**Lesson Overview**

This lesson is taught by [**Peter Norvig**](https://en.wikipedia.org/wiki/Peter_Norvig).

Peter Norvig is a true expert in robotics, artificial intelligence, and computer science. Back before Udacity even existed Peter and Sebastian co-taught a course on self-driving cars. The popularity this course generated when they put it online was a big part of the decision to start this company. Peter is also a writer and his essay [**Teach Yourself Programming in 10 Years**](http://norvig.com/21-days.html) is something I would *highly* recommend to new programmers.

In this lesson you will learn about "search problems" and several algorithms to solve them.

One particular search problem is especially relevant to self driving cars: finding the best route from point A to point B. When this lesson is over you will take everything you've learned in this course and actually implement a Google-maps style route planner.

Here's the graph from the video.

A close up of a map

Description automatically generated

In the video from 1:55 onward, we are looking ahead to a modified algorithm that keeps track of explored states so that they aren't repeated. In the preliminary algorithm, A is repeated since we are not keeping track of explored states. Ideally, we would not add duplicates from backtracking, which we introduce in the following video.

A close up of a clock

Description automatically generated

Here's the graph from the video.

A close up of a map

Description automatically generated

Here's the graph from the video. Note the arrow pointing at Fagaras - this is the city Dr. Norvig is pointing to.

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

Now that we've added quite a few paths to our map, it can be a bit difficult to follow which paths are being checked at each step.

Starting at 1:30, we are checking the path coming into Craiova from Rimnicu Vilcea, and heading toward either Drobeta (366+120 = 486) or Pitesti (366+318 = 504), both of which are worse than our current best of 418.

After that, we already know the path to get to Drobeta was worse than the path to get to Craiova, so there cannot be a more beneficial path heading to that already explored location.

A screenshot of a cell phone

Description automatically generated

**Uniform Cost search** - expands out equally in all directions, may expend additional effort getting to a fairly direct path to the goal.

**Greedy best-first search** - expands outward toward locations estimated as closer to the goal. If a direct path is available, expends much less effort than Uniform Cost; however, it does not consider any routes in which it may need to temporarily take a further away path in order to arrive at an overall shorter path.

**A\* Search** - utilizes both of these - will try to optimize with both the shortest path and the goal in mind. We'll see how this works in the next video.

A picture containing person, water

Description automatically generated

Rimnicu Vilcea and Fagaras hardly increased in total value under \large f*f* even though they added 80 and 99, respectively, in path cost, as their estimated distance to the goal decreased. Since they are much more in the specific direction of our goal of Bucharest, the search space is much more inclined to expand to them under A\*, similar to greedy best-first search.

Watch the videos to understand the texts more.

It's key to note here than any remaining frontiers outside of our goal path, i.e. any additional unexplored paths, must now have true costs greater than our selected shortest path, p.

h2 is always greater than h1 as each misplaced block must always need to move at least a distance of 1, but that distance could be greater than 1.

Why does h2 expand fewer nodes than h1?

As Peter says, h2 is always greater than or equal to h1. To see why this should expand fewer paths, let's imagine a heuristic h3 that always had the exact cost at every node. This heuristic would naturally expand the least number of nodes, as it would know the actual best path to take at each step.

On the other hand, imagine a heuristic h4 that was always 0. This heuristic would expand the most number of nodes of all possible heuristics, as it would have essentially no knowledge of estimated "distance" to the goal, and therefore be more similar to Uniform Cost search.

You can see then that a heuristic that is strictly greater than or equal to another one (assuming it is still admissible!) gets us closer to the perfect heuristic and thereby expands at least the same number of nodes or fewer.

You can consider the path Dr. Norvig is using here as one from Arad (A) to Sibiu (S) to Fagaras (F), it it helps. The cost path is not the exact one used in the previous examples.