CS3243

Jia Cheng

Jan 2023

Formalizations.

Modelling Notation

- State space S; state $s \in S$ (n can also be used to denote state)
- Paths from s to t: $Paths(s,t)$; Path $p = (n_0 = a, a_1, n_1, a_2, n_2, \ldots, n_k, a_k = b)$ (A sequence of states interleaved with actions)
- Actions available at state n: $Actions(n)$
- Transition function: $T : (n_1, a) \mapsto n_2$
- Transition cost: $cost(n_1, a, n_2)$
- Path cost: For $p \in Paths(a,b)$, $cost(p) = cost(a,p,b) = \sum_{i=1}^{k} cost(n_{i-1}, a_i, n_i)$
- Goals set: $Goals$; goal test function: $\forall n \in S, isGoal(n) \iff n \in Goals$

Cost functions and estimates

- $f : \{(n,p) : n \in S \wedge p \in Paths(s,n)\} \to \mathbb{R}_0^+$
- Cost of path taken $g : \{(n,p) : n \in S \wedge p \in Paths(s,n)\} \to \mathbb{R}_0^+$
- Heuristic at state $h : S \to \mathbb{R}_0^+$
- $f = g + h$, to be more rigorous, we can overload h by stating that $\forall p \in Paths(s,n), h(n,p) := h(n)$

In lecture, often it is written $g(n)$, note that the p is implicit.

Heuristic properties

Optimal heuristic An optimal heuristic h^* satisfies $\forall n$,

$$h^*(n) = \min \{cost(n,p,G) : G \in Goals, p \in Paths(n,G)\} \tag{1}$$

Admissible heuristic A heuristic h is admissible if $\forall n \in S$,

$$h(n) \leq h^*(n) \tag{2}$$

Equivalently, $f(n) \leq f(G)$, where $cost(n,p,G) = h^*(n)$.

Consistent heuristic A heuristic h is consistent if $\forall n, n' \in S, \forall a \in Actions(n), T(n,a) = n' \implies$

$$h(n) \leq cost(n,a,n') + h(n') \tag{3}$$

Each of the following statements are equivalent to consistency

- $\forall a, b \in S, \forall p \in Paths(a,b), h(a) \leq cost(p) + h(b)$
- $\forall a, b \in S, \forall p \in Paths(a,b), f(a,p) \leq f(b,p)$

Lemma Suppose a heuristic h is consistent. Then for all $n \in S$, let us choose $G \in Goals, p \in Paths(n, G)$ that satisfies $cost(n, p, G) = h^*(n)$. Then, applying the definition of consistency we get $h(n) \leq cost(n, p, G) + h(G) = h^*(n) + 0 = h^*(n)$. This proves admissibility. $\square$

Theorems on Search Algorithms

Graph Search algorithm v2 The first property of this algorithm is that if we ignore the termination condition, then this algorithm will explore all non-redundant paths. To prove this, we consider the characterization of a path that was not fully explored. Then, there exists a node n_2 on this path, closest to the source, that was not explored. Let n_1 be its predecessor, and consider what happened after n_1 was popped off the priority queue. There are two cases.

1. The first case, is where n_2 was not added to the priority queue. This can only occur if n_2 was visited via another path, and with a lower cost, this says that p is indeed redundant.
2. The second case is where n_2 was added to the priority queue. If we ignore the termination condition of the loop, and assume that all edges have a cost $\geq \epsilon$, we can see that n_2 will eventually be popped of the PQ as well. The reason is that for each $k \in \mathbb{Z}^+$ there are finitely many nodes at most k edges away from the source, which implies there are finitely many nodes of distance at most $k\epsilon$ from the source. This implies that finitely many nodes of distance $k\epsilon$ can be added to the PQ, and by the Archimedean property of real numbers, eventually n_2 will have the lowest cost in the PQ and be popped off.

Hence, if we ignore termination, we know that any path left out by Algorithm 2 is necessarily redundant.

Another way to interpret the above property is that this algorithm would not ignore any path that could be an improvement over the previous, even if the node has been visited.

Corollary. Algorithm 2 is complete. E.g. Suppose a solution exists, and we assume Algorithm 2 doesn't terminate, then eventually the optimal path towards the goal will be explored, which is a contradiction.

Optimality of Algorithm 2 If the heuristic is admissible, then Algorithm 2 is optimal.

Proof. We assume the admissibility of the heuristic h, so that f satisfies $f(n) \leq f(G, p)$ where p is an optimal path taken from n to the closest goal (to n) G. To obtain a contradiction, let us suppose Algorithm 2 finds us a suboptimal tuple (G', p') where $G' \in Goals, p' \in Paths(s, G')$. Note that we may very well have $G' = G$, but p' is suboptimal compared to any optimal path p. In any case, $f(G, p) < f(G', p')$.

Consider the last node n_2 alongst p that was not explored, with n_1 being its predecessor on p. If n_2 was placed on the PQ, then we immediately have a contradiction, as admissibility implies $f(n_2) \leq f(G, p) < f(G', p')$. (We don't denote the subpath of p from s to n_2 for brevity.) Hence, it must be the case that n_2 was not placed on the PQ, but why would this happen? By our previous discussion, this means this subpath of p is redundant, and another optimal subpath from s to n_2 must have been previously explored. This is perfectly fine, as we can cut out the subpath of p from s to n_2 and replace it with this other optimal subpath.

Regardless, this means that we must have explored n_2. If we keep repeating this argument, we will eventually arrive at the conclusion that G itself must have been explored via some optimal path, which is a contradiction. $\square$

Note: We can also rephrase our proof to say something like 'Consider the last node n_2 alongst p that was not explored by **any** optimal path.'

Note: What make Algorithm 2 easy to reason about is that it does not have strict requirements on whether a node has been visited. As long as a path is an improvement, it will be considered on the PQ.

Consistent heuristics and PQs If a consistent heuristic h is used, then we can say the following about the behavior of the PQ. At any time point, let the value at the head of the PQ be u. Then u is $\leq$ any element that *is currently* or *will ever be in the future* in the PQ.

Proof. Let v be any value that is added later to the PQ. Then there exists a sequence $v_0, v_1, \ldots, v_k = v$ such that v_i was added to the PQ as a result of v_{i-1} having been popped off, and v_0 was on the PQ when

u was at the head. (v_0 may very well be u itself). Let $v_{i-1} = f(n_{i-1}, p)$ where p is some path taken from s to n_{i-1}. Let $v_i = f(n_i, p \cdot a \cdot n_i)$ (here $\cdot$ denotes concatenation). Then by consistency, $f(n_{i-1}) \leq f(n_i)$, which implies $v_{i-1} \leq v_i$. $\square$

Corollary. Any earlier value popped from a PQ cannot be strictly larger than a later value popped.

Optimality of Algorithm v3 If the heuristic h is consistent, then Algorithm 3 is optimal. Our proof is shares similarities to the proof of Algorithm v2.

Define n_1, n_2, p, p', G like in the proof of Algorithm v2. If n_2 was added to the PQ, then since h is admissible, it will not be possible to see $f(G', p')$ before popping off $f(n_2)$. If n_2 was not added to the PQ, then n_2 must have been visited via some other path. Using our Corollary to the consistency property, since n_2 was popped off the PQ before n_1 (otherwise n_2 would have been added to the PQ), then $cost(s, \text{ some other path}, n_2) \leq cost(s, \text{ subpath of } p, n_1) \leq cost(s, \text{ subpath of } p, n_2)$. Which says that this other path is not worse than the subpath of p. And we can do a similar cut and paste as in our previous proof.

Eventually, we reach the conclusion that (G, p) must have been explored before (G', p'), a contradiction. $\square$

Remark. While the proofs are similar, also note their differences. Because Algorithm 3 is more strict on the 'visited' state of a node, we cannot just say that an improvement will lead to (n_2, subpath of p) being added to the PQ.