CSCI203

Week 9 – Lecture B

Tweaking Dijkstra

- As we have already seen, Dijkstra's algorithm provides an effective solution strategy for solving the single source/all destinations version of the shortest path problem.
- We will now look at a few simple modification of Dijkstra that will:
  - Improve its practical performance;
  - and/or extend its range of applicability.

Overall Efficiency

- The algorithm as you have seen it so far has efficiency $O(|V|^2 + |E|)$.
- But I said it was $O(|V|.\log|V|+|E|)$.
- How come?
- The answer is simple:
  - As presented we find the next vertex to select by searching a list of candidate vertices and finding the vertex with minimum *D* value.
    - This is a linear search process; $O(|V|)$.
    - We do this for each vertex, also $O(|V|)$.
    - This is $O|V^2|$.
- So, how do we improve on this?

Reaching Peak Efficiency

- The answer is surprisingly simple.
- Replace the candidate list/array *C* with…
- …a priority queue (aka a heap), ordered on $D(v)$.
- Now:
  - Finding the best candidate is O(1);
  - Updating *C* is $O(\log |V|)$.
- Now, over all vertices, we have $|V|.\log|V|$.

Single Destination.

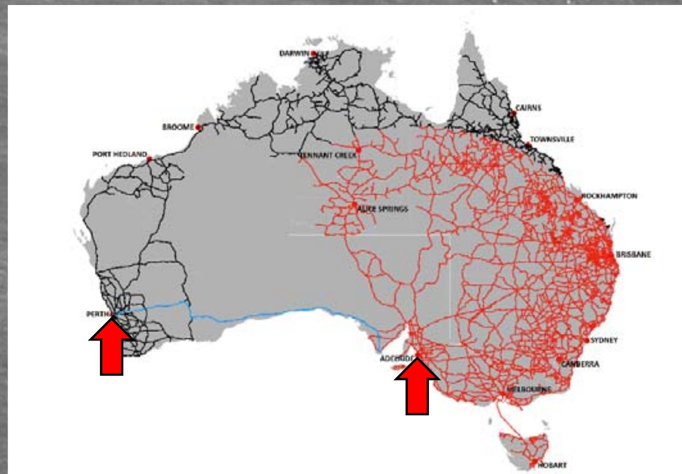
- What do we do if, rather than looking for the shortest paths from a start vertex, s, to all other vertices, we wish to find the shortest path from s to a specific goal vertex, ***g***?
- The answer is easy:
  - Stop when vertex *g* becomes a member of S, the selected set.
- This means that we do not waste time with any vertex further away from s than ***g***.
- This usually reduces the total running time of the algorithm.
  - Why *usually*?

All Sources—Single Destination

- In this case, rather than finding paths from a starting vertex, s, to all other reachable vertices we are looking for the shortest paths to some goal vertex, *g*, from all possible starting vertices.
- How do we do this?
  - Run Dijkstra backwards.
- Specifically:
  - Redefine Adj(*v*) to be the list of set of vertices leading to vertex *v*…
    - …instead of reachable from *v*;
  - Start with D(*g*)=0…
    - …instead of D(s)=0;
  - Let P(*v*) indicate the next vertex in the path…
    - …instead of the prior vertex;
  - Let the selected set, *S*, start at {*g*}…
    - …instead of {*s*}.

The Problem with Dijkstra

- There is a big problem with using Dijkstra on the single source/single destination problem:
  - The order in which the vertices are added.
- Consider the following graph:
  - Say we want to get from Adelaide…
  - …to Perth.
- Dijkstra will add all of the closer vertices first:
- Before we ever get close to the path we seek.



Fixing the Problem

- So, how do we remedy this?
- Before we get to the answer let us take a step back.
- Let us generalize Dijkstra's algorithm.
- The key step in the algorithm is on the process by which we select the next vertex.
  - Specifically, we select the vertex in the candidate set, C, for which the overall distance to the vertex from the source vertex, s, is minimized.
  - $D(v)=P(s, v)$.
- Note that this does not involve the goal vertex, g.

Eyes on the Goal

- What if we could bias the selection towards the goal in some way.
- We can!
- Essentially, we select the minimum not simply of $P(s, v)$ but, instead of $P(s, v) + H(v, g)$.
- This new function, H, is a *heuristic*; an estimate of the remaining distance from each candidate vertex, v, to the goal vertex, g.
- What is a good estimator?

A Good Heuristic

- We require $H(v, g)$ to have one key property:
  - $H(v, g) \leq P(v, g)$;
  - The heuristic estimate must never exceed the actual shortest path length.
  - This requirement guarantees that the final path we find is still the correct answer.
- In our example we have a ready-made heuristic…
- …The Euclidean (straight-line) distance between v and g.
- Provided the graph behaves according the rules of geometry there can never be a shorter path than this.

A*

- The heuristic modification to the vertex selection rule changes Dijkstra’s algorithm into an example of what is called the A* algorithm.
- Although, in the worst case, A* is no faster than Dijkstra in practice it will generally represent a substantial improvement.
- Note: the trick to A* is finding a good heuristic.
- The nearer that $H(v, g)$, the estimated minimum path length from v to g, is to $P(v, g)$, the actual minimum path length, the faster A* will find the solution.

Through the Looking Glass.

- Because of the order in which we saw them, it is easy to think of A* as a generalization of Dijkstra's algorithm.
- This is not the only, or perhaps even the best, way to view this.
- Consider instead this viewpoint:
- Dijkstra's algorithm is simply a special case of A*:
- The one with the worst possible choice of H.
- Specifically, $H(v, g)=0$.