DMA 12g

Carl Dybdahl, Patrick Hartvigsen, Emil Chr. Søderblom

February 1, 2017

Part 1

(a)

We need to show that given a path e_n with $0 \leq n < k$, we can rewrite $-\log(w(e_0) \cdot w(e_1) \cdot \cdots \cdot w(e_n))$ as a sum.

To do this, we exploit the rule $\log(a \cdot b) = \log(a) + \log(b)$. We reason as follows:

$$\begin{aligned} -\log(w(e_0) \cdot w(e_1) \cdot \cdots \cdot w(e_n)) &= -\log(w(e_0)) - \log(w(e_1) \cdot \cdots \cdot w(e_n)) \\ &= -\log(w(e_0)) - \log(w(e_1)) - \log(\cdots w(e_n)) \\ &= \ldots \\ &= -\log(w(e_0)) - \log(w(e_1)) - \cdots - \log(w(e_n)) \end{aligned}$$

(b)

To find the most probable path, we wish to maximize $\prod_{0 \leq n < k} w(e_n)$. This is equivalent to minimizing $-\log \prod_{0 \leq n < k} w(e_n)$, which we have just shown is the same as $\sum_{0 \leq n < k} -\log(w(e_n))$. This is equivalent to finding the shortests path with a weight function $w'(e) = -\log(w(e))$. We use Dijkstra's algorithm for this:

pathfind.pseudocode

```
-- assuming to functions on graphs:
--    G.out(n)    gives the nodes v with a connection [n -> v]
--    G.in(n)     gives the nodes v with a connection [v -> n]

PathFind (G, a, b):
   let Q = new PriorityQueue()
   for node in G.V:
      node.dist = infinity
    -- The first value is the distance/priority, second is the node.
    -- This queue always returns the element with least distance
    -- when popping.
   Q.push(0.0, a)
   while not Q.empty do:
      let (distance, node) = Q.pop()
```

```
      if node.dist > distance then:
         node.dist = distance
         for n in G.out(node) do:
            let cost = -log(G.weight(node, n))
            Q.push(distance + cost, n)
   if b.dist = infinity then:
      return null
   let path = [b]
   while path[0] != a do:
      let piece = member of G.in(path[0]) which minimizes piece.dist
      path = prepend piece to path
      return path
```

Part 2

(a)

$\delta_{BFS}(s, v)$ represents the length of the shortest path from s to v, whereas $\delta_{SP}(s, v)$ represents the cost of the lowest-cost path from s to v. However, when $w(e) = 1$, the cost of a path and its length coincides, and as such δ_{BFS} and δ_{SP} will be equal.

(b)

A node is black during BFS if it has been visited. This means that the processing for that node is done.

The set S represents the nodes for which the shortest distance has been determined. Similarly to a node in BFS being black, a node being in S means that the processing for that node is done.

(c)

The loop invariant states that at the start of each iteration of the while loop, `v.d = δ(s, v)` for each vertex `v`. When Dijkstra's algorithm visits a node `u`, it has already visited all nodes `w` with a distance $\delta(s, w)$ less than $\delta(s, u)$. This means that any node in V with distance less than $\delta(s, u)$ is already in S, and so the nodes `v` in $V \setminus S$ must have a distance greater than or equal to that of `u` from `s`.