

# Informed search

Christos Dimitrakakis

March 7, 2024

# Outline

## The Shortest Path Problem

- The shortest path problem

- Heuristic Search

## General weight shortest path

- General weight shortest path

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$
- ▶ We can equivalently consider state-action **costs**  $c(s, a)$ .

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$
- ▶ We can equivalently consider state-action **costs**  $c(s, a)$ .
- ▶ A policy  $\pi$  specifies a path  $x_1, \dots$  with  $x_{k+1} = \tau(x_k, \pi(x_k))$

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$
- ▶ We can equivalently consider state-action **costs**  $c(s, a)$ .
- ▶ A policy  $\pi$  specifies a path  $x_1, \dots$  with  $x_{k+1} = \tau(x_k, \pi(x_k))$
- ▶ Following a **policy**  $\pi$  from state  $x_1 = x$  has a total cost  $C^\pi(x_1) = \sum_{k=1}^t c(x_k, \pi(x_k))$ .

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$
- ▶ We can equivalently consider state-action **costs**  $c(s, a)$ .
- ▶ A policy  $\pi$  specifies a path  $x_1, \dots$  with  $x_{k+1} = \tau(x_k, \pi(x_k))$
- ▶ Following a **policy**  $\pi$  from state  $x_1 = x$  has a total cost  $C^\pi(x_1) = \sum_{k=1}^t c(x_k, \pi(x_k))$ .



# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$
- ▶ We can equivalently consider state-action **costs**  $c(s, a)$ .
- ▶ A policy  $\pi$  specifies a path  $x_1, \dots$  with  $x_{k+1} = \tau(x_k, \pi(x_k))$
- ▶ Following a **policy**  $\pi$  from state  $x_1 = x$  has a total cost  $C^\pi(x_1) = \sum_{k=1}^t c(x_k, \pi(x_k))$ .

## The shortest path problem

- ▶ Input: **start** nodes  $X$  and **goal** nodes  $Y$  and edge costs  $c : A \rightarrow \mathbb{R}$ .
- ▶ Output: Find a path  $h$  from  $X$  to  $Y$  so that  $C(h) \leq C(h')$  for all  $h'$

# The shortest path problem

- ▶ Traversing arc  $\langle x, y \rangle$  incurs **costs**  $c(\langle x, y \rangle)$
- ▶ Following a **path**  $h$  has a total cost  $C(h) = \sum_{\langle x, y \rangle \in h} c(\langle x, y \rangle)$
- ▶ We can equivalently consider state-action **costs**  $c(s, a)$ .
- ▶ A policy  $\pi$  specifies a path  $x_1, \dots$  with  $x_{k+1} = \tau(x_k, \pi(x_k))$
- ▶ Following a **policy**  $\pi$  from state  $x_1 = x$  has a total cost  $C^\pi(x_1) = \sum_{k=1}^t c(x_k, \pi(x_k))$ .

## The shortest path problem

- ▶ Input: **start** nodes  $X$  and **goal** nodes  $Y$  and edge costs  $c : A \rightarrow \mathbb{R}$ .
- ▶ Output: Find a path  $h$  from  $X$  to  $Y$  so that  $C(h) \leq C(h')$  for all  $h'$

## Notes

- ▶ If the path/policy does not reach a goal, the cost is infinite.
- ▶ We can maximise rewards instead of minimising costs.

# Formalising the shortest path problem

The cost from state  $x$  of a policy that reaches a goal is

$$C^\pi(s) \triangleq \sum_{i=1}^{\infty} c[s_t, \pi(s_t)], \quad s_{t+1} = \tau[s_t, \pi(s_t)], \quad s_1 = s$$

where for every  $s \in Y$ ,  $c(s, a) = 0$  and  $\tau(s, a) = s$  for all actions.

- We can calculate this recursively (from the goal state)

$$C^\pi(s) = \sum_{i=1}^{\infty} c[s_t, \pi(s_t)] \tag{1}$$

$$= c[s, \pi(s)] + \sum_{i=2}^{\infty} c[s_t, \pi(s_t)] \tag{2}$$

$$= c[s, \pi(s)] + C^\pi\{\tau[s, \pi(s)]\}. \tag{3}$$

- The same idea applies for the **shortest** path

$$C^*(s) \triangleq \min_{\pi} C^\pi(s) = \min_a \{c[s, a] + C^*[\tau(s, a)]\}. \tag{4}$$

# Dijkstra's shortest path algorithm: backward search

## Shortest path algorithm

Input: Goal states  $Y$ , starting state  $x$ .

Set  $C(s) = 0$  for all states  $s \in Y$ ,  $F_0 = Y$ .

**for**  $t = 0, 1, \dots$  **do**

**for**  $s' \in F_t$  **do**

$\pi(s) = \arg \min_a c(s, a) + C(\tau(s, a))$

$C(s) = \min_a c(s, a) + C(\tau(s, a))$

**end for**

$F_{t+1} = \text{parent}(F_t)$ .

**if**  $F_{t+1} = \emptyset$  or  $x \in F_t$  **then**

**return**  $\pi, C$

**end if**

**end for**

## Algorithm idea

- ▶ Start from goal states
- ▶ Go back one step each time, adding the cost.
- ▶ Stop whenever there are no more states to go back to, or if we reach the start state.

# Optimality proof

## Theorem

$$C(s) = C^*(s)$$

## Proof

- ▶ If  $s \in Y$ , then  $C(s) = 0 = C^*(s)$ .
- ▶ For any other  $s'$ ,  $s = \text{parent}(s')$ : we will show that: if  $C(s') \leq C^*(s')$  then  $C(s) \leq C^*(s)$ .

$$\begin{aligned} C(s) &= \min_a \{c(s, a) + C(\tau(s, a))\} && \text{(by definition)} \\ &\leq \min_a \{c(s, a) + C^*(\tau(s, a))\} && \text{(by induction)} \\ &\leq \min_a \left\{ c(s, a) + C^{\pi'}(\tau(s, a)) \right\}, \quad \forall \pi' && \text{(by optimality)} \\ &\leq C^{\pi}(s), \quad \forall \pi. \end{aligned}$$

For the optimal policy  $\pi^*$ ,  $C^{\pi^*}(s) = C^*(s)$ , so  $C(s) \leq C^*(s)$ . Finally,

$$C^*(s) \geq C^{\pi}(s) = C(s) \geq C^*(s),$$

since  $C^{\pi}(s) = C(s)$  for the policy returned by the algorithm.

# Partial graphs

- ▶ Why do we need search?
- ▶ We do not want to calculate on the whole graph
- ▶ We use **search** to find the shortest path more efficiently (perhaps).
- ▶ We denote the total cost of some path  $x_1, \dots, x_t$  as:

$$C(x_1, \dots, x_t)$$

- ▶ The remaining cost from  $x_t$  to the goal using some policy  $\pi$  as
- $$C^\pi(x_t)$$

## Generic search

We define heuristic search in the context of shortest-path problems.

We now consider a general method for searching a node in the frontier.

**input**  $G = \langle N, E \rangle$ : Graph.

**input**  $f : N \rightarrow \mathbb{R}$ : evaluation function.

**input**  $x$ : Start node

**function** Heuristic Search( $G, x, h$ )

$S' = \emptyset$ : Nodes searched.

$F = \{x\}$ . Initialise the frontier

$c_x = 0$ . Initialise the cost of node  $x$

**while**  $F \neq \emptyset$  **do**

$n = \arg \min_{i \in F} f(i)$ . Select "best" node.

$F = F \setminus \{n\}$ . Remove  $n$  from the frontier.

**if**  $n \notin S'$  **then**

$B = \text{child}(n) \setminus S'$ . Get the set of unsearched children of  $n$ .

$\forall b \in B, b_i = c_n + c(n, b)$ . Calculate the total cost to each child  $b$ .

$S' = S' \cup \{n\}$ . Add  $n$  to the list of searched nodes.

$F = F \cup B$ . Add  $n$ 's children to the frontier.

**end if**

**end while**

## A\* search

We now consider a general method for searching a node in the frontier.

**input**  $G = \langle N, E \rangle$ : Graph.

**input**  $h : N \rightarrow \mathbb{R}$ : heuristic function.

**input**  $x$  : Start node

**function** A-Star( $G, x, h$ )

$S' = \emptyset$  : Nodes searched.

$F = \{x\}$ . Initialise the frontier

$c_x = 0$ . Initialise the cost of node  $x$

**while**  $F \neq \emptyset$  **do**

$n = \arg \min_{i \in F} c_i + h(i)$ . Select minimum cost + heuristic node.

$F = F \setminus \{n\}$ . Remove  $n$  from the frontier.

**if**  $n \notin S'$  **then**

$B = \text{child}(n) \setminus S'$ . Get the set of unsearched children of  $n$ .

$\forall b \in B, b_i = c_n + c(n, b)$ . Calculate the total cost to each child  $b$ .

$S' = S' \cup \{n\}$ . Add  $n$  to the list of searched nodes.

$F = F \cup B$ . Add  $n$ 's children to the frontier.

**end if**

**end while**

► You can see that  $h = 0$  corresponds to minimum-cost search.



# Admissible heuristics

- ▶ If  $h$  is arbitrary, then the search can fail.
- ▶ We need  $h$  to be admissible. In particular,

$$C^*(n) \geq h(n).$$

# Admissibility of $A^*$

## Theorem

$A^*$  returns an optimal solution if

- ▶ The graph has a bounded branching factor.
- ▶ All costs are greater than  $\epsilon > 0$
- ▶ The heuristic is admissible, i.e.  $0 \leq h(n) \leq C^*(n)$  for all  $n \in N$ .

## Proof

- ▶ **Existence.** There is a finite number of paths that will be explored, as the longest possible path to a goal is  $C^*(0)/\epsilon$ .
- ▶ **Optimality.** The proof is by contradiction. Let us assume that  $A^*$  finds some  $\pi \neq \pi^*$  so that  $C(\pi) > C(\pi^*)$ . That means that at some node  $n$  on the path there is an action  $a^*$  on the optimal policy, but we keep expanding the path  $x_1, x_2, \dots$  of  $\pi$ . However, since  $C(\pi) > C(\pi^*)$  there must be some  $t$  such that  $C(n, x_1, \dots, x_t) > C^{\pi^*}(n)$ . But then, to expand  $\pi$  requires that  $C(n, x_1, \dots, x_t) + h(x') < h(x) \leq C^{\pi^*}(n)$ .

# General weight shortest path

- ▶ In this problem, actions can have positive or negative costs.
- ▶ Negative edges generate problems if we have cycles

# Bellman-Ford Algorithm

```
 $C(0) = 0. C(i) = \infty, \text{ for } i \neq 0.$   
for  $i \in 1, \dots, |N| - 1$  do  
  for all edges  $(i, j)$  do  
    if  $C(i) + c(i, j) < C(j)$  then  
       $c(j) = C(i) + c(i, j)$   
    end if  
  end for  
end for  
for all edges  $(i, j)$  do  
  if  $C(i) + c(i, j) < C(j)$  then  
    error "Negative cycle"  
  end if  
end for
```

- ▶ Succinctly, the algorithm is just like Dijkstra, but it ensures it goes at most  $|N| - 1$  times through all vertices, and has a sanity check as no more updates should be possible at the end.
- ▶ Instead of keeping a track of explored nodes, it uses the fact that  $C$  is initialised to infinity.