24.3 A* Algorithm

Professor Hug's Lecture on Shortest Paths

A*

We ended the section on Dijkstra's by discussing a possible way to make Dijkstra's short-circuit and just stop once it hits a given target. Is this good enough?

To answer the above question, we need to sit down and think about how dijkstra's really works. Pictorially, Dijkstra's starts at the source node (imagine the source node being the center of a circle.) And Dijkstra's algorithm now makes concentric circles around this point, in increasing radii, and 'sweeps' these circles, capturing points.

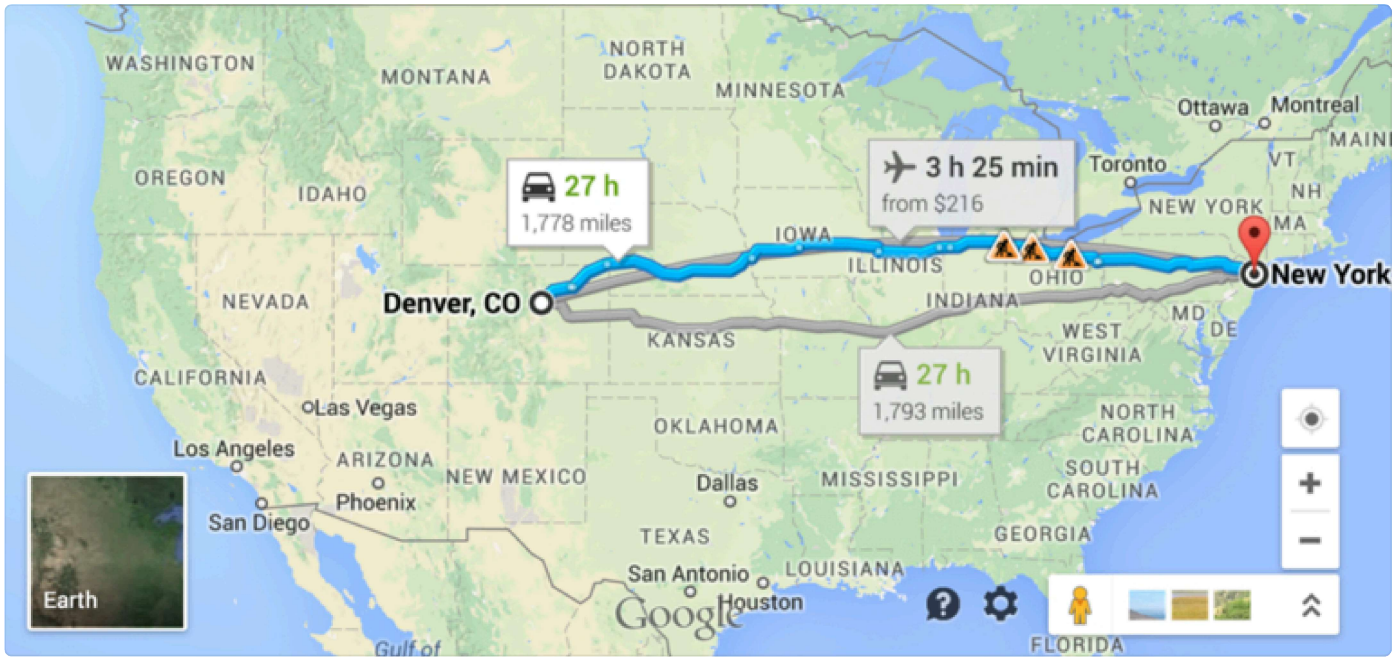
So... the first node Dijkstra's visits is the city closest to the source, then the city next-closest, then the city next-closest, and so on. This sounds like a good idea. What Dijkstra's

is doing is first visiting all the cities that are 1-unit distance away, then 2 unit-distance away, and so on. In concentric circles.

Now imagine the following: on a map of the US, start somewhere in the center, say, Denver. Now I want you to find me a path to New York using Dijkstra's. You'll end up traversing nodes in 'closest concentric circle' order.

You'll make a small circle first, just around Denver, visiting all the cities in that circle. Eventually, your circles will get bigger, and you'll make a circle that passes through Las Vegas (and would have visited, by now, all the other cities that fall within the circle.) Then, your circle will be big enough to engulf Los Angeles and Dallas... but you're nowhere close to New York yet. All this effort, all these circles, but still... so far from the target. Short-circuiting helps, but only if you actually hit the target node fast.

If only there existed a way to use your prior knowledge: the fact that new-york was eastwards, so you could "hint" your algorithm to prefer nodes that are on the east instead of those that are on the west.



A*

Introducing: A Star

No, not the sun. It's an algorithm called A*.

Observe the following: Dijkstra's is a "true" (i.e., not an estimate) measure of the distance **to** a node from the source. So, say, you visit a city in Illinois and your source was Denver, then by that time, you have a true measure of the distance **to** Denver. What we're missing is: some janky, rough estimate of the distance from a node **to** the target node, New York. That would complete the picture. Because then, if you sum these two things up (the measure from the source to the node + the estimate from the node to the target), you get (an estimate from the source to the target.) Of course, the better your original estimate from the node to the target, the better your estimate from the source to the target, the better your A* algorithm runs.

So, let's modify our Dijkstra's algorithm slightly. In Dijkstra's, we used **bestKnownDistToV** as the priority in our algorithm. This time, we'll use **bestKnownDistToV+estimateFromVToGoal** as our heuristic.

Here is a demo!

Chicken And Egg

We have a problem. How do we know what the estimate is? I mean, the estimate itself is a **distance**, and we're using A* to **find** the distance from some node to some other node.

It seems like we're in an instance of the classic chicken and egg problem. "What came first? The chicken or the egg?" Aside, FYI, one reddit user had an idea about this.

Well, it's called an estimate because it's exactly that. We use A* to get the **true** shortest path from a source to a target, but the estimate is something we approximate. Coming up with good estimates is hard sometimes.

But to give you an example in our Denver - New York case. What we might do is just look up the GPS Coordinates of these cities, and calculate the straight line distance between those somehow. Of course, this wouldn't be correct because there's probably no straight line that one could take from Denver to NYC, but it's a fairly good estimate!

Bad Heuristics

Suppose that the shortest path from Denver to New York goes through some city C. Suppose that my GPS is broken, and so I think that this city C is infinity far away from

everything, and I set the estimated distance to C from every other node in the graph to ∞.

What will happen? Well A* will basically never want to visit this city. (Remember what our priorities are in the priority queue; for this city, the priority will always be ∞∞, even if I visit the immediate neighbors of this city. The estimated distances from the immediate neighbors of this city to this city were set to ∞∞ after all.)

So... now what? We lose. A* breaks. We get the wrong answer back. Oops.

The takeaway here is that heuristics need to be good. There are two definitions required for goodness.

1. Admissibility. heuristic(v, target) ≤≤ trueDistance(v, target). (Think about the problem above. The true distance from the neighbor of C to C wasn't infinity, it was much, much smaller. But our heuristic said it was ∞∞, so we broke this rule.)
2. Consistency. For each neighbor v of w:
   - heuristic(v, target) ≤≤ dist(v, w) + heuristic(w, target)
   - where dist(v, w) is the weight of the edge from v to w.

Previous
24.2 Dijkstra's Algorithm

Next
24.4 Summary

Last updated 1 year ago