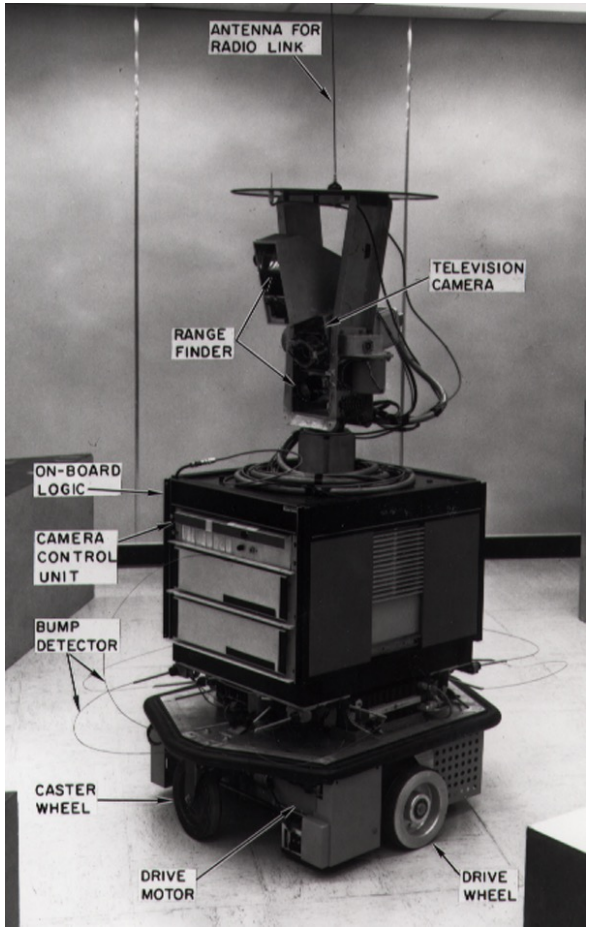
Lecture 21: A* Search

Mark Hasegawa-Johnson

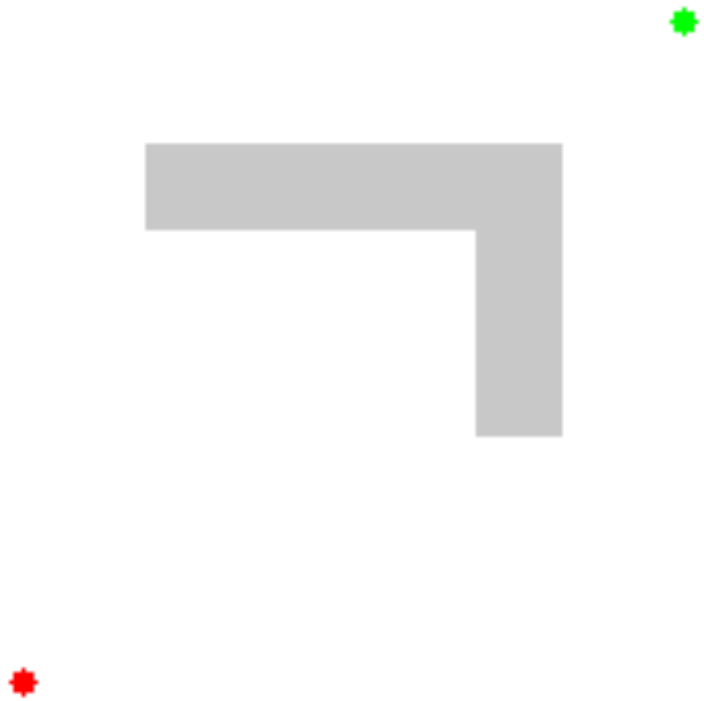


Contents

- A* search: Using a heuristic to help choose which node to expand
- Admissible search
- How to design a heuristic
- Consistent search

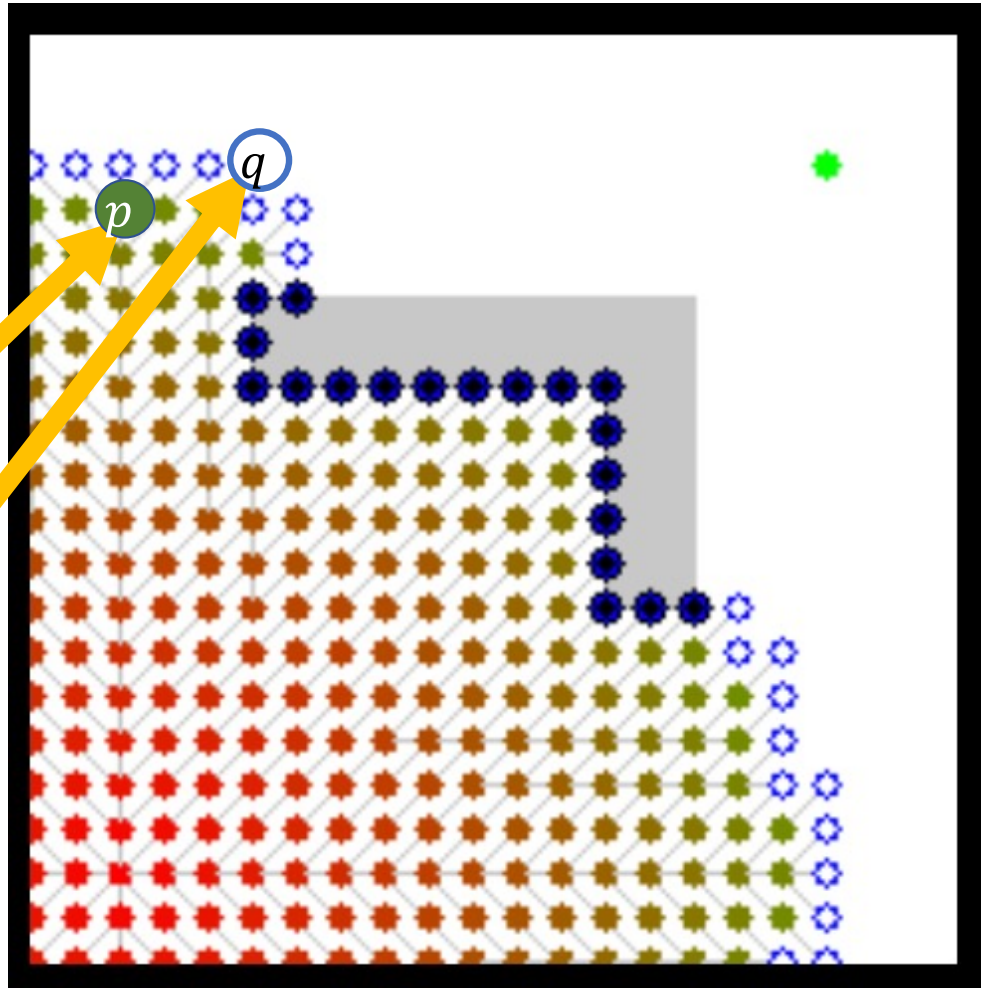
Why is BFS slow?

- 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 $O\{b^d\}$



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.



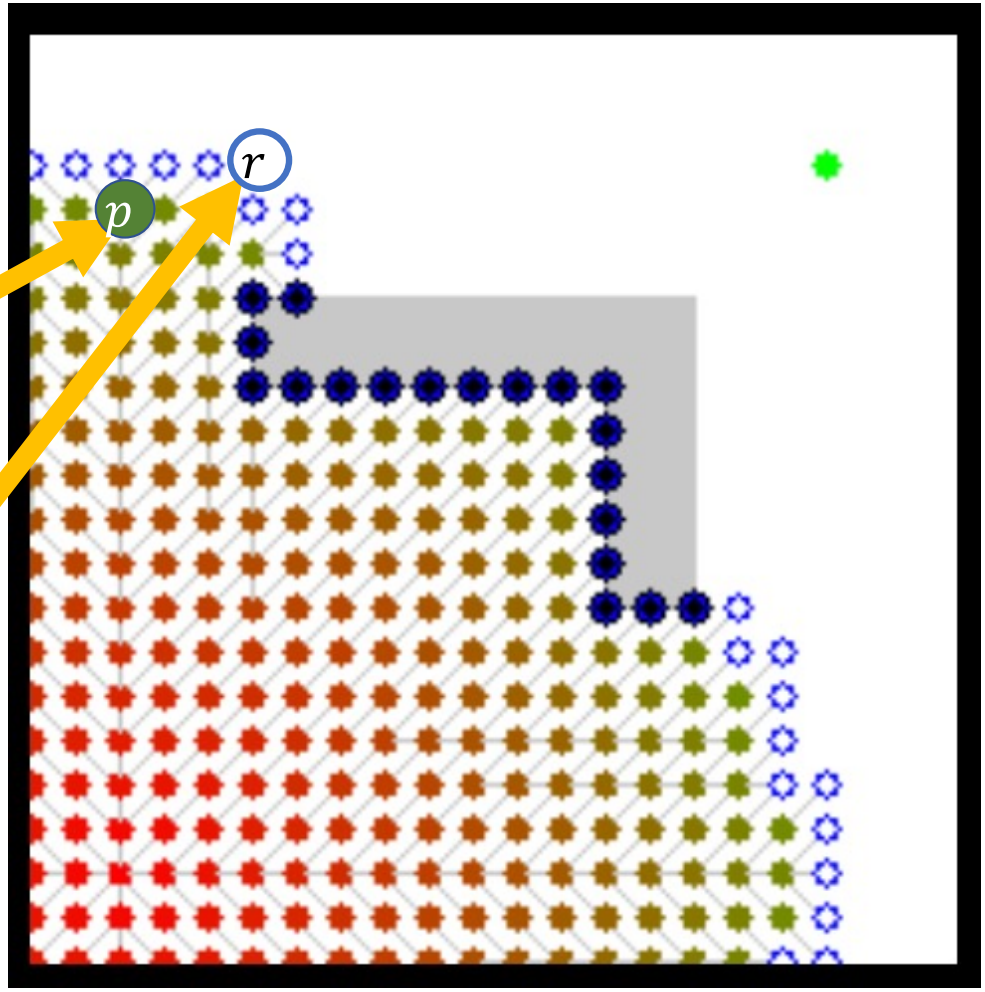
Why was Dijkstra slow?

- 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?

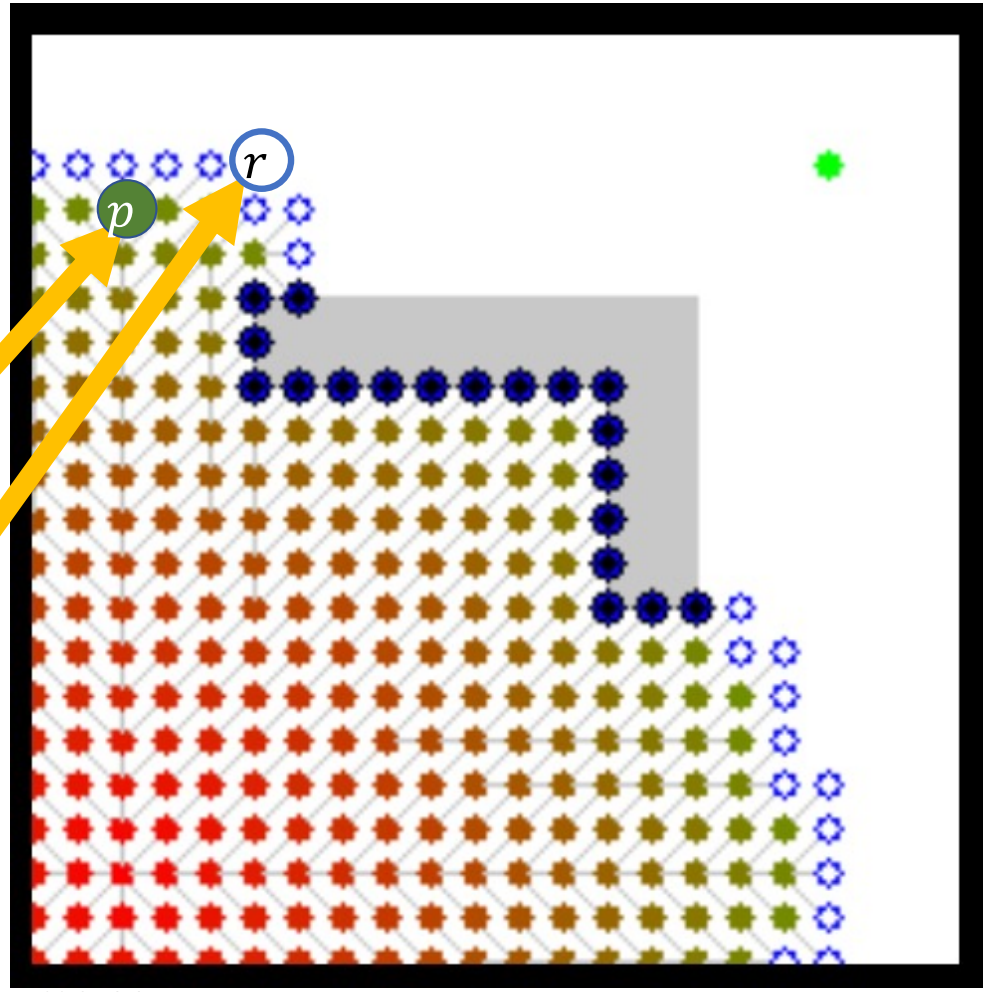
- This node has

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

- 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 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 **total length of the path through node p** is

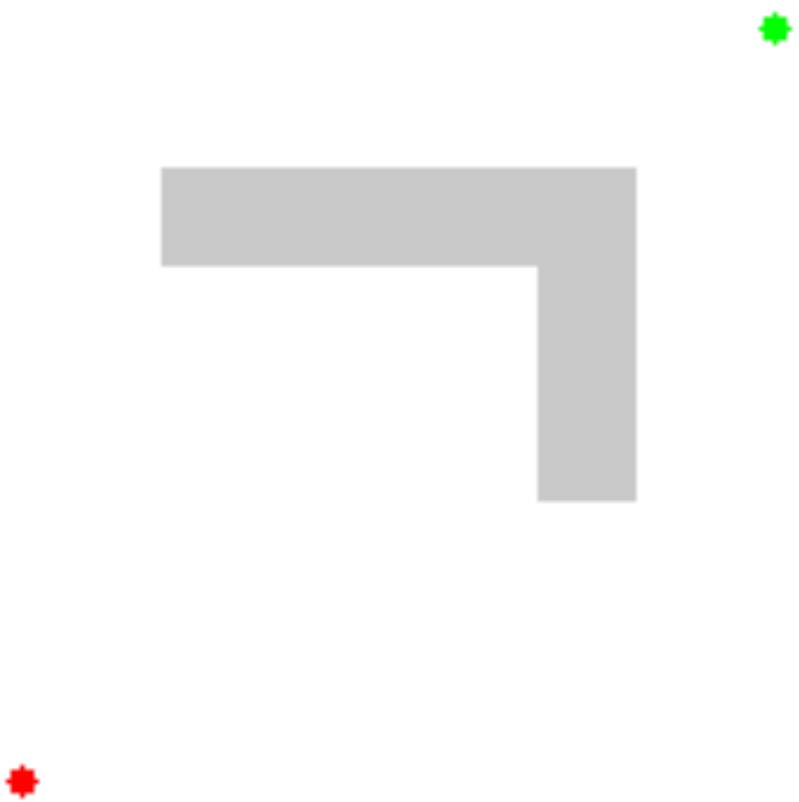
$$f(p) = g(p) + h(p) \approx d(Start, p) + d(p, Goal)$$

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
- Admissible search
- How to design a heuristic
- Consistent search

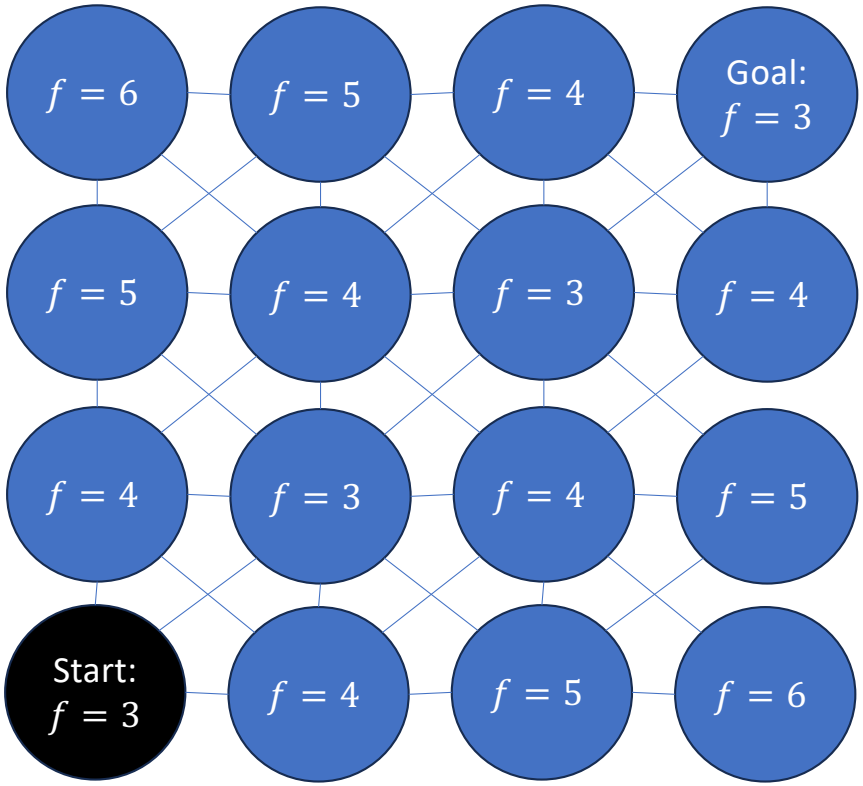
A* search: Key concepts

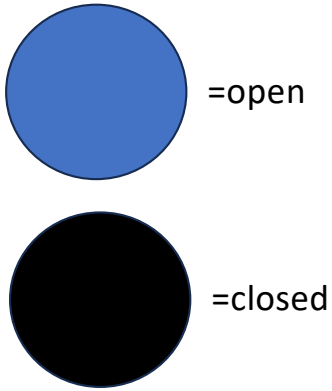
- $f(n) =$ distance of the shortest path from start to goal that passes through node n
- $g(n) =$ distance of the shortest path from start to node n
- $h(n) =$ distance of the shortest path from node n to goal

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

A* search: Key concepts

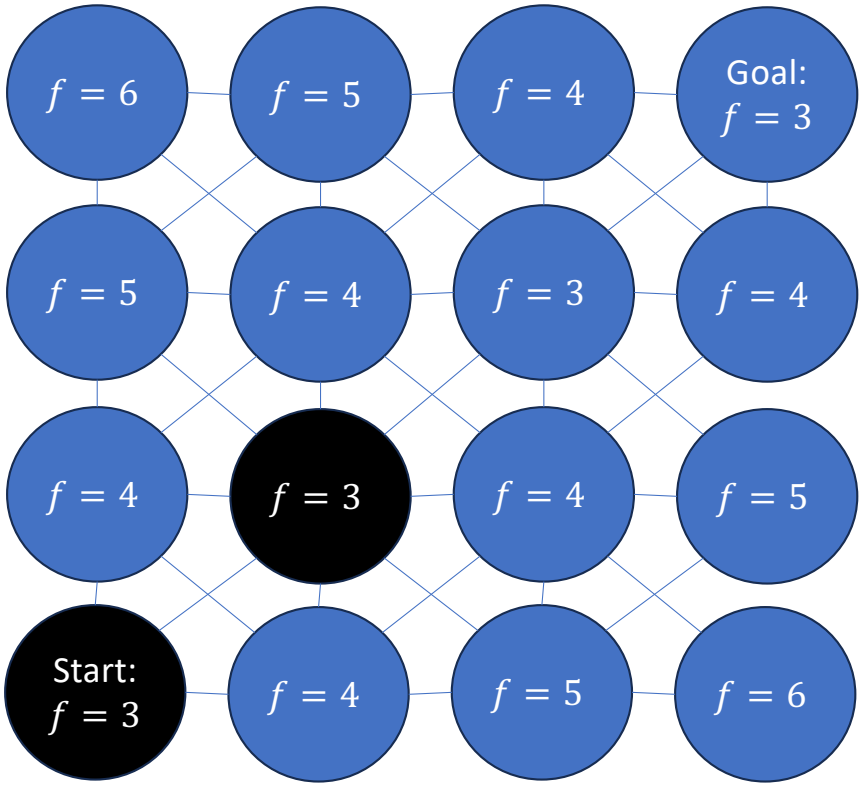
If nodes are sorted in increasing order of $f(n)$, then A* finds the shortest path to goal without evaluating any extra nodes.

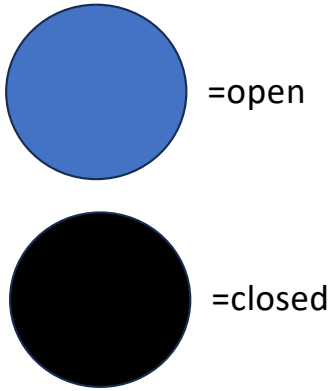




A* search: Key concepts

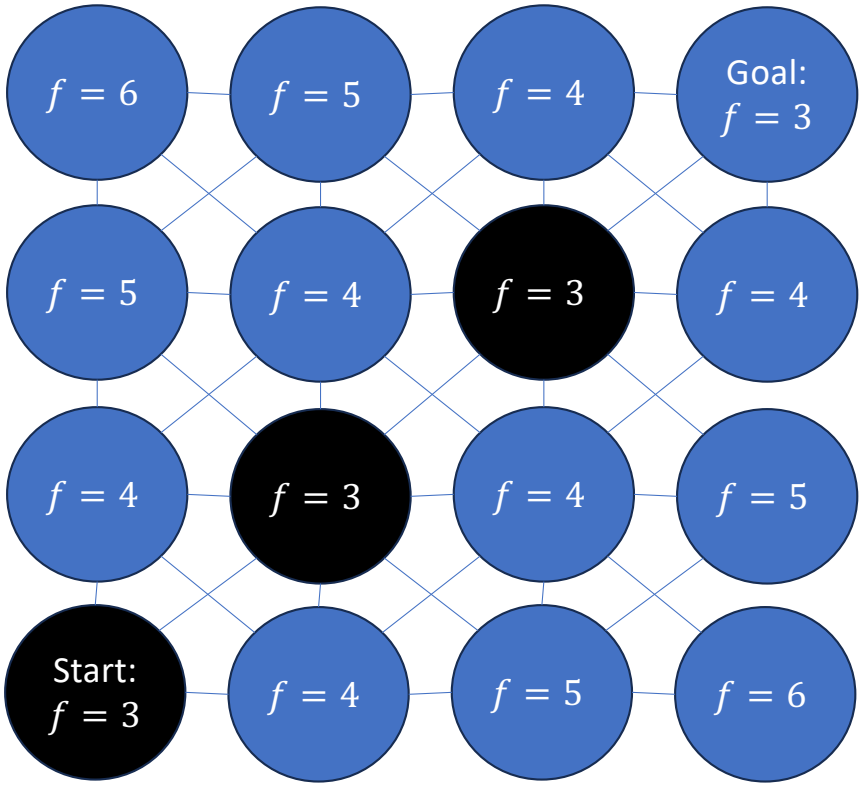
If nodes are sorted in increasing order of $f(n)$, then A* finds the shortest path to goal without evaluating any extra nodes.

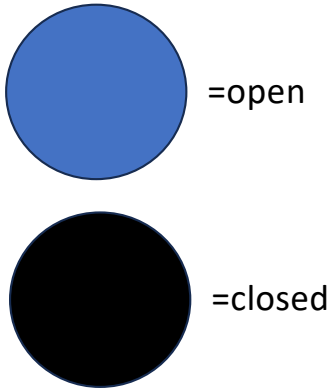




A* search: Key concepts

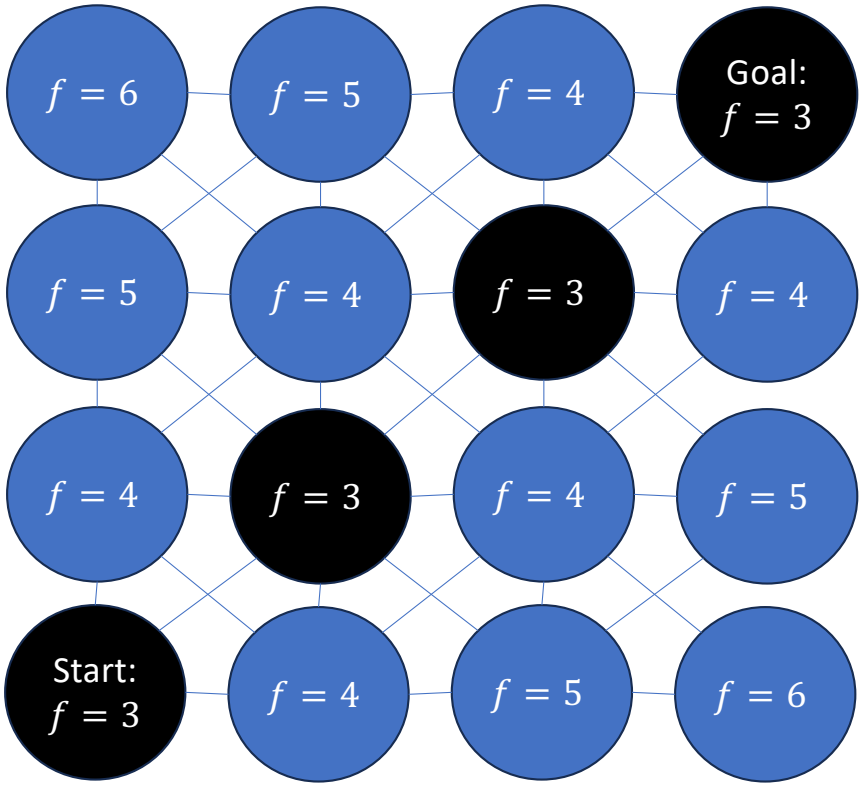
If nodes are sorted in increasing order of $f(n)$, then A* finds the shortest path to goal without evaluating any extra nodes.

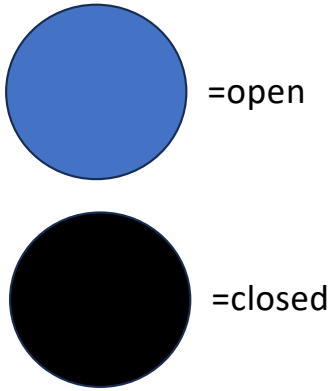




A* search: Key concepts

If nodes are sorted in increasing order of $f(n)$, then A* finds the shortest path to goal without evaluating any extra nodes.





The elephant in the room

The only way to know h(n) exactly is to solve the search problem.

A* search: Key concepts

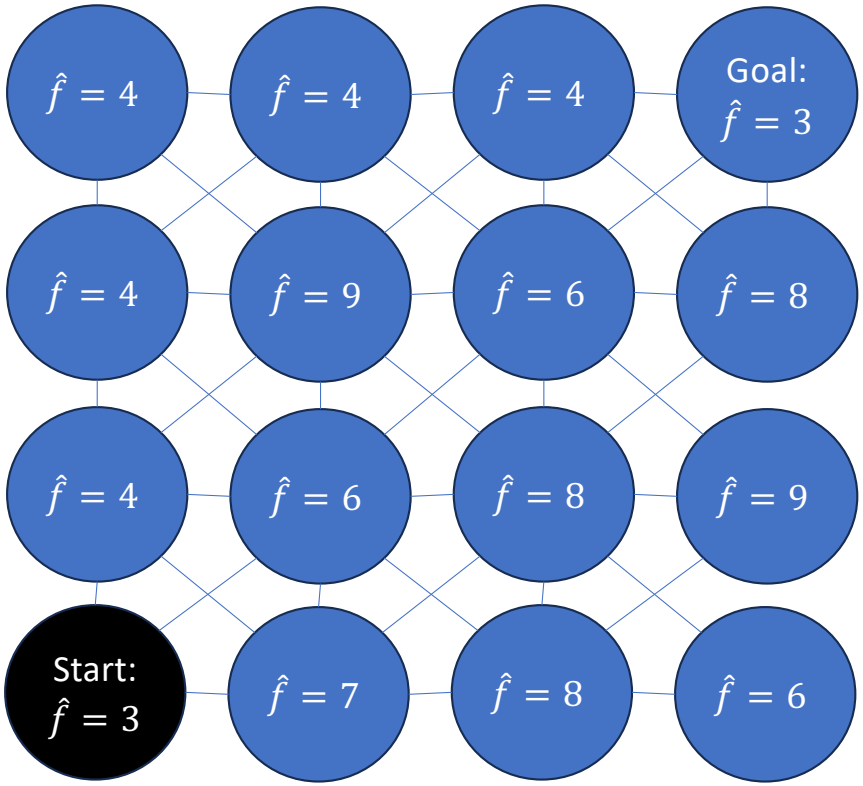
- $\hat{f}(n) =$ estimate of the distance of the shortest path from start to goal that passes through node n
- $g(n) =$ distance of the shortest path from start to node n
- $\hat{h}(n) =$ estimate of the distance of the shortest path from node n to goal

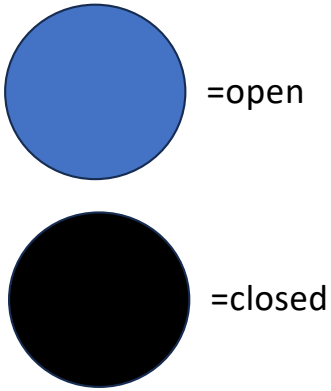
$$\hat{f}(n) = g(n) + \hat{h}(n)$$

A difficulty: what is $\hat{h}(n)$? Can it be just anything?

Can $\hat{h}(n)$ be just anything?

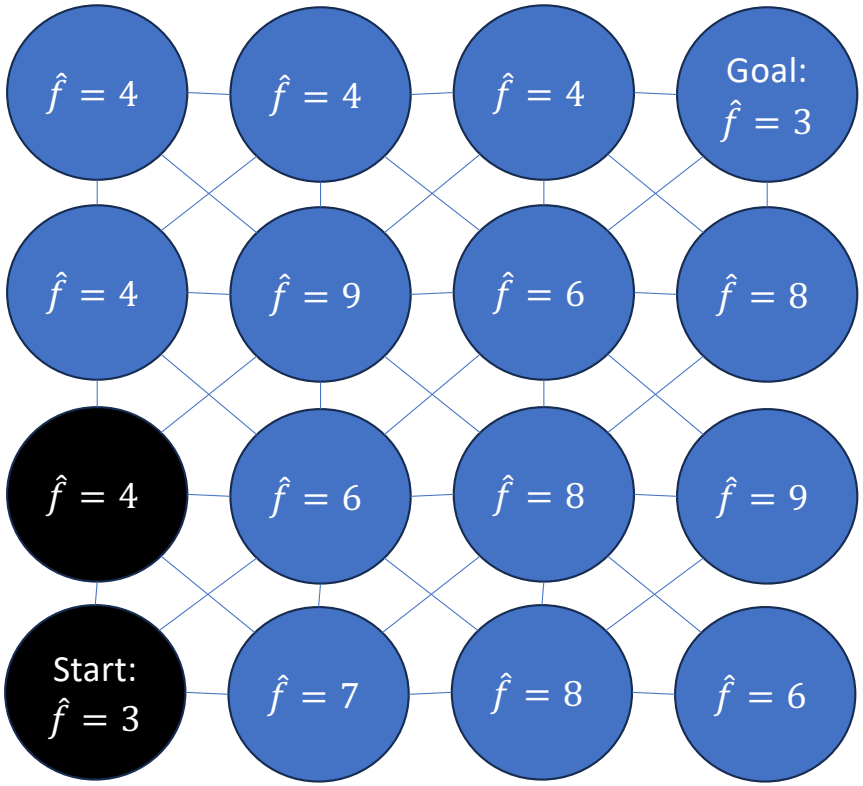
No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.

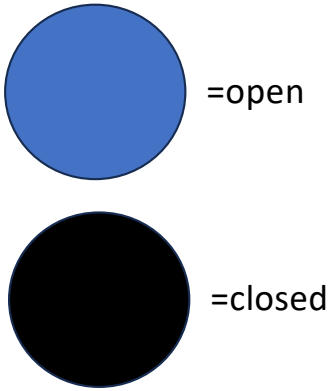




Can $\hat{h}(n)$ be just anything?

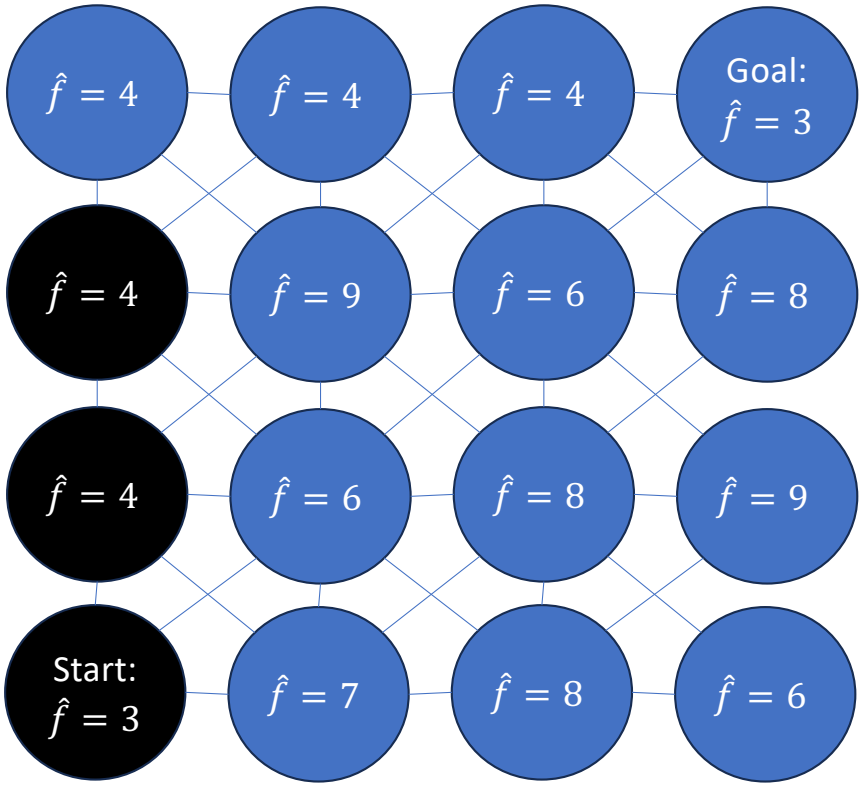
No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.

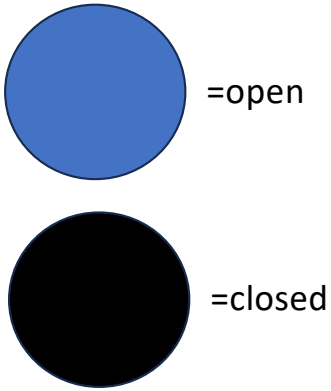




Can $\hat{h}(n)$ be just anything?

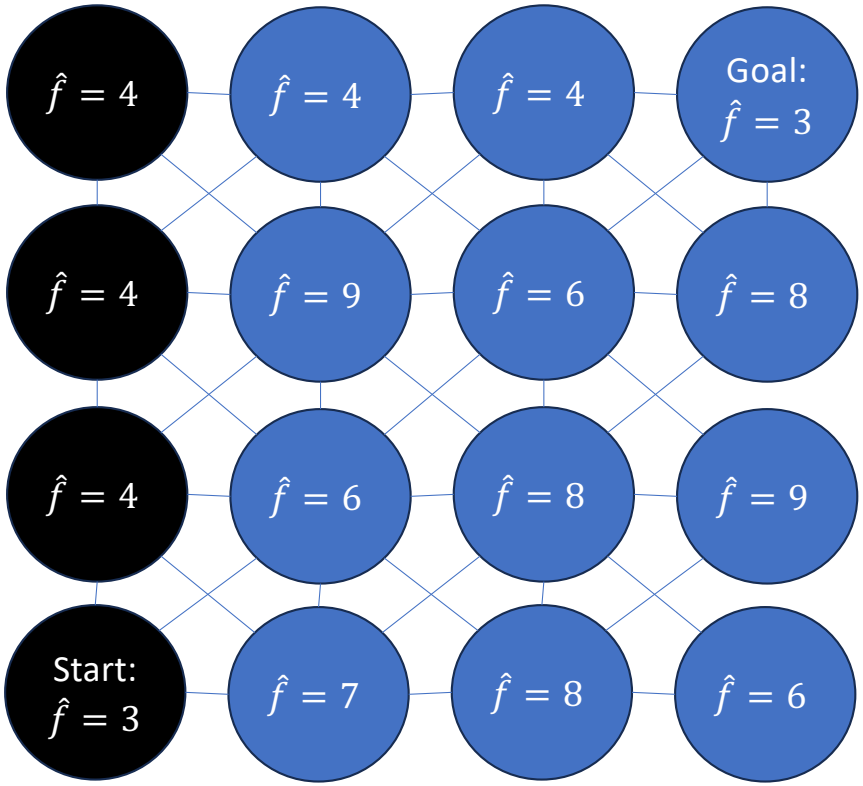
No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.

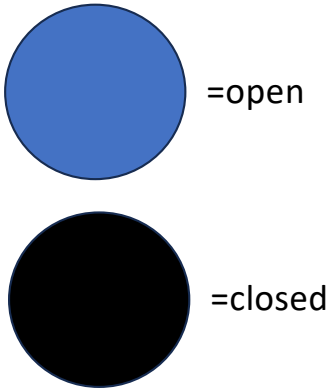




Can $\hat{h}(n)$ be just anything?

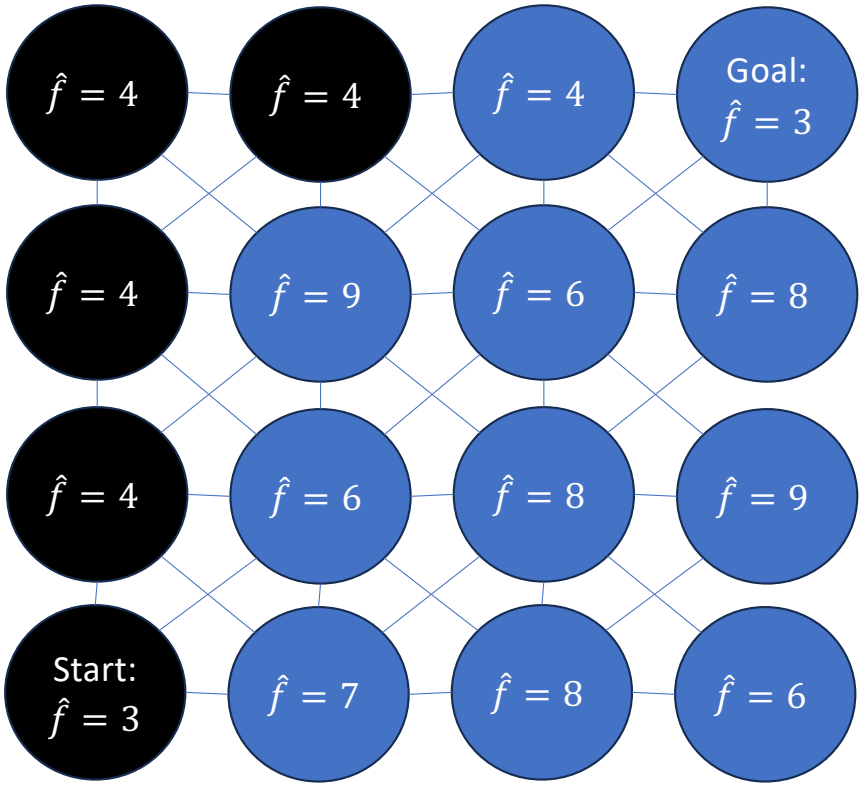
No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.

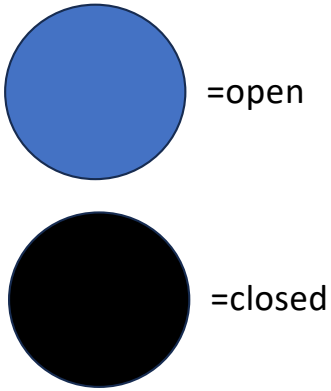




Can $\hat{h}(n)$ be just anything?

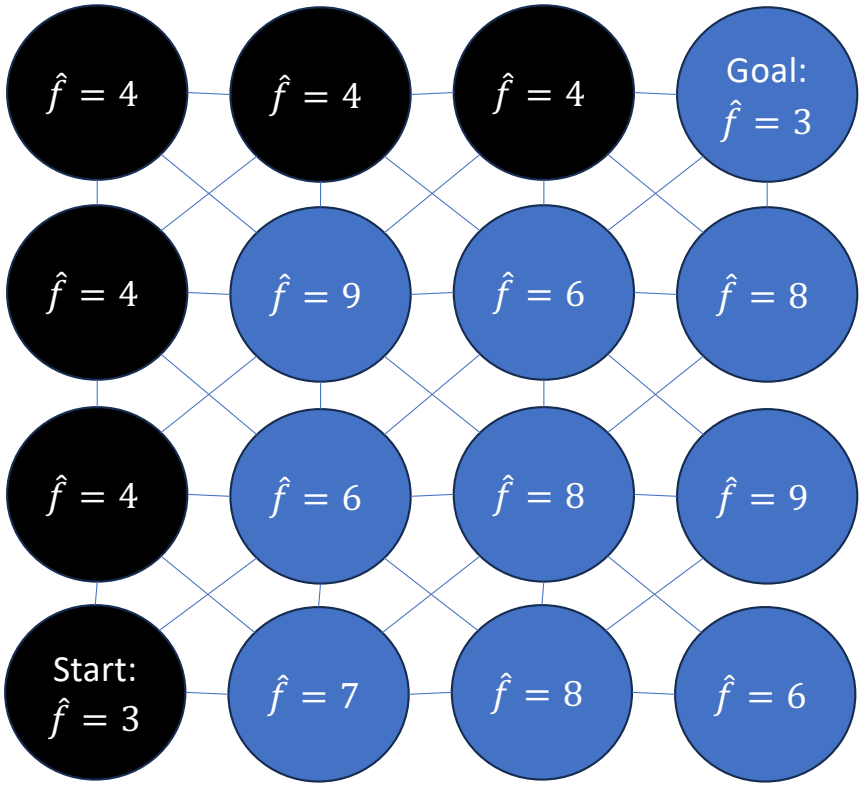
No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.

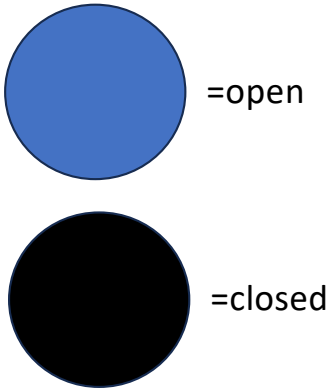




Can $\hat{h}(n)$ be just anything?

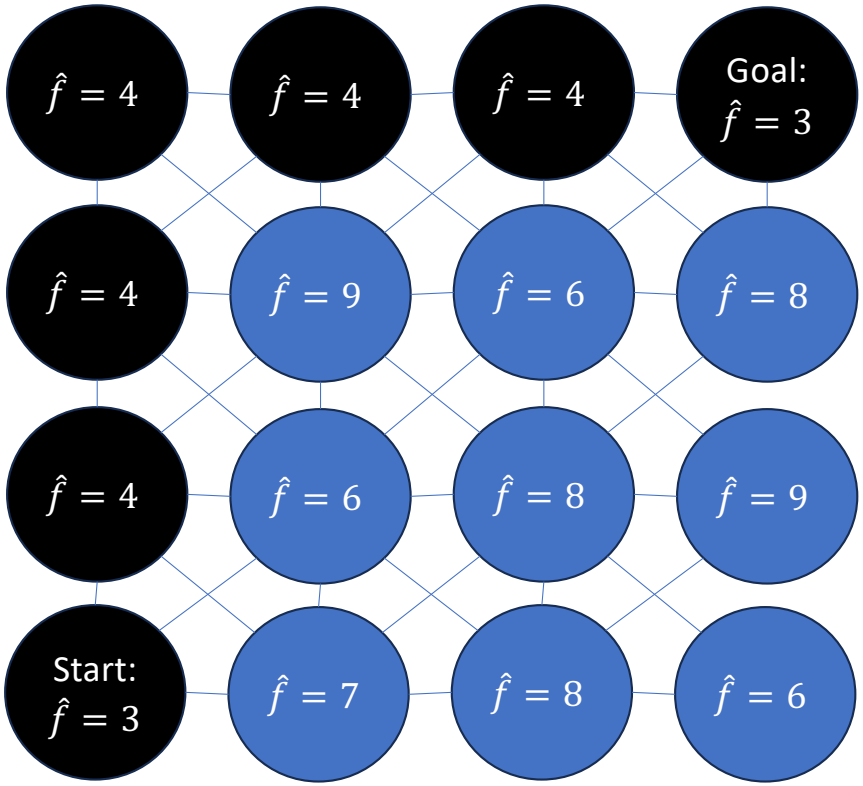
No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.

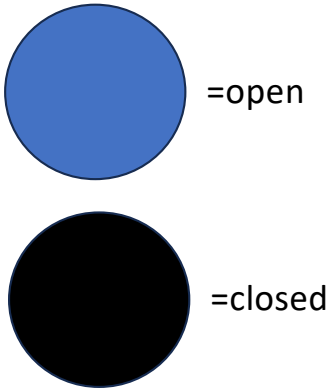




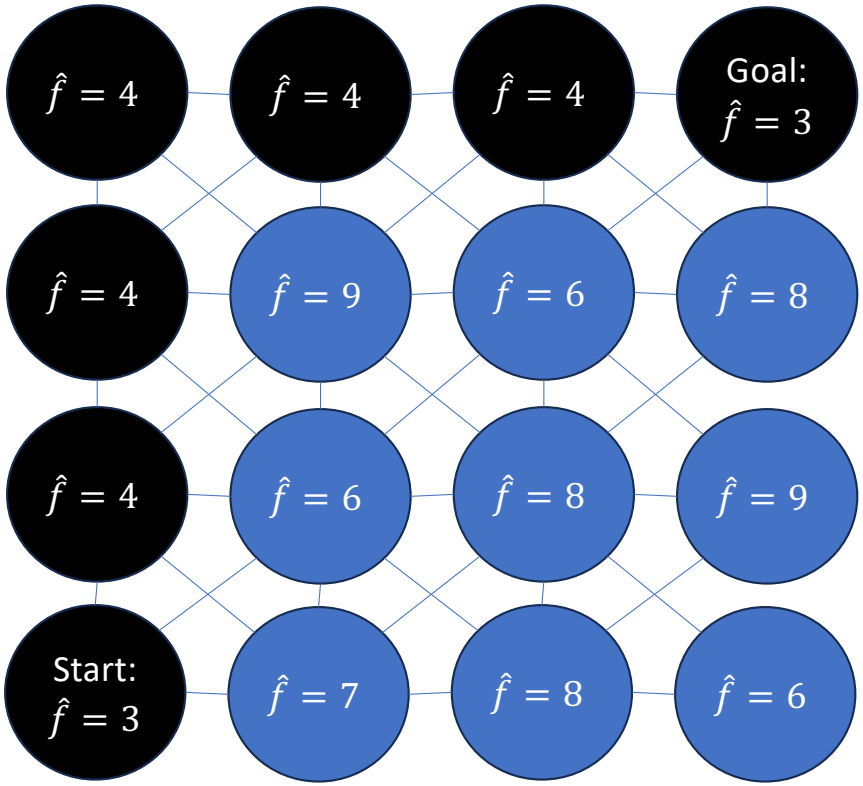
Can $\hat{h}(n)$ be just anything?

No. If $\hat{f}(n) = g(n) + \hat{h}(n)$ and we allow $\hat{h}(n)$ to be just anything, then the A* search algorithm is not admissible.





Why did it fail?



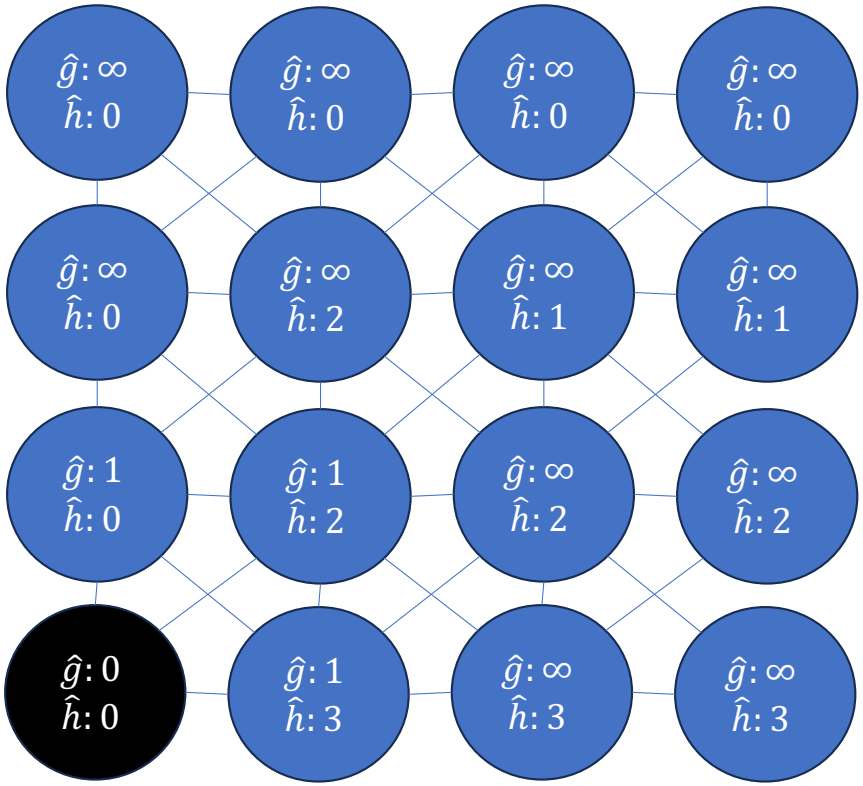
- $\hat{f}$ of the goal node was correctly estimated, but…
- …some of the nodes on the shortest path had unrealistically high values of $\hat{f}$, which prevented us evaluating the shortest path.
- Solution: Make sure that all nodes on the shortest path have $\hat{f}(n) \leq f(n) = f(\text{Goal})$

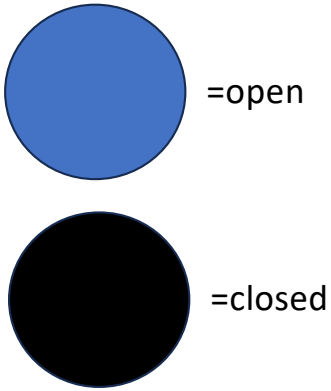
A* search: Key concepts

- $\hat{f}(n) = g(n) + \hat{h}(n)$ is estimated cost of shortest path from start to goal through n. If $\hat{f}(n) \leq f(n)$ for nodes on the shortest path, then those will get evaluated before the goal, and A* will find the shortest path.
- If we can guarantee $\hat{h}(n) \leq h(n)$ for all nodes, then:

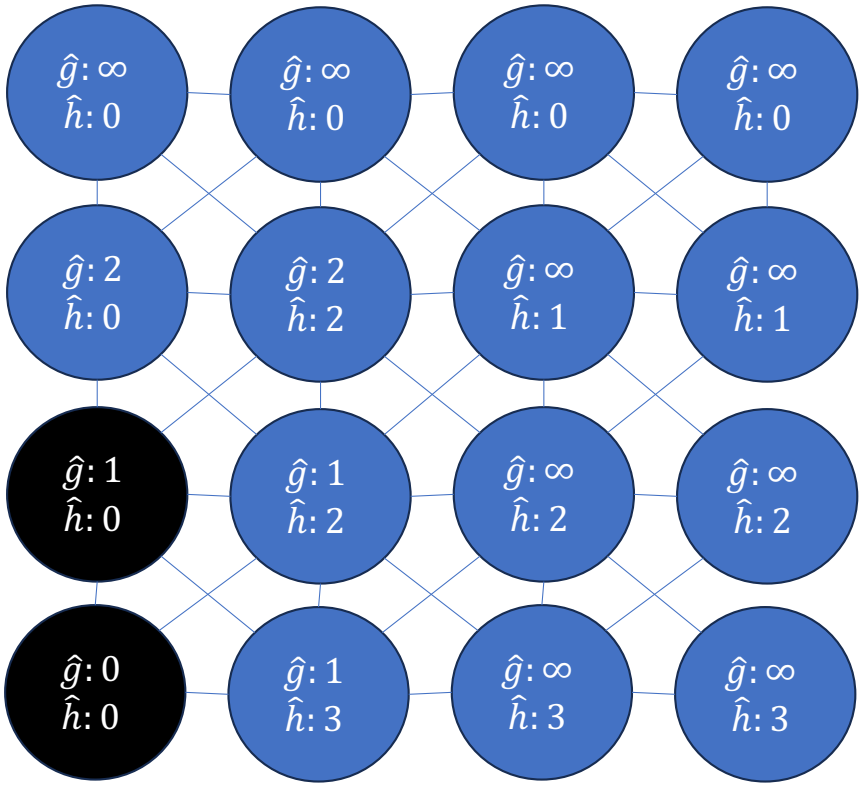
$$\hat{f}(n) = g(n) + \hat{h}(n) \leq g(n) + h(n) = f(n)$$

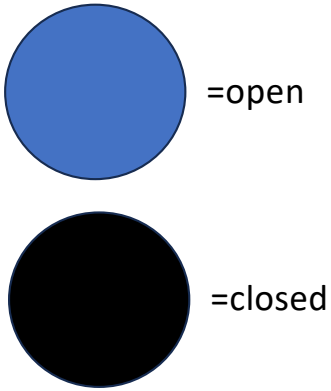
Admissibility: $\hat{h}(n) \leq h(n)$ guarantees $\hat{f}(n) \leq f(n)$



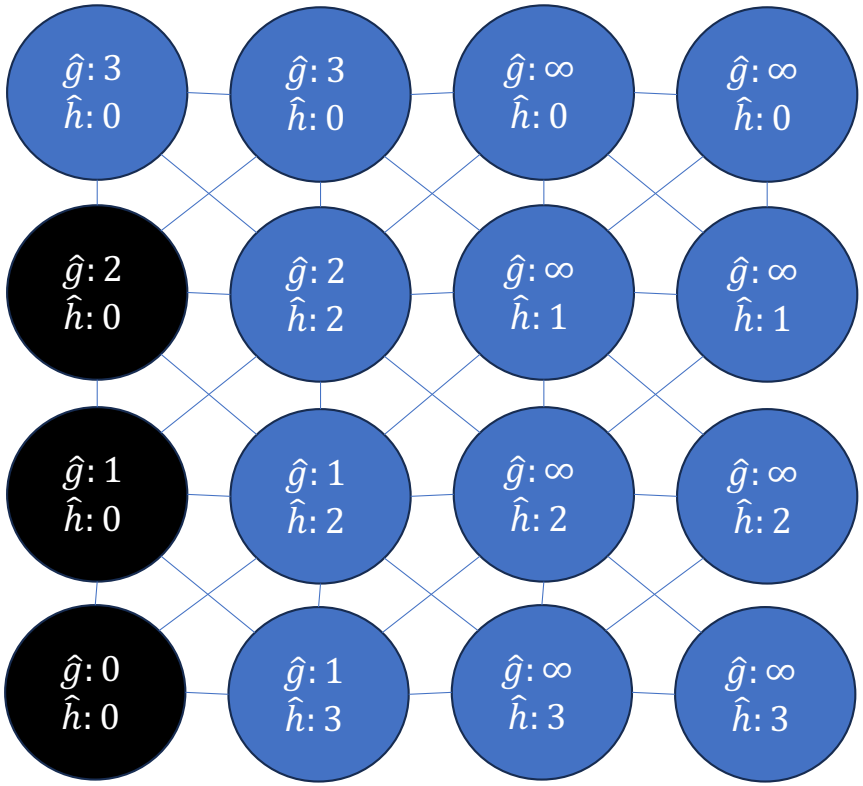


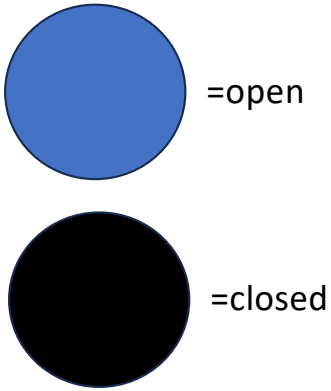
Admissibility: $\hat{h}(n) \leq h(n)$ guarantees $\hat{f}(n) \leq f(n)$



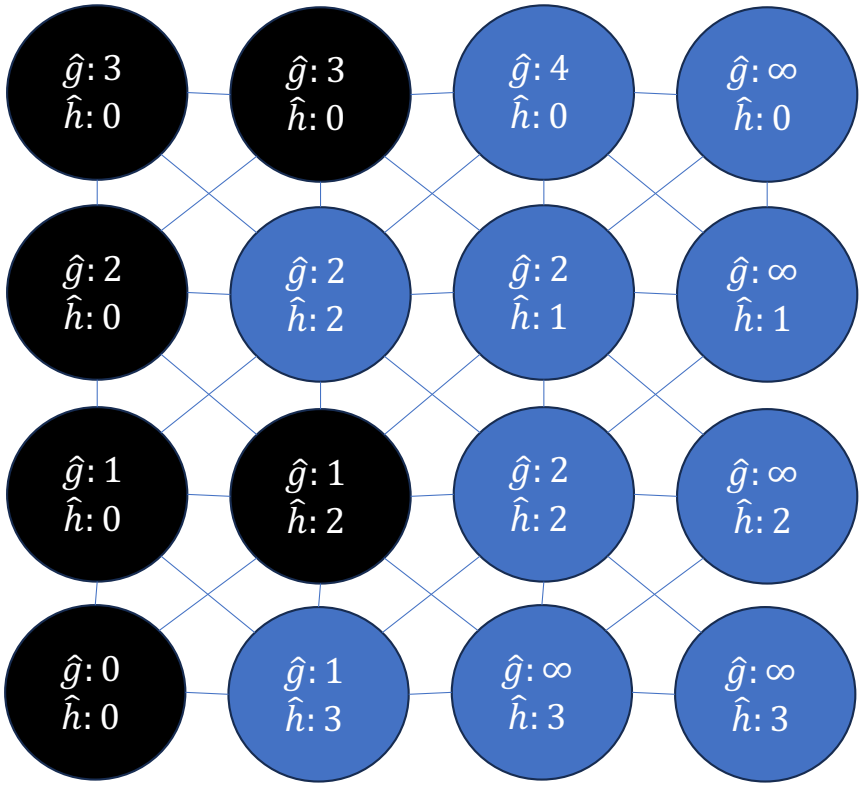


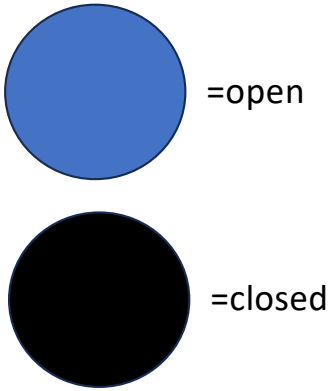
Admissibility: $\hat{h}(n) \leq h(n)$ guarantees $\hat{f}(n) \leq f(n)$



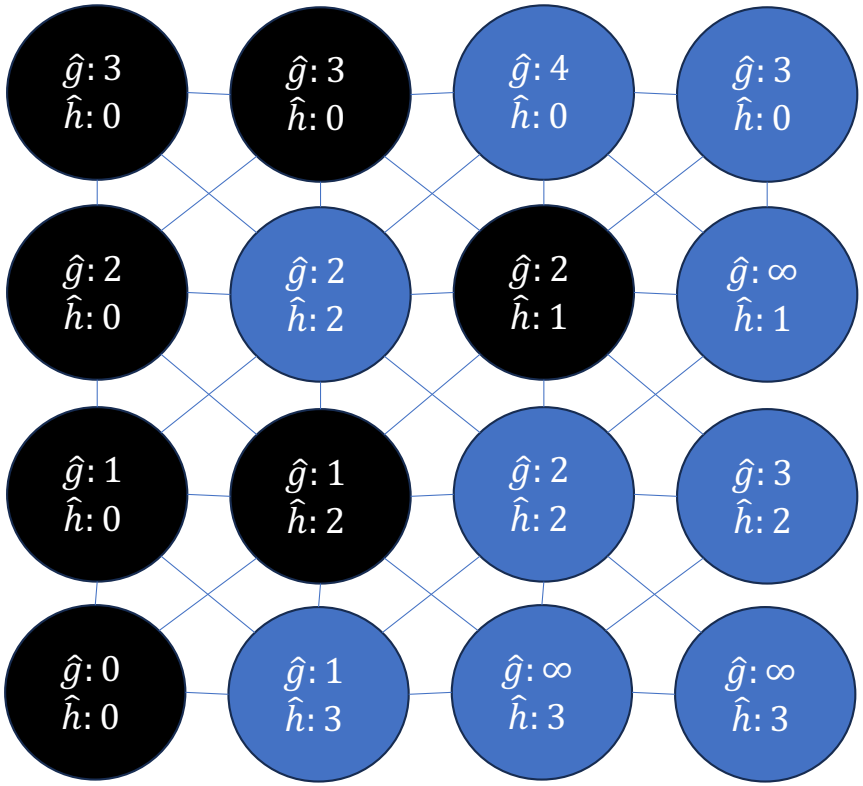


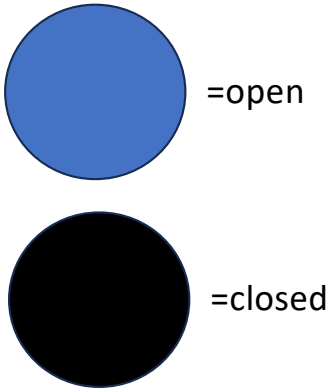
Admissibility: $\hat{h}(n) \leq h(n)$ guarantees $\hat{f}(n) \leq f(n)$



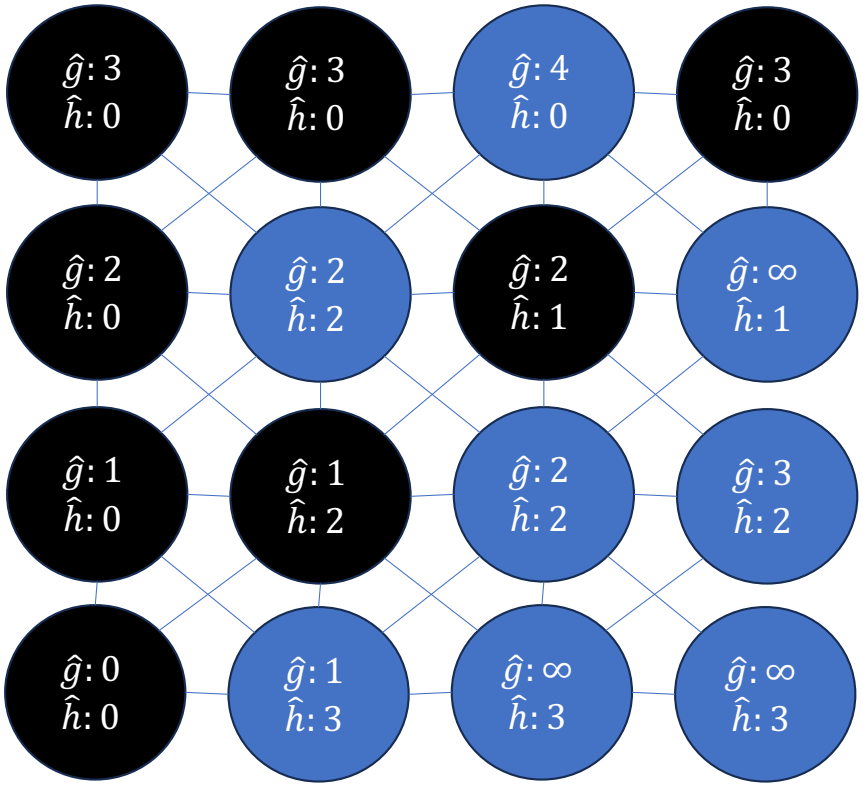


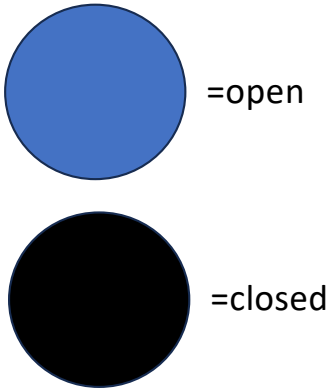
Admissibility: $\hat{h}(n) \leq h(n)$ guarantees $\hat{f}(n) \leq f(n)$





Admissibility: $\hat{h}(n) \leq h(n)$ guarantees $\hat{f}(n) \leq f(n)$





A* search algorithm

Initialize: compute $\hat{h}(n)$ for all nodes in some manner that guarantees $\hat{h}(n) \leq h(n)$. Set $\hat{g}(s) = 0$.

Iterate:

1. Choose the open node, n, with the lowest $\hat{f} = \hat{g} + \hat{h}$.
2. If $n \in$Goal, terminate. We have found the shortest path!
3. For all neighbor nodes $m \in \Gamma(n)$:
    1. If m is still open, set $\hat{g}(m) = \min(\hat{g}(m), \hat{g}(n) + d(n,m))$
    2. If m is already closed but $\hat{g}(n) + d(n,m) < \hat{g}(m)$, re-open it

Try the quiz!

Contents

- A* search: Using a heuristic to help choose which node to expand
- Admissible search
- How to design a heuristic
- Consistent search

Heuristics: Easy and Hard

- The easy heuristic: Notice that $\hat{h}(n) = 0$ always satisfies $\hat{h}(n) \leq h(n)$!
  - Using $\hat{h}(n) = 0$ means sorting by $\hat{f}(n) = g(n) + 0 = g(n)$
  - This is Dijkstra's algorithm!
- All other heuristics are hard
  - Must be designed for each search problem, separately, based on your knowledge about the problem
  - Is it worth it?

Computational complexity of A*

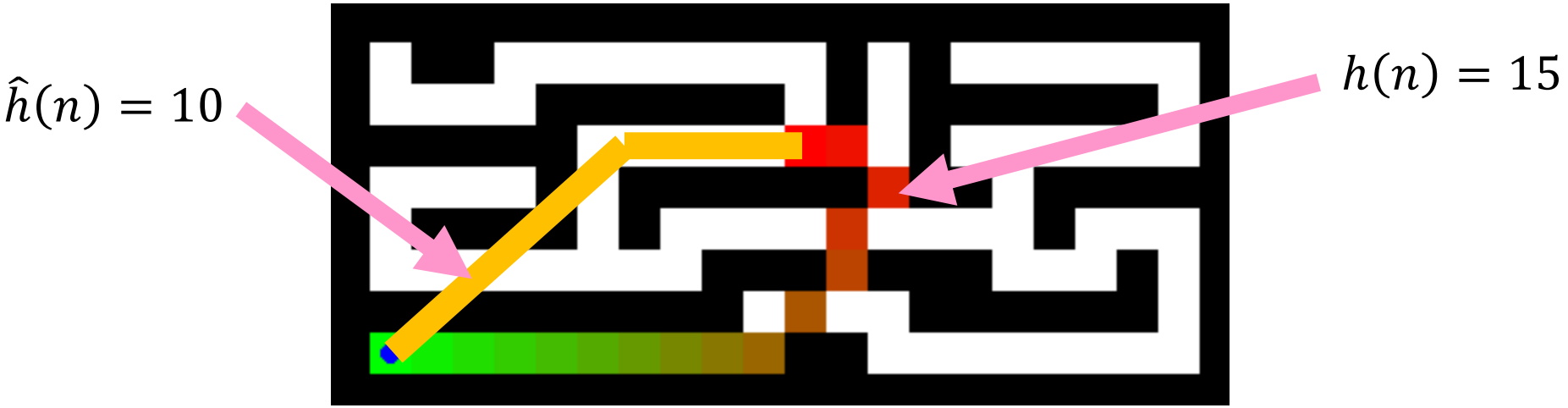
Notice: the number of nodes we expand is the number that have $\hat{f}(n) < g(\text{Goal})$. The larger $\hat{h}(n)$ is, the less computation will be required to find the optimal path. Therefore, we want to design $\hat{h}(n)$ to be as large as possible, subject to the constraints that

- We can somehow prove that $\hat{h}(n) \leq h(n)$
- Calculating $\hat{h}(n)$ for all nodes is less computationally expensive than it would be to just use Dijkstra's algorithm

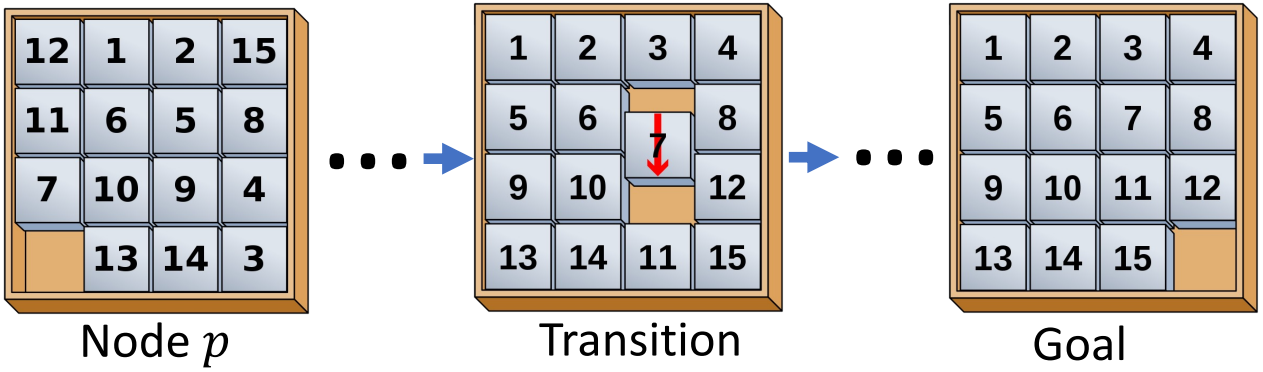
Designing a heuristic by relaxing constraints

- Often, good heuristics are computed by eliminating the constraints that make the problem hard
- Maze Example: If we just eliminate the walls, then $h(n)$ would equal the Chebyshev distance from n to Goal, i.e.,

$$\hat{h}(n) = \max(|x_{\text{Goal}} - x_n|, |y_{\text{Goal}} - y_n|)$$

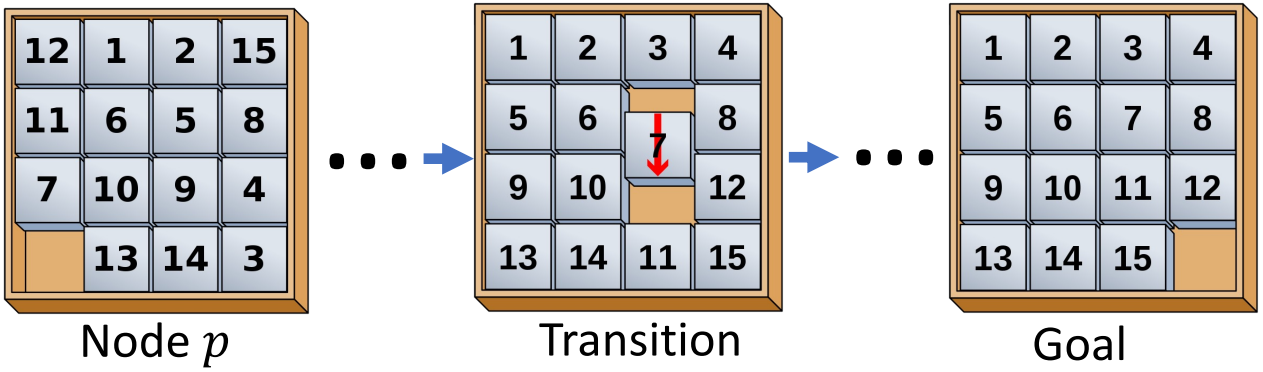


Example: the 15-puzzle



- 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



Node p Transition Goal

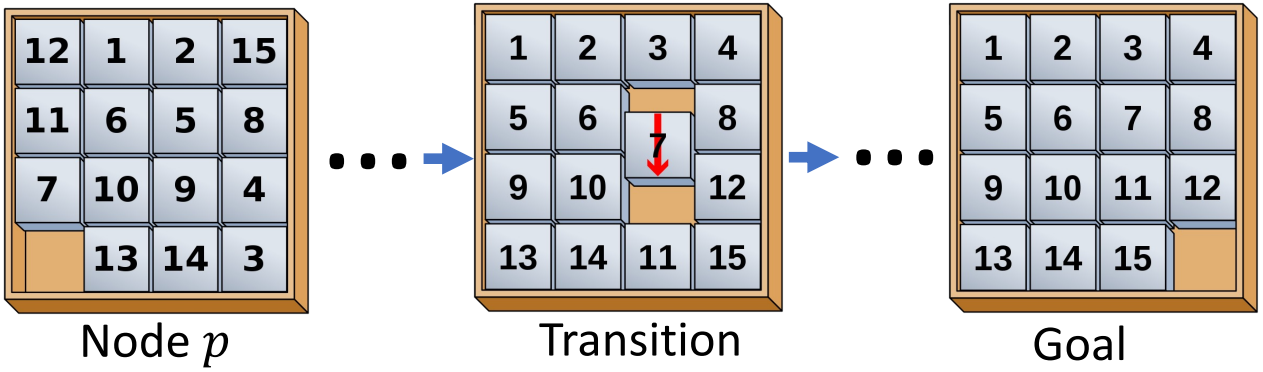
We can design a heuristic (which makes A* search much faster) by just ignoring the constraint.

$$\hat{h}(n) = \sum_{\text{tile}=1}^{15} \begin{array}{c}\#\text{squares tile would}\\ \text{have to move if there were no}\\ \text{other tiles in the way}\end{array}$$

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

$$\hat{h}(n) \leq h(n)$$

Example: the 15-puzzle



Node p Transition Goal

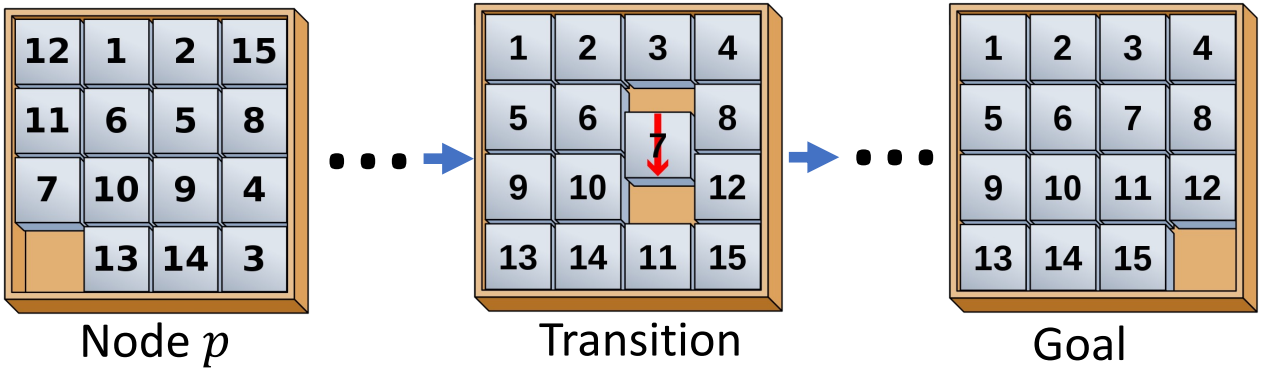
BFS solution of the 15-puzzle expands 54,000,000,000 nodes.

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

i.e., 0.000003% of the computational cost.

$$\hat{h}(n) = \sum_{\text{tile}=1}^{15} \begin{array}{c}\#\text{squares tile would}\\ \text{have to move if there were no}\\ \text{other tiles in the way}\end{array}$$

Cost of A* search



The cost of A* search is still $\mathcal{O}\{b^d\}$ in the ***worst case***, but the heuristic means that we only expand nodes with $\hat{f}(n) < g(\text{Goal})$, which, in the ***typical case***, may be much fewer than b^d.

- In this example, BFS has complexity $\mathcal{O}\{4^d\}$.
- A* has complexity close to $(1.13)^d$.

Contents

- A* search: Using a heuristic to help choose which node to expand
- Admissible search
- How to design a heuristic
- Consistent search

A* search algorithm

What if we don't want to have to do this?

Initialize: compute $\hat{h}(n)$ for all nodes in some manner that guarantees $\hat{h}(n) \leq h(n)$. Set $\hat{g}(s) = 0$.

Iterate:

1. Choose the open node, n, with the lowest $\hat{f} = \hat{g} + \hat{h}$.
2. If $n \in$Goal, terminate. We have found the shortest path!
3. For all neighbor nodes $m \in \Gamma(n)$:
    1. If m is still open, set $\hat{g}(m) = \min(\hat{g}(m), \hat{g}(n) + h(n,m))$
    2. If m is already closed but $\hat{g}(n) + h(n,m) < \hat{g}(m)$, re-open it

Admissible vs. Consistent Heuristics

- Admissible heuristic: Guarantee that, the first time we find the Goal node, we will find it with minimum cost:

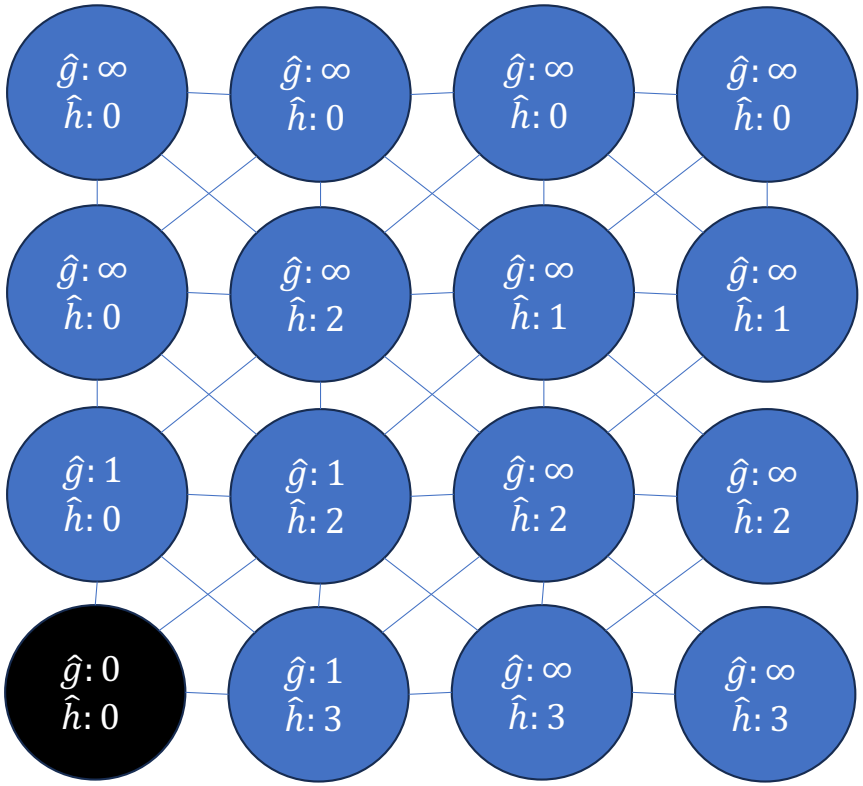
$$\hat{h}(n) \leq h(n)$$

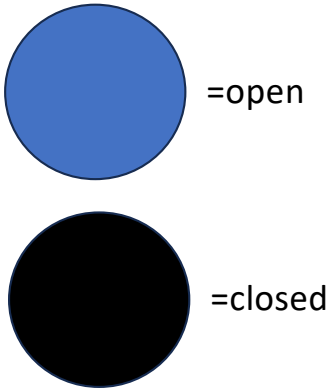
- Consistent heuristic: Guarantee that, the first time we find node m, we will find it with minimum cost:

$$\hat{h}(n) - \hat{h}(m) \leq h(n, m)$$

Example

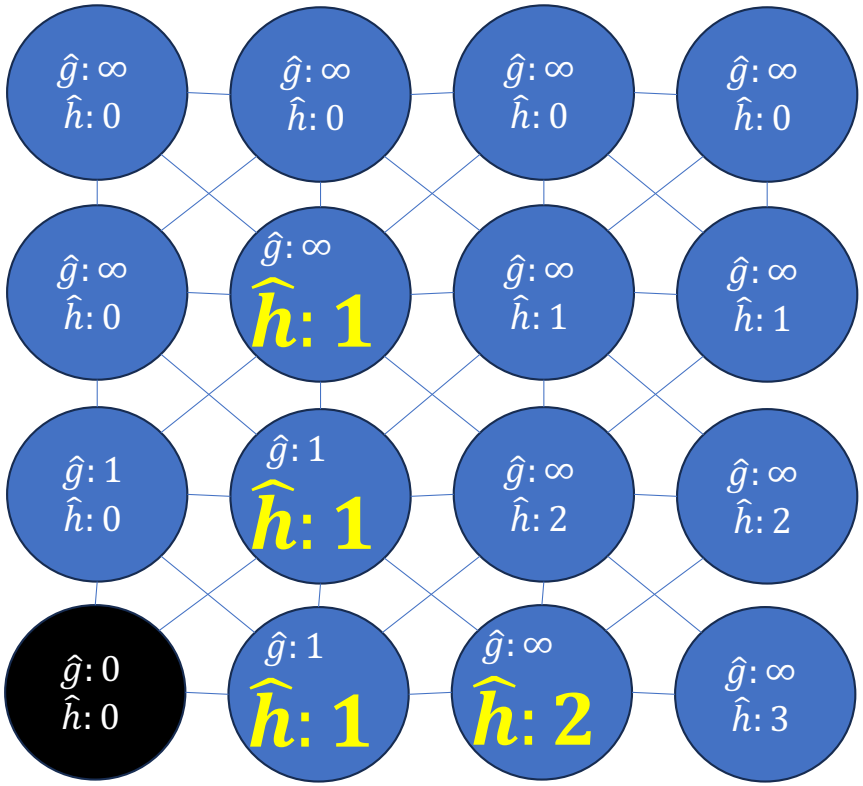
In this problem, all neighbors have $h(m,n) = 1$. This heuristic does not satisfy $\hat{h}(n) - \hat{h}(m) \leq h(n,m)$, so we might have to re-open nodes.

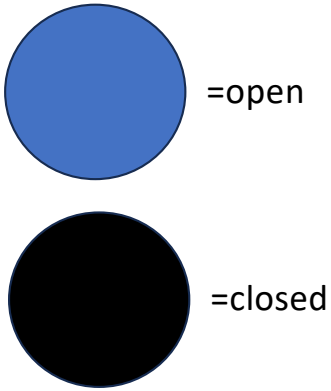




Example

In this problem, all neighbors have $h(m,n) = 1$. This heuristic sastisfies $\hat{h}(n) - \hat{h}(m) \leq h(n,m)$, so we will never have to re-open a closed node.





Conclusions

- A* is admissible if you use an admissible heuristic, and optimal (lower computation than any other exact search algorithm!) if you use a consistent heuristic.
- Admissible: $\hat{h}(n) \leq h(n)$
- Consistent: $\hat{h}(n) - \hat{h}(m) \leq h(n, m)$
- $\hat{h}(n) = 0$ is a valid heuristic (Dijkstra's algorithm), but usually we want to invent an $\hat{h}(n)$ as large as we can, subject to one of the two constraints above (depending on whether or not we want to re-open closed nodes).