Informed search

Christos Dimitrakakis

March 8, 2024

Outline

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 π specifies a path $x_1, \ldots$ 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 π specifies a path $x_1, \ldots$ with $x_{k+1} = \tau(x_k, \pi(x_k))$
- Following a policy π 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 π specifies a path $x_1, \ldots$ with $x_{k+1} = \tau(x_k, \pi(x_k))$
- Following a policy π 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 π specifies a path $x_1, \ldots$ with $x_{k+1} = \tau(x_k, \pi(x_k))$
- Following a policy π 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 $c : A \to \mathbb{R}$.
- 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 π specifies a path $x_1, \ldots$ with $x_{k+1} = \tau(x_k, \pi(x_k))$
- Following a policy π 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 $c : A \to \mathbb{R}$.
- 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

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

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)

$$\begin{aligned} C^\pi(s) &= \sum_{i=1}^{\infty} c[s_t, \pi(s_t)] && (1) \\ &= c[s, \pi(s)] + \sum_{i=2}^{\infty} c[s_t, \pi(s_t)] && (2) \\ &= c[s, \pi(s)] + C^\pi\{\tau[s, \pi(s)]\}. && (3) \end{aligned}$$

- The same idea applies for the shortest path

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

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 $t = 0, 1, \ldots$ **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
  $F_{t+1} = \texttt{parent}(F_t)$.
  if $F_{t+1} = \emptyset$ or $x \in F_t$ **then**
    return π, 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 $s', s = \texttt{parent}(s')$: we will show that: if $C(s') \le C^*(s')$ then $C(s) \le C^*(s)$.

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

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

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

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 $x_1, \ldots, x_t$ as:
$$C(x_1, \ldots, x_t)$$
- The remaining cost from x_t to the goal using some policy π 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 $f : N \to \mathbb{R}$: 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**
    $B = \texttt{child}(n) \setminus S'$. 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 $h : N \to \mathbb{R}$: 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**
        $B = \texttt{child}(n) \setminus S'$. 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 that $\epsilon > 0$
- 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 as 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 $x_1, x_2, \ldots$ of π. However, since $C(\pi) > C(\pi^*)$ there must be some t such that $C(n, x_1, \ldots, x_t) > C^{\pi^*}(n)$. But then, to expand π requires that $C(n, x_1, \ldots, x_t) + h(x') < h(x) \leq C^{\pi^*}(n)$.

Calculating Upper and Lower Bounds

Starting from a set of leaf nodes S_0

Upper bound $U(s) \geq C^*(s)$ for $s \in S_0$

Setting $U(0) \geq C^*(0)$ and recursing:

$$U(s) = \min_{a \in A_s} c(s,a) + U[\tau(s,a)]$$

By induction, we can prove that this is an upper bound on C^*:

$$U(s) = \min_{a \in A_s} c(s,a) + U[\tau(s,a)] \geq \min_{a \in A_s} c(s,a) + C^*[\tau(s,a)] = C^*(s).$$

Lower bound $L(s) \leq C^*(s)$ for $s \in S_0$

$$L(s) = \min_{a \in A_s} c(s,a) + L[\tau(s,a)]$$

Similarly, we can prove that it is a lower bound:

$$L(s) = \min_{a \in A_s} c(s,a) + L[\tau(s,a)] \leq \min_{a \in A_s} c(s,a) + C^*[\tau(s,a)] = C^*(s)$$

Branch and bound

The algorithm is rather simple to describe in words.

- [1] Set $s = 0$.
- [1.1] Select action a^* minimising $c(s, a) + L(\tau(s, a))$.
- [1.2] Discard subtrees (s, a) for which $c(s, a) + L(\tau(s, a)) \geq c(s, a^*) + L(\tau(s, a^*))$.
- [1.3] Proceed to $s = \tau(s, a)$ and go to 1.1. unless we are at a leaf.
- [2] Expand the leaf node, and generate new leaf nodes with corresponding upper and lower bounds.
- [3] Calculate L, S for the corresponding subtree.
- [4] Go to 1.

General weight shortest path

- In this problem, actions can have positive or negative costs.
- Negative edges generate problems if we have cycles
- However, the basic algorithmic idea is again Dynamic Programming

Bellman-Ford Algorithm

In state-action notation, the algorithm is simply

- $C_0(0) = 0$, $C_i(0) = \infty$ for all $i \neq 0$.
- For $k \in 1, \dots, |S|$:
$$C_k(s) = \min_a c(s,a) + C_{k-1}(\tau(s,a))$$

Bellman-Ford Algorithm

$C(0) = 0$. $C(i) = \infty$, for $i \neq 0$.
for $i \in 1, \ldots, |N| - 1$ **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.