02-613 Week 4

Algorithms and Advanced Data Structures

Aidan Jan

September 17, 2025

A-Star Algorithm

Suppose we have a GPS, and we want to get from Pittsburgh to Philly. Currently, we have discussed how to do that with Dijkstra's algorithm, however:

- This requires us to explore paths to nodes on the whole map!
- Slow, and most calculations are unnecessary.
- Is there a better way?

In our GPS case, we would want to tell the algorithm, somehow, you should search paths going east.

- At the same time though, depending on where you are, not every path should go east. Maybe a highway starts slightly to the west, and it will save time.
- How do we decide how far west is allowed?

What if we use the straight line distance as an estimate to which node (city) to visit next?

- In addition to the `d[]` $= \infty$ for the start of Dijkstra's, we also initialize every city with a `h[]` $=$ (straight line distance to destination).
- The `h[]` is called a *heuristic*.

```
for u in V, d[u] = INTEGER_MAX, p[u] = 0, F = V
d[s] = 0
while F != Q:
    f[u] = d[u] + h[u]
    u = (vertex in F with min f[u])
    remove u from F
    for v in Neighbor(u):
        if d[u] + h[v] < d[v]:
            p[v] = u
            d[v] = d[u] + h[v]
```

- Note that F is a heap, which allows for quick removal of the minimum item.

Heuristics

What do we choose the `h[]` function to be? We chose distance in this case, but what else may be good?

- **Theorem:** A* is guaranteed to find the optimal route if $h(u)$ is *admissable*.
- In this case, the best heuristic we can come up with is the time to travel from one city to another, since that instantly finds a solution.

- The worst heuristic we can come up with is $h(u) = 0$, since that would cause us to explore every node just like Dijkstra's.
- In other words, the heuristic is always an *underestimate* of the correct solution.
  - If it wasn't an underestimate, it may cause us to miss the optimal path!
- We want to show that with a good enough heuristic, A* would find the optimal route, even without exploring every node.

Proof

Let P* be the optimal path from point A to point B. This path includes nodes $w_0, \ldots, w_m$, with w_0 also being the start, and w_m is the last node before the end, t.

- **Lemma:** When each node w_0 to w_{m-1} was last removed from the heap, $d(w_1) = d^*(w_1)$
- This is provable by induction.
  - Base: $d[s] - d[w_0] = 0$
  - Hypothesis: We assume that every next node is also on the optimal path. Therefore,

$$\begin{aligned} d(w_{k+1}) &= d(w_k) + len(w_k, w_{k+1}) \\ &= d^*(w_k) + len \\ &= d^*(w_{k+1}) \end{aligned}$$

Suppose for the purpose of contradiction, the A* algorithm found a non-optimal solution. Let w_m be a node that we decided to visit over t, the ending node.

$$\begin{aligned} f[t] &= d[t] + h[t] \\ &= d[t] \\ &\leq f[w_m] \\ &= d[w_m] + h[w_m] \\ &= d^*[w_m] + h[w_m] \\ &\leq d^*[w_m] + h^*[w_m] \\ &= length(p^*) \end{aligned}$$

p^* is the known optimal solution, and h^* is the optimal heuristic.

- Remember that $h[t]$ is just 0, since it is the ending node.
- Since we visited w_m instead of t, that means $f[w_m]$ must have been lower, hence step 3.
- $d^*[w_m]$ gives the same value as $d[w_m]$, since we assume the path we took to get there was optimal. This is based on the lemma.
- We know that $h[w_m] \leq h^*[w_m]$, since the heuristic is an underestimate. $h[w_m]$ was an estimate from w_m to t, $h^*[w_m]$ was the actual distance.
- Since the sum of the optimal distance plus the optimal heuristic is just the length of the optimal path, this path through w_m must be optimal. However, this is a contradiction since this path was defined as non-optimal.

Bellman-Ford Algortihm

What if our graph contains negatively weighted edges? Now Dijkstra's doesn't work since for Dijkstra's, once a node is visited, it is not visited again.

However, with the introduction of negatively weighted edges, we have a problem to address first: **Negative Weight Cycles**

- What if a cycle on a graph has a negative weight overall? In this case, the solution would be infinitely long since it is always "worth it" to go around the cycle one more time.
- To fix this, we need to first detect the negative weight cycles before attempting to solve for the lowest weight path between two points.

Now, we can start. The problem statement: Given a directed graph with edge weights $(u, v) \in \mathbb{R}$:

1. Determine if it contains a negative cycle
2. If so, return infinity. If not, find the shortest $s \to t$ path.

The algorithm goes as follows:

- Let d_s[u] be the current estimated $s \to ud$
- d[s] $=$ 0, d[u] $= \infty \forall u \neq s$
- **Ford Step:** Find edge (u, v) such that d_s[v] $>$ d_s[u] $+$ d(u, v). Set d_s[v] $=$ d_s[u] $+$ d(u, v).

Theorem

If you cannot relax (via Ford), then d_s[u] is the shortest $s \to u$ path $\forall u$

Proof

We first need some lemmas:

Lemma 1: After Step i, either for any v $\in$ V, d_s[v] $= \infty$, or there exists an $s \to v$ path of length d_s[v].

To prove this lemma, we can use induction. First, if d_s[v] $= \infty$, we can't say anything. For the other case:

- When $i = 0$ (base case), d_s[s] $=$ 0 (which is less tha infinity), there exists an $s \to s$ path of weight 0.
- Let's assume (inductive step) the lemma is true for $i - 1$. In the i-th step, we update edge (u, v) where d_s[v] $=$ d_s[u] $+$ d(u, v).
- From here, we know that since d_s[u] was updated at step $i-1$ or earlier, there exists a path from $s \to u$ of length d_s[u].
- Therefore, there exists a path $s \to u \to v$ of distance d_s[u] $+$ d(u, v).

Lemma 2: When no more Ford steps are possible, for all paths P_{sv}, $s \to v$, length(P_{sv}) $\geq d_s$[v]

Thie lemma can also be proven by induction:

- When $|P_{sv}| = 1$ (base case), then the path is a single edge. Therefore, it cannot be relaxed. d_s[v] $\leq$ d(s, v).
- Assume the lemma is true for all $|P_{sv}| \leq (k - 1)$. (Inductive case).
- Let P_{sv} be an $s \to v$ path of length k edges. $P_{sv} = P_{su} + (u, v)$ (Note that P_{su} has length 1.)
- cost(P_{sv}) $=$ cost(P_{su}) $+$ d(u, v)
- Inductive Hypothesis $\geq d_s$[u] $+$ d(u, v) $\geq d_s$[v].

Implementation

```
Initialize d[u] = infinity forall u, d[s] = 0
for i in [n]:
    for (u, v) in E:
        if d[v] > d[u] + d(u, v):
            d[v] = d[u] + d(u, v)
            parent[v] = u
```

Shortest Paths Summary

- BFS, O(n + m), unweighted graphs
- Dijkstra, O(m log(n)), positive edge weights, single source all paths
- A-Star, O(m log(n)), needs heuristic
- Bellman-Ford, O(mn), arbitrary large weights (incl. negatives)