While most people can look at a map of cities and find a good route from one town to the next, a computer must be taught how to find these routes. A simple approach might involve slowly expanding outwards in all directions, investigating every nearby route before moving farther out from the start. However, this approach can be fairly slow as the computer will spend a lot of time looking west when its ultimate destination is east, and worse, it will need to keep track of all those different routes west, quickly running out of memory. A different approach may focus solely on moving forever towards its goal when it can, but this can create a problem if a mountain range, wall, rough terrain, or other obstacle is in the way. The computer may end up having to retrace steps to find a path around the obstacle, resulting in a longer-than-necessary route.

Combining these two concepts of looking at nearby routes first, but giving preference to routes that take you closer to your goal is one way you can reduce the pitfalls of each individual approach, and the basic idea behind the popular pathfinding algorithm known as "A*", or "A Star" (Russell & Norvig, 2010).

The A* Pathfinding Algorithm

A* is usually comprised of several different components:

A **frontier**, which is a list of nodes (cities, positions in a game world, or other forms of 'places' or 'states') that are the next available options to start exploring possible routes from (Rios & Chaimowicz, 2011). These are usually nodes that are at the ends of already discovered routes. The first node that is listed on the frontier before A* even starts running is the node you are starting from.

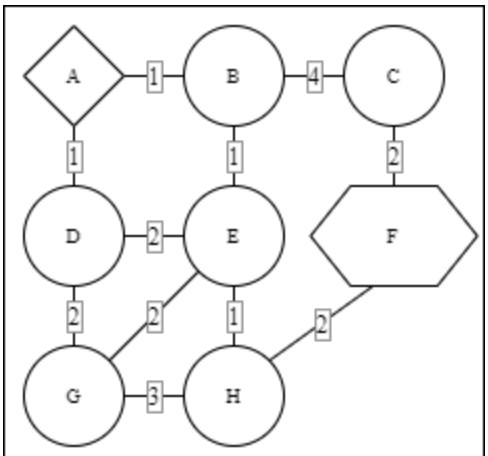
You may also need a **list of nodes** that have previously been on the frontier, as a node should never return to the frontier once it has been selected to be removed from the frontier list to be investigated (Rios & Chaimowicz, 2011).

A **route database** of currently known 'fastest' routes to each and every node the A* algorithm has discovered so far (Rios & Chaimowicz, 2011). It may be useful to think of this as the algorithm writing down directions to each city next to the city on a map. An A* algorithm traveling from San Francisco to New York City that is currently looking to see if Oklahoma City is a good choice of city to drive through would record the fastest route it has found so far from San Francisco to Oklahoma City as *the* only currently known route to Oklahoma City. This way if A* needs the route from San Francisco to Nashville, and Oklahoma City might be one city it could travel through, it already has the full route from San Francisco to Oklahoma City saved in its memory. If a faster route to Oklahoma City or any other node on the map is ever found, A* erases the old route and replaces it with the faster one, so it only ever tracks one route per node.

A method of calculating the **true cost** of a route to any node discovered so far (Rios & Chaimowicz, 2011). You could simply record the total cost of a set of directions to a city when you discover that route, or you could record the cost of each individual leg of the journey and

add those costs up each time you need the total cost of a known route, or any other option that gets you the real, actual cost from the start to any possible destination found so far.

An estimated remaining cost to go from the end of any currently known route to the goal, otherwise known as a **heuristic** (Rios & Chaimowicz, 2011). This estimate is best if it *never* overestimates the true cost, either in total remaining distance or in each individual leg of the remaining journey. Any overestimate may result in A* ignoring a path that potentially could be shorter than a different path it's choosing to explore instead. Returning to the Oklahoma City example, if you know the route to OKC and want to decide if OKC is a location worth exploring from, your heuristic may simply be the straight-line 'as-the-crow-flies' distance from OKC to NYC. Since you can't generally go a shorter distance than a straight line, and roads tend to not be perfectly arrow straight along long distances, this won't overestimate the eventual true cost, but will still come fairly close. The closer you can make your estimates to the eventual true answer *without going over*, the better A* will perform. If the cost you care about is fuel, rather than distance, you might do some calculations involving the maximum speeds each state between OKC and NYC has and its impact on fuel efficiency, multiplied by the distance, or some other calculation.

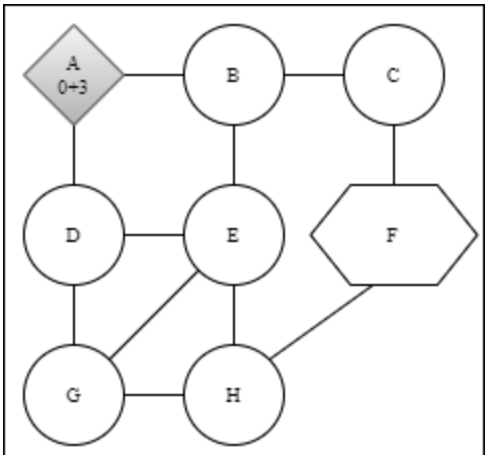


(Figure 1) Cities A-H, with connecting roads and and abstract costs of driving on those roads. A is the start. F is the destination.

How A* works:

At the beginning, the start of the path is placed on the **frontier**. (Figure 2)

A* then searches through the frontier, looking at each node listed, and taking the **known true cost** to reach that node and adding it to the **estimated remaining cost** to go from that node to the goal (Rios & Chaimowicz, 2011). At the very beginning, the only node on the frontier will likely have a known true cost of 'zero', because it's the start, and a high estimated remaining cost. Out of all the options on the frontier, it selects whichever node has the smallest total cost, and pulls that node off of the frontier, making sure to note that that node should **never** be placed on the frontier again.

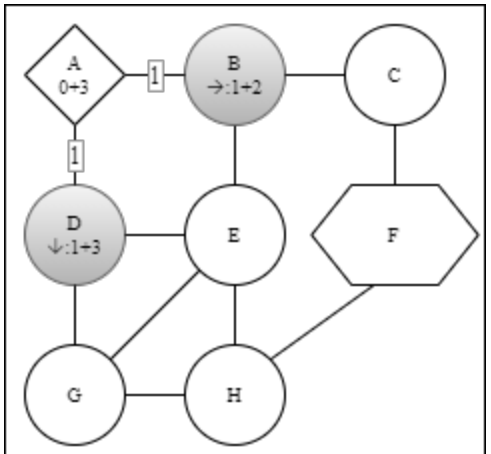


(Figure 2) City A is on the frontier. For now, lets assume each line only has a cost of 1. Numbers listed on A is 'true' cost plus estimated cost remaining.

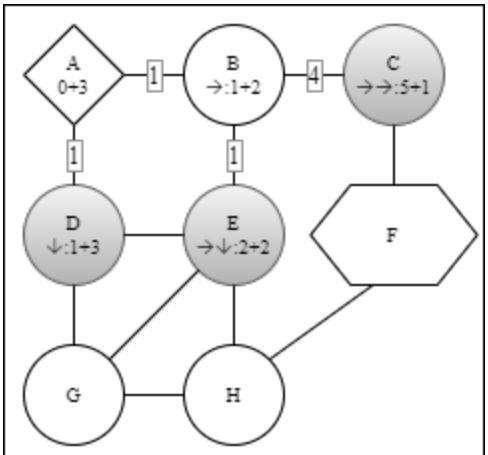
A* then searches every route out of that selected node (Figure 3), and records directions to each and every location it finds, replacing any previously discovered more expensive routes-known

with any cheaper ones it finds (Rios & Chaimowicz, 2011). For every destination it finds, it also places each of those destinations on the frontier if those nodes have never been on the frontier before.

A* then returns to the step where it searches through the frontier (Figures 4 through 8), looking for the node with the cheapest combination of known true cost and estimated remaining cost, and

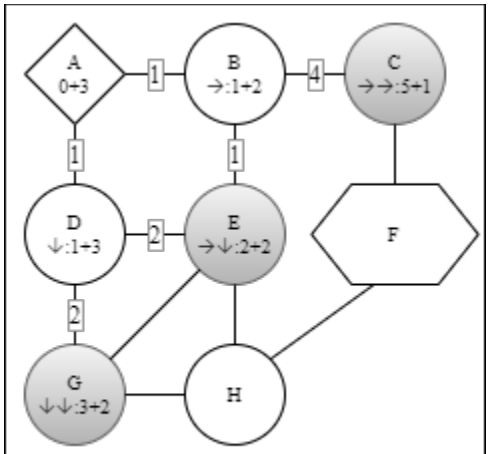


(Figure 3) Cities B and D are on the frontier. The directions to them have been recorded, along with the route cost and estimated remaining cost.

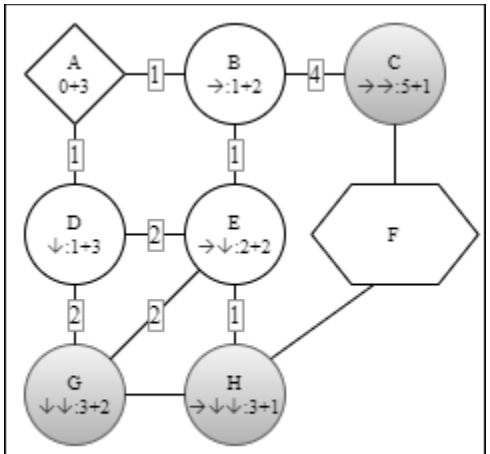


(Figure 4) City B only had two more 'steps' before reaching F, so lets explore from there first.

repeats the above process until the cheapest node it is pulling off of the frontier is the ultimate destination goal it was seeking (Rios & Chaimowicz, 2011). This goal (for example, NYC), having been placed on the frontier, will have a known route, and that known route will be the shortest, cheapest route from the start (San Francisco?) to the goal.



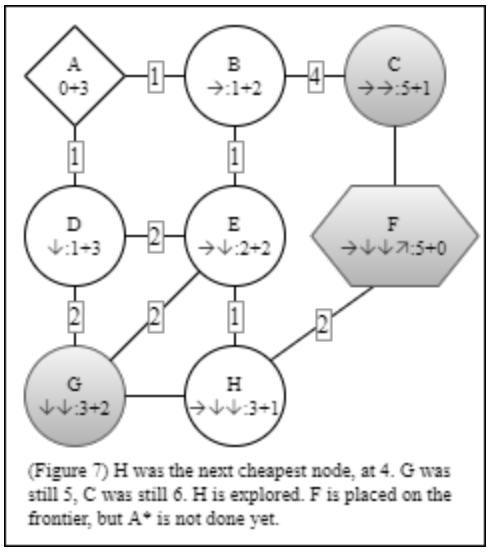
(Figure 5) Both D and E had equal costs (4), so either is a valid choice. C is far more expensive looking (6), so it is not explored yet.



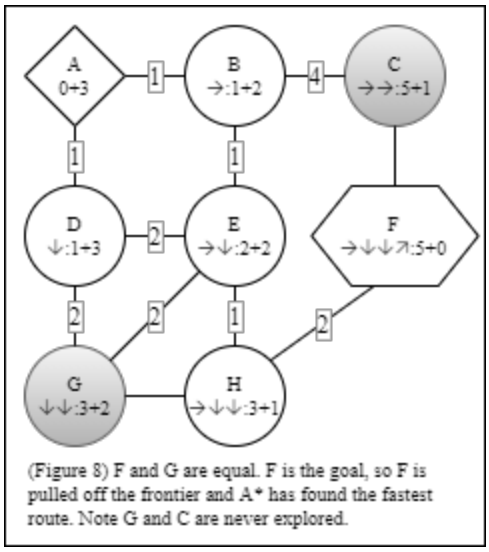
(Figure 6) E was the next cheapest node, at 4. G was 5, C was 6. E is explored.

Why A* works:

As A* finds routes to destinations, it finds *a* cost of reaching that destination. The true cost along that path increases. If that path ends farther from the goal than the step before it, both the true cost and estimated remaining cost increase, resulting in a very expensive combination for that node, virtually ensuring that route will be one of the last chosen from the frontier. If the node is closer to the goal, estimated remaining cost will decrease and the true cost so far will increase. Since estimates never overestimate, at best this trade-off will be even, but will often increase in true cost more than it decreases in estimated remaining cost. Then, if a particular path is expensive (a winding, rough road up and down several mountains), the true cost will increase far more than the estimated cost decreases, possibly making a slight detour a far cheaper node to look at.



(Figure 7) H was the next cheapest node, at 4. G was still 5, C was still 6. H is explored. F is placed on the frontier, but A* is not done yet.



(Figure 8) F and G are equal. F is the goal, so F is pulled off the frontier and A* has found the fastest route. Note G and C are never explored.

By continuing to look at only the cheapest 'best guess' routes while slowly replacing those estimates with true costs the closer to the goal you get, A* will inevitably find the fastest route from start to finish in far faster time and with less memory than more simple methods.

Works Cited

Rios, L. H. O., & Chaimowicz, L. (2011). PNBA*: A parallel bidirectional heuristic search algorithm. In *ENIA VIII Encontro Nacional de Inteligê ncia Artificial*.

Russell, S. J., & Norvig, P. (2010). A* search: Minimizing the total estimated solution cost. In *Artificial Intelligence: A Modern Approach* (3rd ed., pp. 93-99). Upper Saddle River, NJ: Prentice-Hall.