- At each iteration,
- instead of expanding the node with the lowest $g(p)$,
- …expand the node with the lowest $g(p) + h(p)$

(In this example, $h(x)$ =Euclidean distance to Goal)

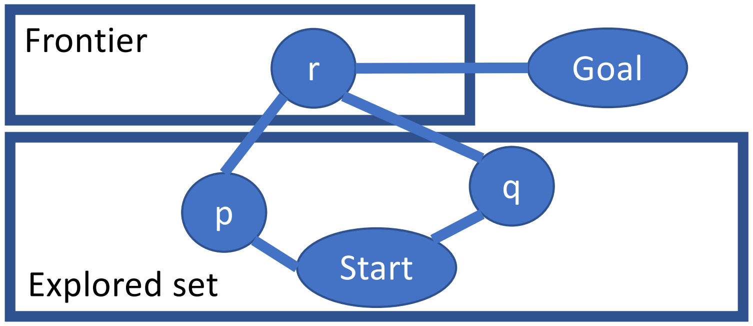


By Subh83 - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=14916867

Contents

- A* search: Using a heuristic to help choose which node to expand
- Proof that Dijkstra’s algorithm is optimal
- Heuristics that allow A* to be optimal:
    - Consistent: $h(p) \leq d(p,r) + h(r)$
    - Admissible: $h(p) \leq d(p, Goal)$
- Designing a consistent heuristic by relaxing constraints

Proof that Dijkstra's algorithm is optimal

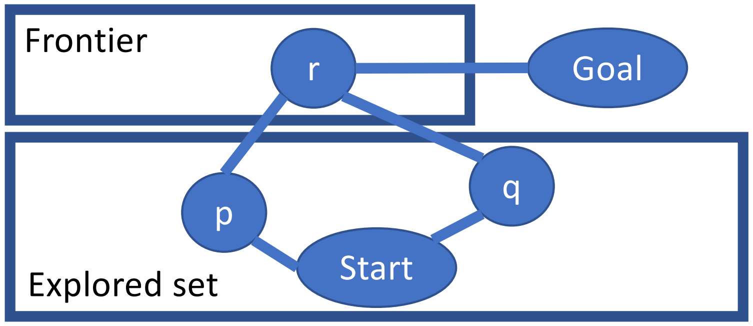


- Suppose that the shortest path to node r goes through node p
- Then the correct value of $g(r) = g(p) + d(p,r)$
- In order to make sure $g(r)$ is set correctly, we need to make sure that p is expanded before r
- But that is guaranteed, because $g(p) < g(r)$

Contents

- A* search: Using a heuristic to help choose which node to expand
- Proof that Dijkstra's algorithm is optimal
- Heuristics that allow A* to be optimal:
  - Consistent: $h(p) \le d(p,r) + h(r)$
  - Admissible: $h(p) \le d(p, Goal)$
- Designing a consistent heuristic by relaxing constraints

How can we make sure A* finds the shortest path to node r?



- Suppose the shortest path to r goes through p. Then, in order to make sure $g(r)$ is set correctly, we need to make sure that p is expanded before r, so it will set $g(r)$ to

$$g(r) = g(p) + d(p,r)$$

- If we are using A* search, then p is expanded before r if:

$$g(p) + h(p) \leq g(r) + h(r)$$

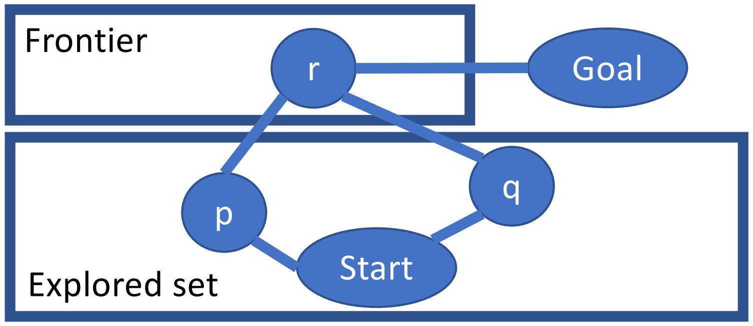
- Combining these two equations: A* finds the shortest path to node x if

$$g(p) + h(p) \leq g(p) + d(p,r) + h(r)$$

…or, in other words,

$$h(p) \leq d(p,r) + h(r)$$

Consistent heuristic: the first time r is expanded, it is expanded with the right cost

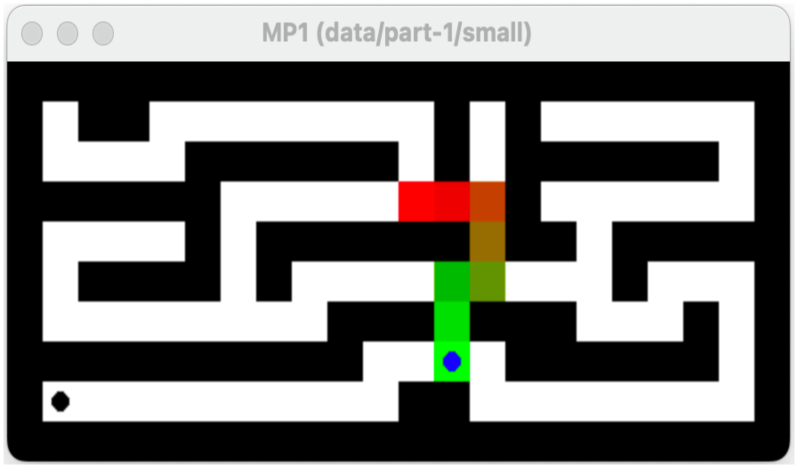


A* guarantees that r has the correct $g(r)$ the first time it is expanded (moved from the frontier to the explored set) if, for every p such that $g(p) < g(r)$,

$$h(p) \leq d(p,r) + h(r)$$

A heuristic with this property is called **consistent**.

Example of a consistent heuristic: Manhattan distance



- Consider a maze in which only L-R and U-D moves are possible
- If there were no walls between $p = (y_p, x_p)$ and $r = (y_r, x_r)$, then their distance would be the number of horizontal steps, plus the number of vertical steps. We call this the "Manhattan distance," and write it as:

$$\|p - r\|_1 = |y_p - y_r| + |x_p - x_r|$$

- If there are walls, then the distance may be larger:

$$d(p, r) \geq |y_p - y_r| + |x_p - x_r|$$

Example of a consistent heuristic: Manhattan distance

Suppose the location of the goal is (y_g, x_g).
Suppose we define the heuristic as

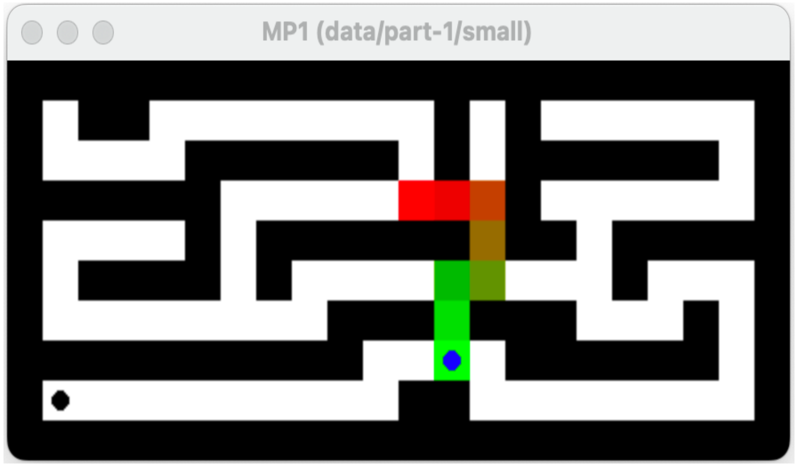
$$h(p) = |y_p - y_g| + |x_p - x_g|$$
$$h(r) = |y_r - y_g| + |x_r - x_g|$$

Notice we are guaranteed that:

$$|y_p - y_g| \le |y_p - y_r| + |y_r - y_g|$$
$$|x_p - x_g| \le |x_p - x_r| + |x_r - x_g|$$

If you add those two equations together, you discover that:

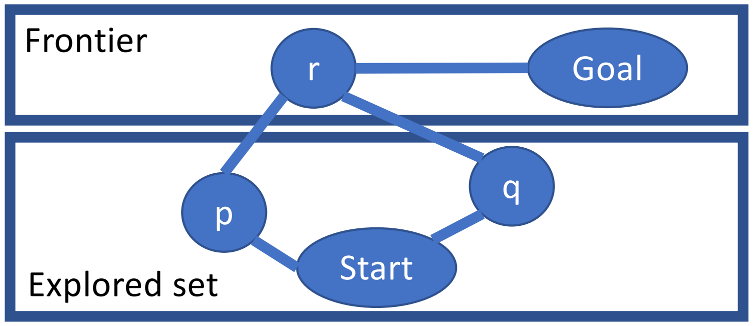
$$h(p) \le d(p, r) + h(r)$$



Contents

- A* search: Using a heuristic to help choose which node to expand
- Proof that Dijkstra's algorithm is optimal
- Heuristics that allow A* to be optimal:
  - Consistent: $h(p) \leq d(p,r) + h(r)$
  - Admissible: $h(p) \leq d(p, Goal)$
- Designing a consistent heuristic by relaxing constraints

Admissible heuristic: Consistent only for the Goal



Suppose that we don’t care whether or not $g(r)$ is correct the first time r is expanded (e.g., maybe we will allow ourselves to expand it over and over again, in case $g(r)$ is wrong the first time).

However, suppose we want to make sure that the first time the Goal is expanded, it has the correct optimal path.

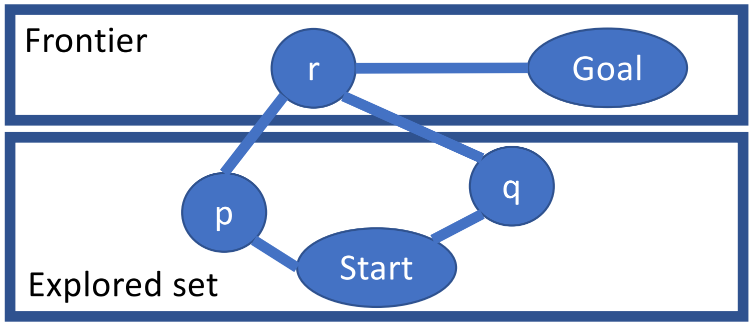
We can do this if $h(p)$ satisfies:

$$h(p) \leq \; d(p, Goal) + h(Goal)$$

…but why not just set $h(Goal) = 0$? In that case, we have the very simple constraint that

$$h(p) \leq d(p, Goal)$$

Admissible heuristic: Consistent only for the Goal



A heuristic that satisfies the constraint:

$$h(p) \leq d(p, Goal)$$

…is called **<u>admissible</u>**.

- All consistent heuristics are also admissible, but not vice versa.
- Sometimes, you might have a search problem where you can't figure out how to make your heuristic consistent, but you can figure out how to make it admissible.
- A* search with an admissible heuristic is still optimal if other nodes (besides Goal) get re-expanded every time they get popped from the frontier.

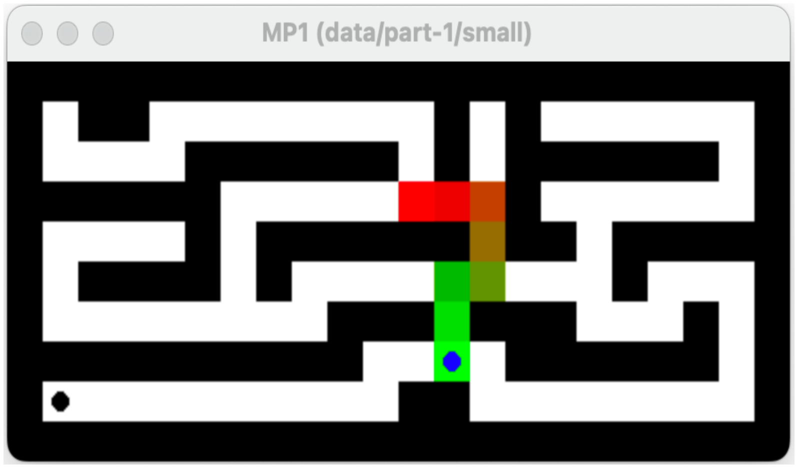
Try the quiz!

- Try the quiz: https://us.prairielearn.com/pl/course_instance/129874/assessment/2333392
- D(F,G)=1, h(F)=0
- D(E.G)=3, h(E)=1
- D(B,G) = 9, h(B)=4
- D(C,G)=7, h(C) =2
- D(D,G) =9, h(D) =10
- D(A,G) = 11, h(A) = 3

Contents

- A* search: Using a heuristic to help choose which node to expand
- Proof that Dijkstra’s algorithm is optimal
- Heuristics that allow A* to be optimal:
  - Consistent: $h(p) \leq d(p,r) + h(r)$
  - Admissible: $h(p) \leq d(p, Goal)$
- Designing a consistent heuristic by relaxing constraints

Inventing a heuristic by relaxing constraints



Remember how we invented the heuristic $h(p) \ = |y_p - y_g| + |x_p - x_g|$ for the simple maze? We noticed that:

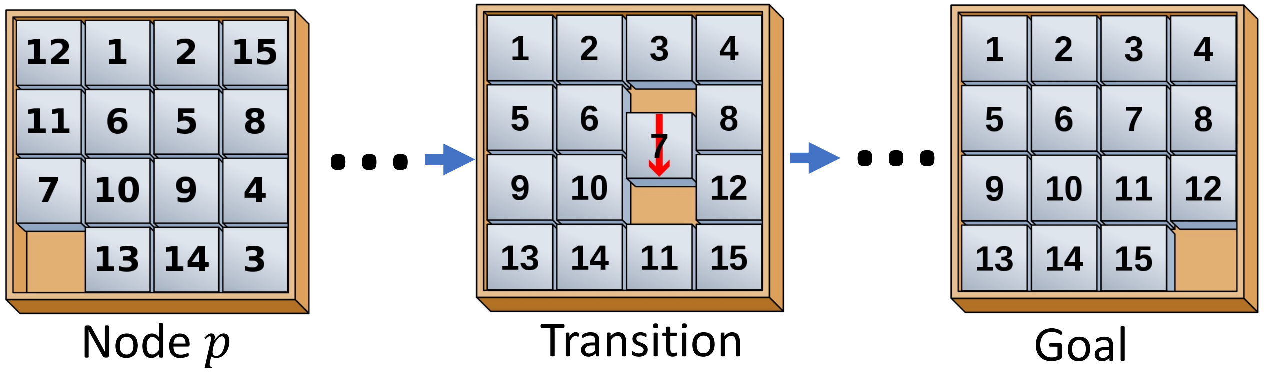
- If there were no walls between $p = (y_p, x_p)$ and $Goal = (y_g, x_g)$, then their distance would be the number of horizontal steps, plus the number of vertical steps. If there are walls, then the distance is larger:

$$d(p, Goal) \geq |y_p - y_g| + |x_p - x_g|$$

This is an example of a general principle: you can invent a heuristic by noticing what makes your problem hard (e.g., walls), and getting rid of it.

Example: the 15-puzzle

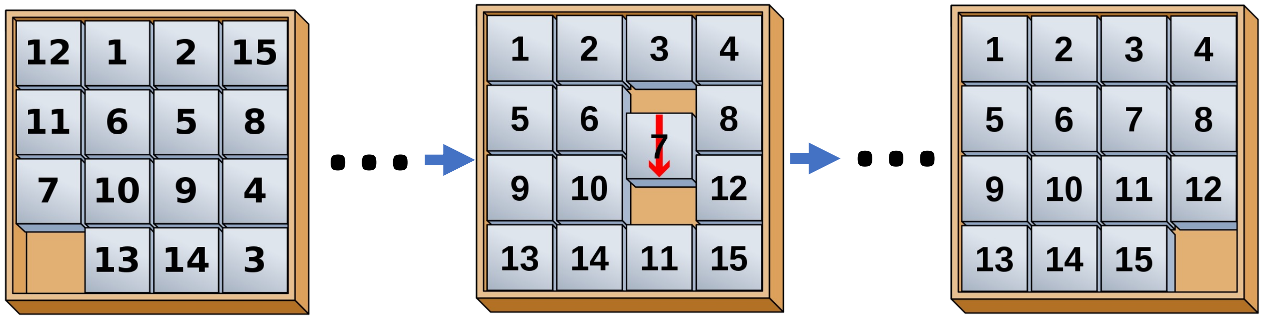
Public Domain, https://commons.wikimedia.org/wiki/File:15-puzzle_magical.svg



- For another example, consider the 15-puzzle: Shift one tile at a time until the puzzle reaches the goal state.
- What makes it hard is that you can’t move the 1-tile to its correct square, because the 12-tile is in the way.

Example: the 15-puzzle

Public Domain, https://commons.wikimedia.org/wiki/File:15-puzzle_magical.svg



Node p Transition Goal

We can design a heuristic (which makes A* search much faster) by just ignoring the constraint.

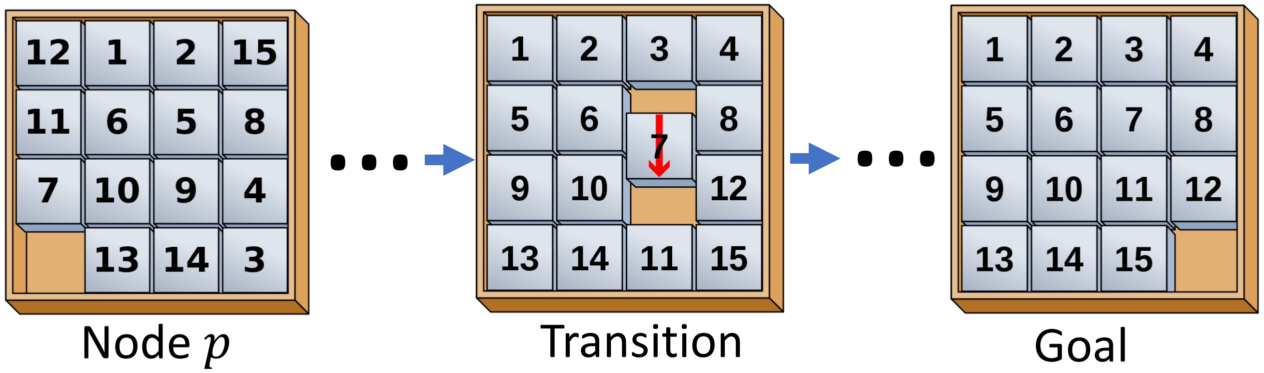
$$h(p) = \sum_{tile=1}^{15} \begin{matrix} number\ of\ squares\ tile\ would \\ have\ to\ move\ if\ there\ were\ no \\ other\ tiles\ in\ the\ way \end{matrix}$$

Since we can't really move the tiles in that way, we are guaranteed that

$$h(p)\ <=\ d(p, Goal)$$

Example: the 15-puzzle

Public Domain, https://commons.wikimedia.org/wiki/File:15-puzzle_magical.svg



BFS solution of the 15-puzzle expands 54,000,000,000 nodes.

A* solution, using the following heuristic, expands 1641 nodes.

$$h(p) = \sum_{tile=1}^{15} \begin{matrix} \textit{number of squares tile would} \\ \textit{have to move if there were no} \\ \textit{other tiles in the way} \end{matrix}$$

Contents

- A* search: Using a heuristic to help choose which node to expand
- Proof that Dijkstra’s algorithm is optimal
- Heuristics that allow A* to be optimal:
    - Consistent: $h(p) \leq d(p, r) + h(r)$
    - Admissible: $h(p) \leq d(p, Goal)$
- Designing a consistent heuristic by relaxing constraints