Artificial Intelligence
3 – Solving problems by Searching

Arthur Bit-Monnot

INSA 4IR

Missionaries and Cannibals[1]

- Three missionaries and three cannibals are on one side of a river
- they have a boat that can hold one or two people.
- their goal is to get everyone to the other side
- but if the cannibals ever outnumber the missionaries on either side of the river, the missionaries will be eaten.

[1] Amarel, Saul (1968). Michie, Donald (ed.). "On representations of problems of reasoning about actions". Machine Intelligence.

Section 1

Search Problem

Search Problem

Focus: single agent, fully observable, deterministic, static, discrete

Components:

- **state space:** set of all possible states the environment can be in
- **initial state:** the state in which the agent starts
- **goal states**: one or several state that we want to reach
- **actions:** the set of actions that the agent can take in a given state
- **transition model:** a description of what each action does
- **action cost**: the cost of each action

Search Problem as a Graph

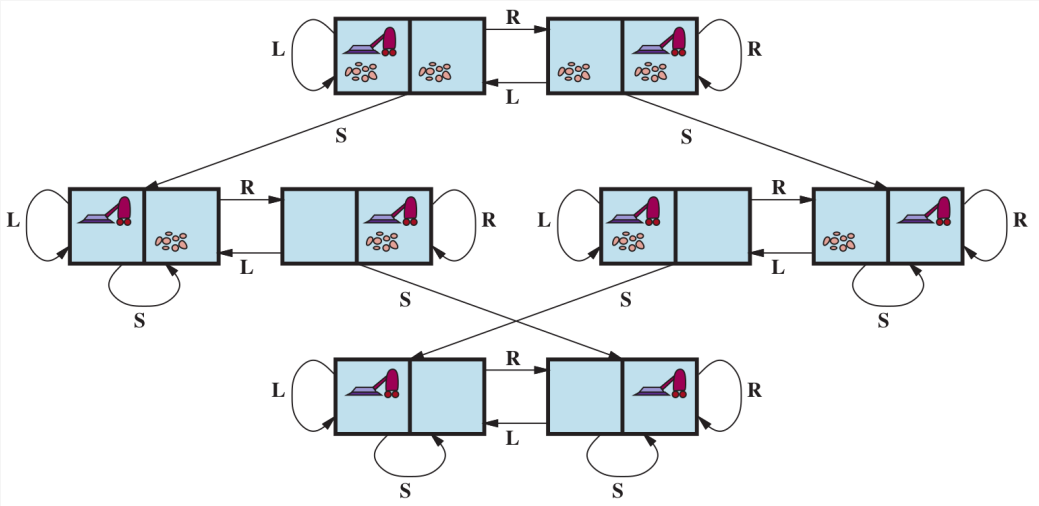
Graph:

- each **state** is a **node**
- each **action** is an **edge**:
  - source: state in which the action is taken
  - target: state after the action is taken (given by transition model)
  - label: action name (and cost)

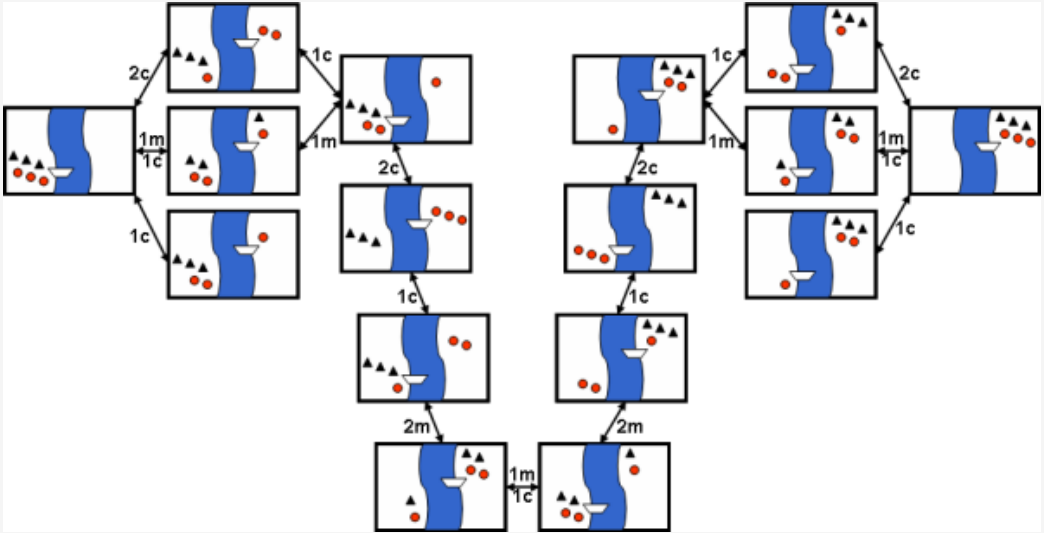
Search problem:

- find a path (~sequence of actions)
- from the initial state
- to one of the goal states

Search Problem: Vaccum world



Search Problem: Missionaries and Cannibals (source)



Easy: just Dijsktra the crap out of it!

Complexity: $O(N \times log(N) + E)$

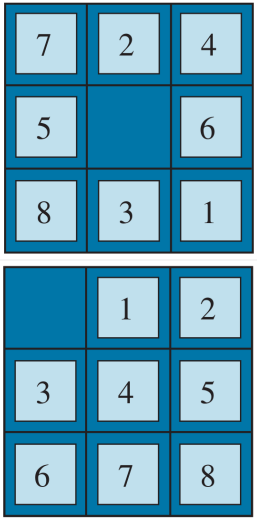
- N: number of nodes
- E: number of edges

POLYNOMIAL TIME!

Graph sizes:

- Vacuum world: 8 nodes, 8*3 edges
- Missionaries and Cannibals: 16 nodes, 34 edges

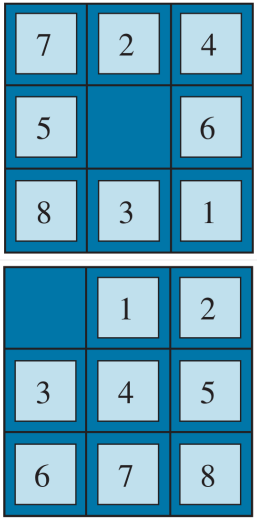
Search Problem: 8-puzzle



- state space: all possible configurations of the puzzle
- initial state: a configuration of the puzzle (e.g. on top left)
- goal state: configuration on bottom left
- actions: move a tile to the empty space
- action cost: 1

How many possible states are there?

Search Problem: 8-puzzle

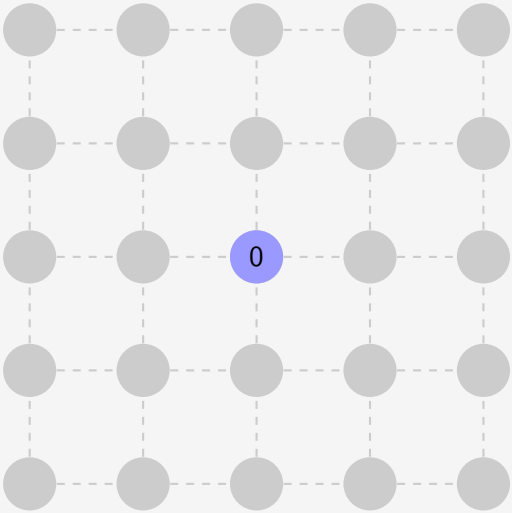


- state space: all possible configurations of the puzzle
- initial state: a configuration of the puzzle (e.g. on top left)
- goal state: configuration on bottom left
- actions: move a tile to the empty space
- action cost: 1

How many possible states are there?

- 8-puzzle: $9! = 362{,}880$
- 16-puzzle: $16! = 20{,}922{,}789{,}888{,}000$

Best-First Search visualization



Unseen

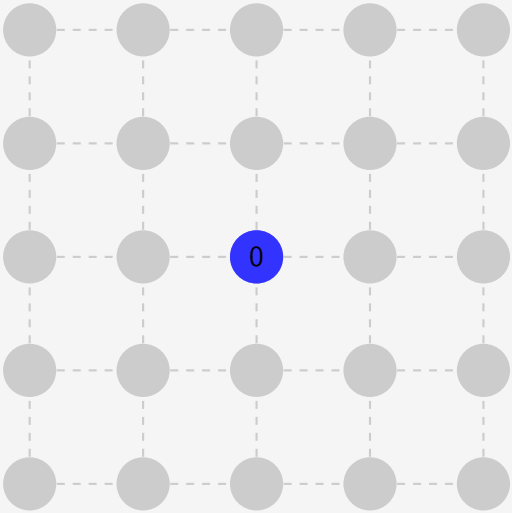
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

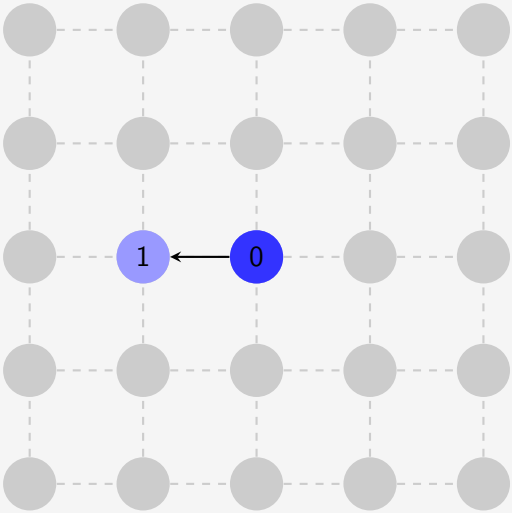
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

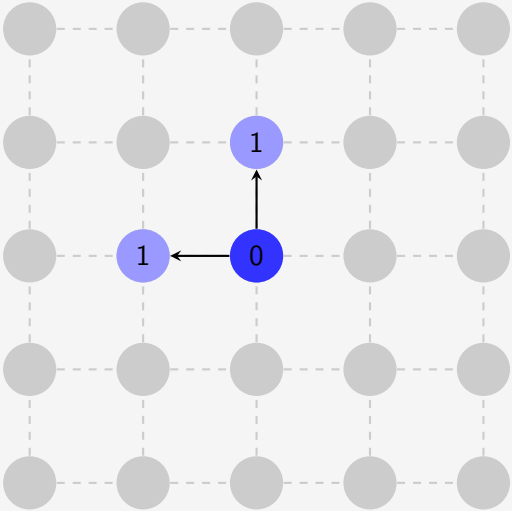
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

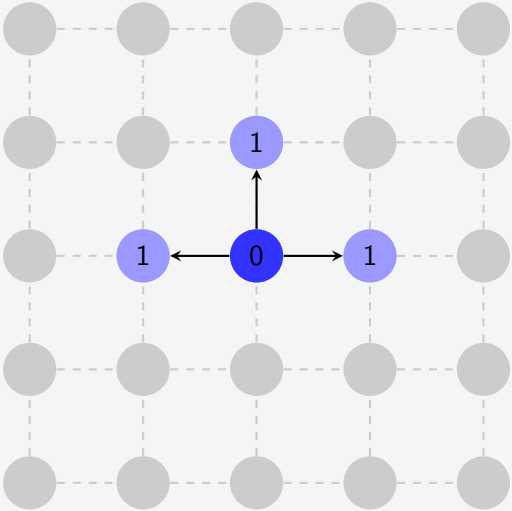
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

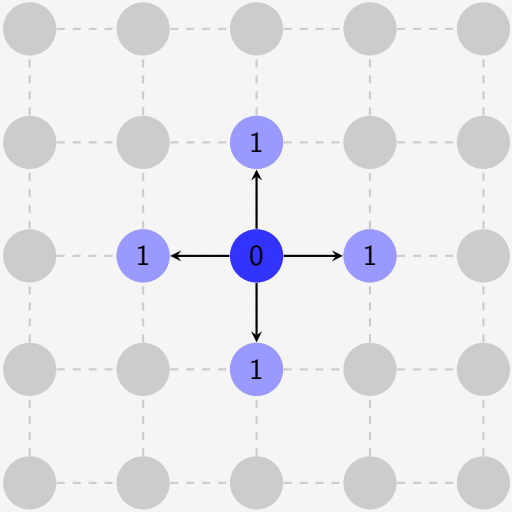
Expanded

Frontier

3 best cost to node

$\rightarrow$ last action on best path

Best-First Search visualization



Unseen

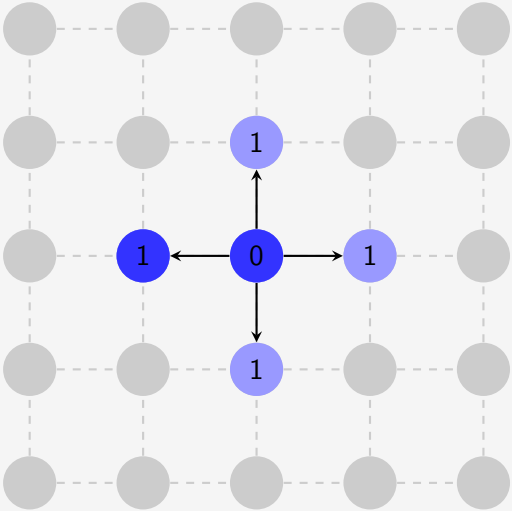
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

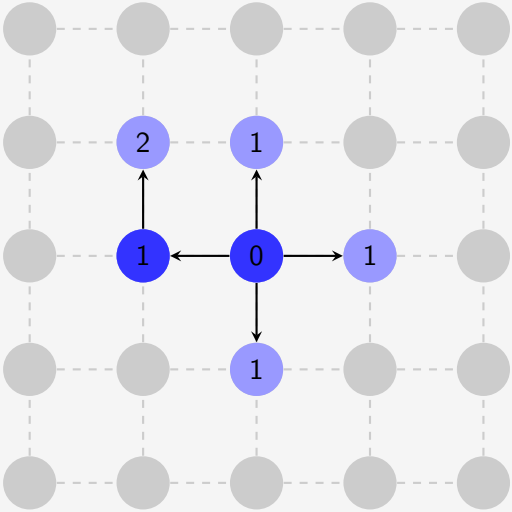
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

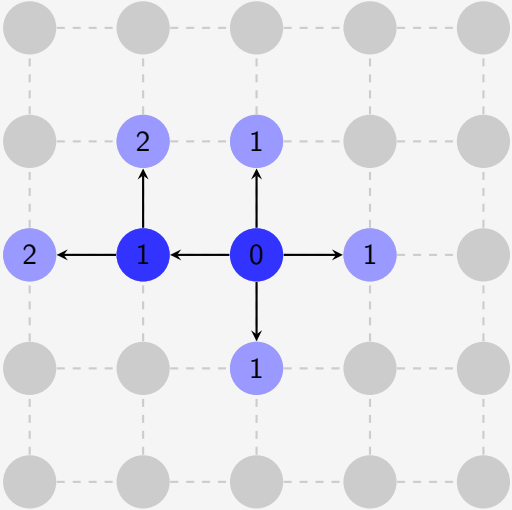
Expanded

Frontier

3 best cost to node

$\rightarrow$ last action on best path

Best-First Search visualization



Unseen

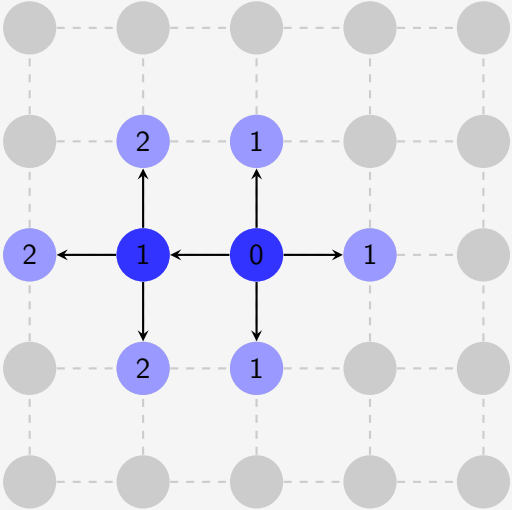
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

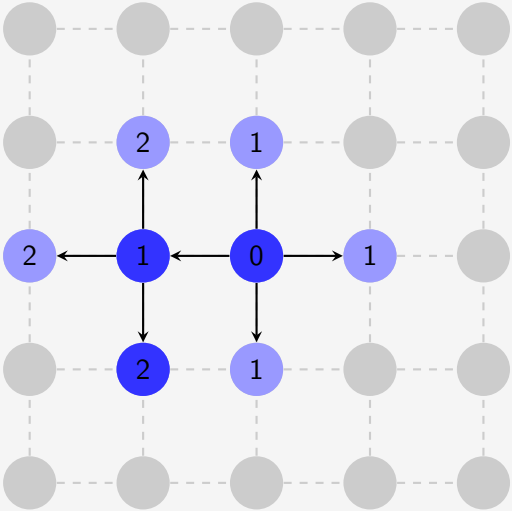
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

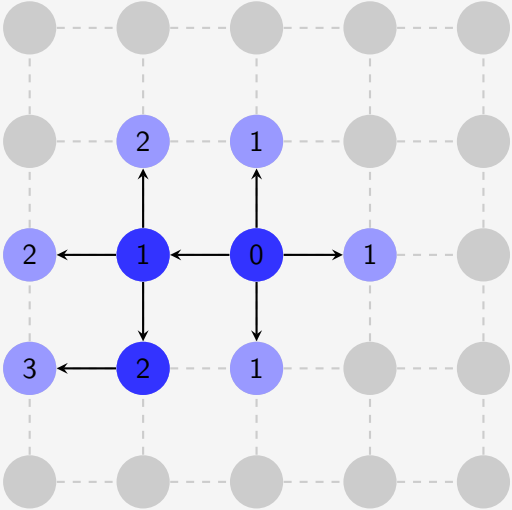
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

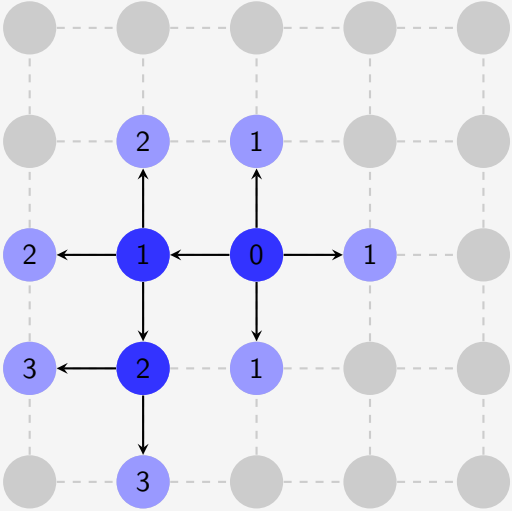
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

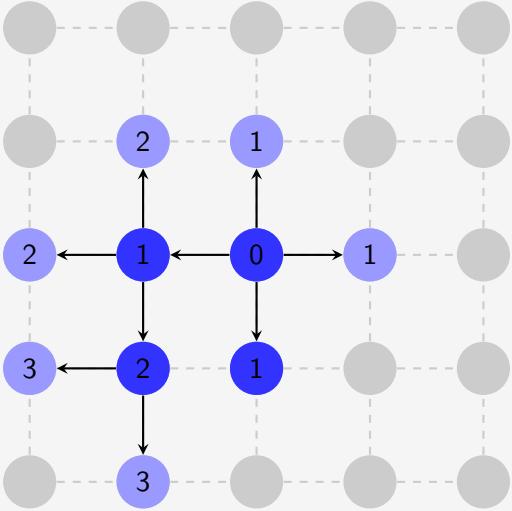
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

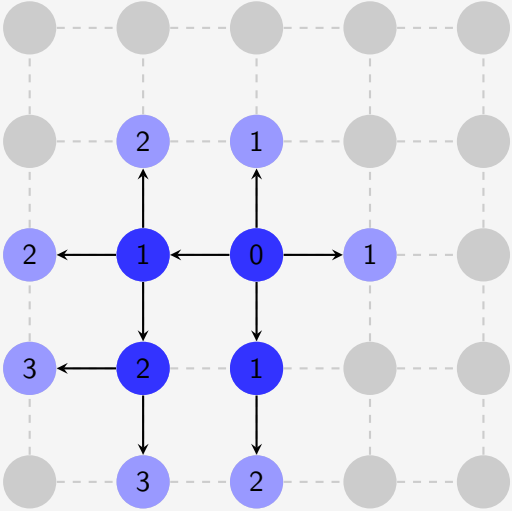
Expanded

Frontier

3 best cost to node

$\rightarrow$ last action on best path

Best-First Search visualization



Unseen

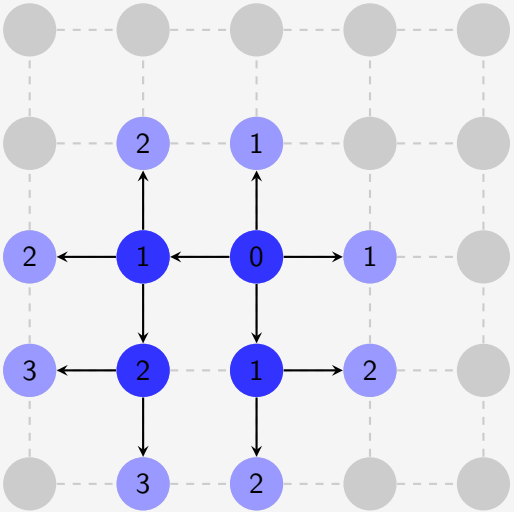
Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search visualization



Unseen

Expanded

Frontier

3 best cost to node

→ last action on best path

Best-First Search: algorithm

function BEST-FIRST-SEARCH(s_{init}, f)
  Frontier $\leftarrow \{(s_{init}, f(s_{init})\}$ // Priority queue, ordered by increasing f(s)
  Expanded $\leftarrow \emptyset$ // Nodes already expanded
  PathCost[s_{init}] $\leftarrow 0$
  while Frontier is not empty **do**
    $s \leftarrow$ node in Frontier with lowest $f(s)$
    if $s \in$ Expanded **then continue** // Already Expanded
    if ISGOAL(s) **then return** path from s_{init} to s (reconstructed from Predecessor)
    for $a \in$ ACTIONS(s) **do** // Actions applicable in s
      $s' \leftarrow$ RESULT(s, a) // Doing action a in s results in s'
      $c \leftarrow$ PathCost[s] + cost(s, a, s')
      if $s' \notin$ PathCost or $c <$ PathCost[s'] **then** // New path or better path to s'
        PathCost[s'] $\leftarrow c$ // Record best cost to s'
        Predecessor[s'] $\leftarrow (s, a)$ // Record how we got to s'
        Frontier $\leftarrow$ Frontier $\cup \{(s', f(s'))\}$ // Add s' to the queue
    Expanded $\leftarrow$ Expanded $\cup \{s\}$ // Mark s as expanded
  return failure (no path found)

Graph search without the Graph

We don't need to store the graph

Implicitly defined by the transition model:

- $\textsc{Action}(s)$: returns the set of actions applicable in state s (outgoing edges)
- $\textsc{Result}(s, a)$: returns the state resulting from doing action a in state s (target node)

The *Best* in Best-First Search

- Which node of the frontier is to be selected next?
  - i.e. what is the definition of $f(s)$?

[2] depth: number of *actions* from initial state

The *Best* in Best-First Search

- Which node of the frontier is to be selected next?
  - i.e. what is the definition of $f(s)$?

Some you already know:

- Breadth first search:[2]
  - $f(s) = \text{depth}(s)$
  - $\approx$ FIFO frontier

[2] depth: number of *actions* from initial state

The *Best* in Best-First Search

- Which node of the frontier is to be selected next?
  - i.e. what is the definition of $f(s)$?

Some you already know:

- Breadth first search:[2]
  - $f(s) = \text{depth}(s)$
  - $\approx$ FIFO frontier
- Depth first search:
  - $f(s) = -\text{depth}(s)$
  - $\approx$ LIFO frontier

[2] depth: number of *actions* from initial state

The *Best* in Best-First Search

- Which node of the frontier is to be selected next?
  - i.e. what is the definition of $f(s)$?

Some you already know:

- Breadth first search:[2]
  - $f(s) = \text{depth}(s)$
  - $\approx$ FIFO frontier
- Depth first search:
  - $f(s) = -\text{depth}(s)$
  - $\approx$ LIFO frontier
- Dijkstra:
  - $f(s) = \textsc{PathCost}(s)$

[2] depth: number of *actions* from initial state

Dijkstra's algorithm

assumption: the action cost is strictly positive $(c(a) > 0)$

Lets say we expand a state s and generate a new state s' with action a:

$$\begin{aligned} \text{PathCost}[s'] &= \text{PathCost}[s] + c(a) \\ &> \text{PathCost}[s] \end{aligned}$$

- when expanding a state with cost C,
  - all future states will have a cost of at least C
  - all previous states (expanded states) have a cost of at most C

Dijkstra's algorithm

assumption: the action cost is strictly positive $(c(a) > 0)$

Lets say we expand a state s and generate a new state s' with action a:

$$\begin{aligned} \text{PathCost}[s'] &= \text{PathCost}[s] + c(a) \\ &> \text{PathCost}[s] \end{aligned}$$

- when expanding a state with cost C,
  - all future states will have a cost of at least C
  - all previous states (expanded states) have a cost of at most C

Corollary:

- Dijkstra's algorithm is cost-optimal
  - when it expands a (goal) state, it has found the optimal path to it

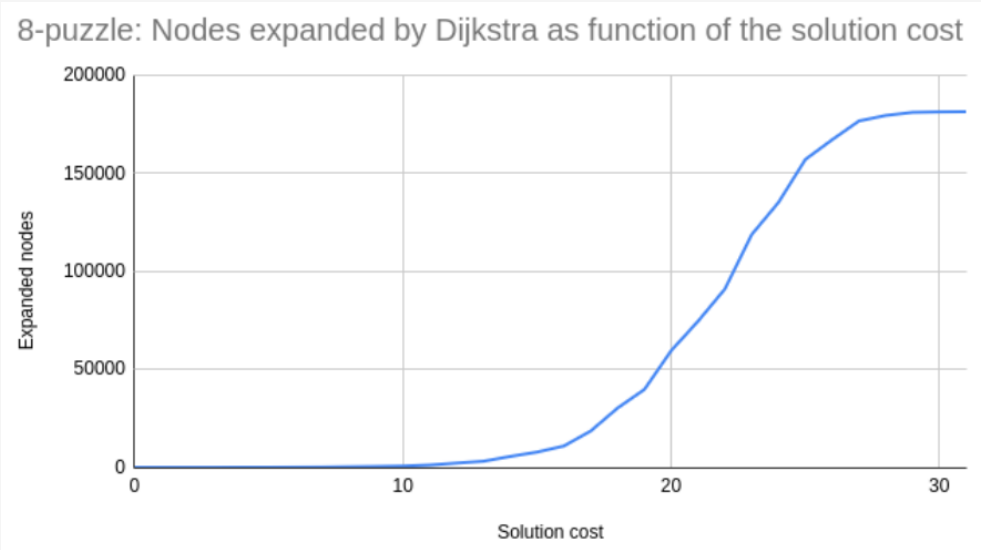
Dijkstra's algorithm: complexity

- $O(N \times log(N) + E)$
  - N: number of expanded nodes
  - E: number of edgees followed

How many nodes do we extract from the frontier?

- $N = |\{s \mid \textsc{PathCost}(s) \leq C^*\}|$
  - C^*: cost of the optimal path to the goal
- $E = O(N \times b)$
  - b: branching factor (number of actions per state)

Dijkstra's algorithm: scaling with solution cost



Informed Best-First Search

- Dijkstra: no information about the goal
- $g(s) = cost(s_{init} \rightarrow s)$: cost of the optimal path from the s_{init} to s
- $h(s) = \hat{cost}(s \rightarrow s_{goal})$: estimated cost of the optimal path from s to s_{goal}
- $f(s) = g(s) + h(s)$
  - priority of s: estimated cost of the optimal path through s

Informed Best-First-Search

Key idea:

- I am in Toulouse and want to go to Bordeaux
- I have established that I need 1h to go to Albi
  - $g(Albi) = cost(Toulouse, Albi) = 1h$
- I know that I need **at least** 2h to go from Albi to Bordeaux
  - $h(Albi) = 2h \leq cost(Albi, Bordeaux)$
- Hence I need at least 3h to go from Toulouse to Bordeaux through Albi
  - $f(Albi) = 3h$
- I should only consider paths that go through Albi once I have no other options for less than 3 hours

A* algorithm

A*: Best-First Search with $f(s) = g(s) + h(s)$

- when $h(s)$ **always underestimates** the true cost to the goal
  - said to be **admissible**

$$h(s) \leq cost(s \rightarrow s_{goal})$$

- A* (with admissible heuristic)
  - it is guaranteed to find the optimal path
  - when it expands the goal state, it has found the optimal path to it

A* algorithm: Optimality

A is optimal when $h(s)$ is admissible*

Proof (sketch):

- A* expands the nodes in increasing order of $f(s)$
- when a state s is expanded, any state s' with f(s') $<$ f(s) has already been expanded
- for the goal state s_{goal}
  - $f(s_{goal}) = g(s_{goal}) = C^*$
- if a state s' is not expanded when reaching the goal
  - $f(s') \geq f(s_{goal}) = C^*$
  - admissibility: $cost(s_{init} \rightarrow s' \rightarrow s_{goal}) \geq f(s')$
  - $cost(s_{init} \rightarrow s' \rightarrow s_{goal}) \geq C^*$
  - a path going through s' is suboptimal

Admissible heuristic: Routing

Path planning:

What is an admissible heuristic for finding the **shortest** path (traveled distance) between two cities?

Admissible heuristic: Routing

Path planning:

What is an admissible heuristic for finding the **shortest** path (traveled distance) between two cities?

- $h_{SLD}(s)$: straight line distance between s and s_{goal}

Admissible heuristic: Routing

Path planning:

What is an admissible heuristic for finding the **shortest** path (traveled distance) between two cities?

- $h_{SLD}(s)$: straight line distance between s and s_{goal}

What is an admissible heuristic for finding the **fastest** path (traveled time) between two cities?

Admissible heuristic: Routing

Path planning:

What is an admissible heuristic for finding the **shortest** path (traveled distance) between two cities?

- $h_{SLD}(s)$: straight line distance between s and s_{goal}

What is an admissible heuristic for finding the **fastest** path (traveled time) between two cities?

- $h_{TT}(s) = \frac{h_{SLD}(s)}{v_{max}}$
  - v_{max}: maximum speed limit on the road network (130km/h)

What makes for a good heuristic

Extreme cases:

- $h(s) = 0$: Uninformed search (Dijkstra's algorithm)
- $h(s) = cost(s, s_{goal})$: Perfect heuristic, but as hard to compute as the solution itself
  - search would only stay on optimal paths

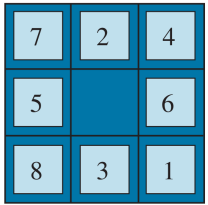
Trade-off:

- something that is reasonably fast to compute (~polynomial time)
- something that is as close as possible to the optimal cost
  - you expand ALL states with $f(s) < C^*$
- but never overestimates the cost to the goal
  - otherwise you lose optimality

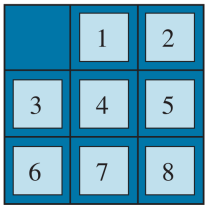
$\Rightarrow$ *Define a relaxed version of the problem by removing some constraint*

Admissible heuristics: 8-puzzle

s_1



s_{goal}

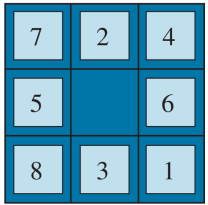


Observation: each action only moves one piece

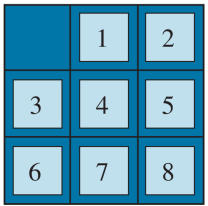
Observation: each misplaced piece must be moved at least once

Admissible heuristics: 8-puzzle

s_1



s_{goal}



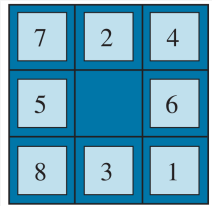
Observation: each action only moves one piece

Observation: each misplaced piece must be moved at least once

- $h_1(s)$: number of misplaced tiles[a] (hamming distance)
  - each misplaced tile requires an action
  - $h_1(s_1) = 8$

Admissible heuristics: 8-puzzle

s_1

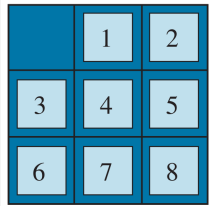


Observation: each action only moves one piece

Observation: each misplaced piece must be moved at least once

- $h_1(s)$: number of misplaced tiles[a] (hamming distance)
  - each misplaced tile requires an action
  - $h_1(s_1) = 8$

s_{goal}

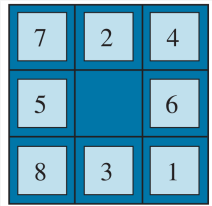


Observation: tile 7 must be moved at least 3 times (2 vertically and 1 horizontally)

- $h_2(s)$: sum of Manhattan distances
  - sum of the distances of each tile to its goal position
  - $h_2(s_1) = 3 + 1 + 2 + 2 + 3 + 3 + 2 = 18$

Admissible heuristics: 8-puzzle

s_1



Observation: each action only moves one piece

Observation: each misplaced piece must be moved at least once

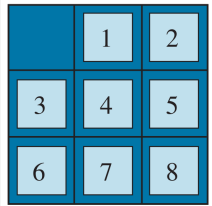
- $h_1(s)$: number of misplaced tiles[a] (hamming distance)
  - each misplaced tile requires an action
  - $h_1(s_1) = 8$

Observation: tile 7 must be moved at least 3 times (2 vertically and 1 horizontally)

- $h_2(s)$: sum of Manhattan distances
  - sum of the distances of each tile to its goal position
  - $h_2(s_1) = 3 + 1 + 2 + 2 + 3 + 3 + 2 = 18$

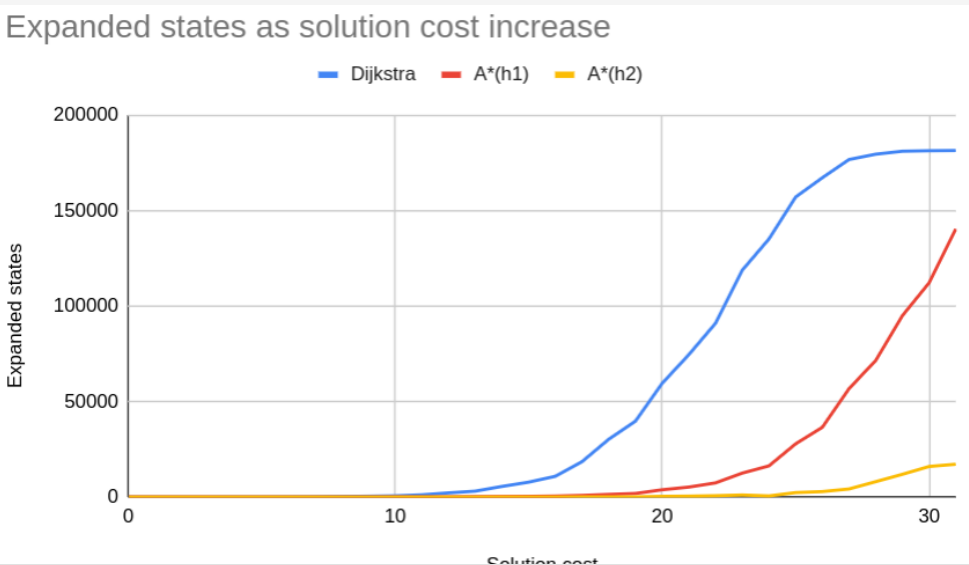
Relaxation: assume pieces can be moved independently

s_{goal}



[a] The "blank" is NOT a tile!

8-puzzle: A* expansions



A* algorithm: complexity and limitations

Even with h_2, A* expands ~10% of the states in the most complex cases

- it may be slow (time complexity: $O(N \times log(N) + E)$)
- it *will* run out of memory on larger problems (space complexity: $O(N)$)

A* algorithm: complexity and limitations

Even with h_2, A* expands ~10% of the states in the most complex cases

- it may be slow (time complexity: $O(N \times log(N) + E)$)
- it *will* run out of memory on larger problems (space complexity: $O(N)$)

For a given heuristic, there is **no algorithm "faster"** than A* while retaining optimality

A* algorithm: complexity and limitations

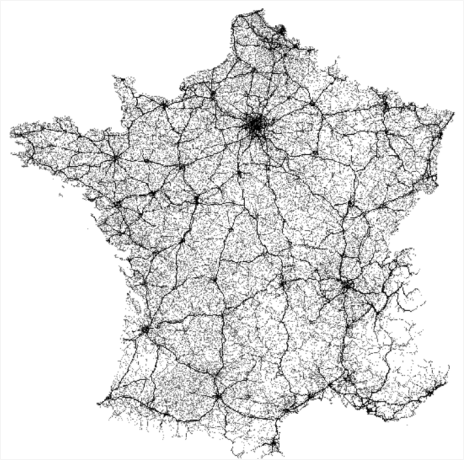
Even with h_2, A* expands ~10% of the states in the most complex cases

- it may be slow (time complexity: $O(N \times log(N) + E)$)
- it *will* run out of memory on larger problems (space complexity: $O(N)$)

For a given heuristic, there is **no algorithm "faster"** than A* while retaining optimality

- but there are algorithms that are more memory efficient
  - e.g. iterative deepening A* (IDA*)
- one could improve heuristic functions
  - more advanced relaxations (e.g. pattern databases for 15-puzzle)
- resort to suboptimal search
  - e.g. Greedy Best-First Search
  - non-admissible heuristics:w

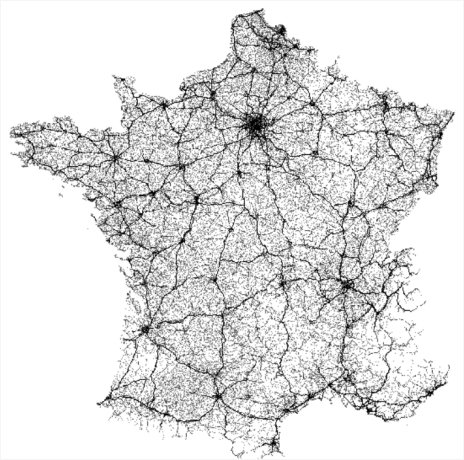
Suboptimal routing heuristics



On a road network we could measure the *detour index*:

$$\text{detour index} = \frac{\text{distance on road}}{\text{distance in straight line}} \approx 1.3$$

Suboptimal routing heuristics



On a road network we could measure the *detour index*:

$$\text{detour index} = \frac{\text{distance on road}}{\text{distance in straight line}} \approx 1.3$$

What about using $h(s) = 1.3 \times h_{SLD}(s)$?

- more accurate on average (closer to true cost)
  - $\Rightarrow$ (much?) faster
- but may be overestimate
  - $\Rightarrow$ suboptimal

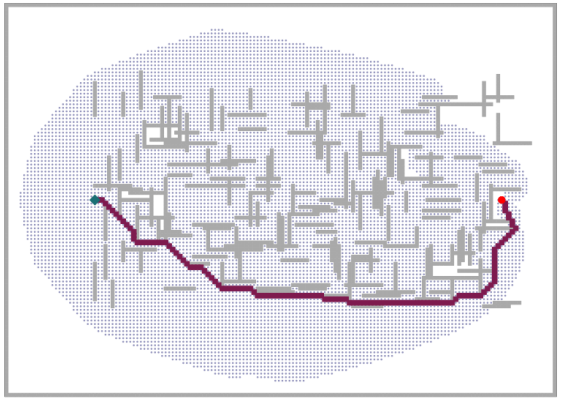
Weighted A*

$$f(s) = g(s) + W \times h(s)$$

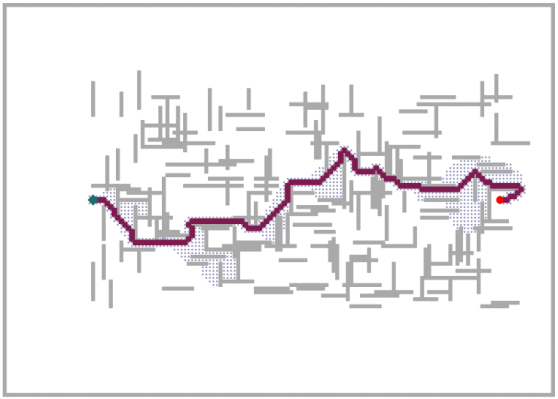
$W > 1$ is a weight factor that makes the goal more attractive (favors states with low $h(s)$)

- suboptimal: solution cost in $[C^*, W \times C^*]$

Grid search: A* vs Weighted A* (W=2)



(a)



(b)

(a) A* search with manhattan distance heuristic (b) Weighted A* with W=2. Weighted A* expands 7 times fewer nodes but finds a suboptimal path, 5% more costly.

(Main) Best-First Search variants

$$f(s) = g(s) + W \times h(s)$$

A*	$f(s) = g(s) + h(s)$	$(W = 1)$
Dikjstra	$f(s) = g(s)$	$(W = 0)$
Weighted A*	$f(s) = g(s) + W \times h(s)$	$(1 < W < \infty)$
Greedy Best-First Search	$f(s) = h(s)$	$(W = \infty)$

Program

Next week: First Lab on 8-puzzle

Search algorithms (Dijkstra, (weighted) A*, Greedy Best-First Search)

Homework:

- solve some 8-puzzles (Online game)
- practice some Rust (Getting Started)

Next lectures:

- making complex decisions under uncertainty
  - sequential, stochastic environments
  - MDP, stochastic games