Homework 12

Tushar Jain, N12753339
CSCI-GA 1170-003 - Fundamental Algorithms

December 10, 2017

Problem 1. In a directed graph $G = (V, E)$, each edge $(u, v) \in E$ has an associated independent value $r(u, v) \in \mathbb{R}$, $0 \leq r(u, v) \leq 1$ which represents how secure the channel from vertex u to vertex v is, i.e. the probability that the channel will not fail. Give the most efficient algorithm you can to find the most reliable path between two given vertices s and t, and state its running time.

Hint: Notice, here we need to maximize the product, while we know how to minimize the sum. To turn maximum into minimum, use $max_{(a_1,...,a_k)\in S}(a_1, ..., a_k) = min_{(a_1,...,a_k)\in S}(\frac{1}{a_1}, ..., \frac{1}{a_k})$. To turn product into sum, use $\log(ab) = \log(a) + \log(b)$. Make sure you carefully explain how you follow these hints.

Solution.
We can achieve this in many ways.

One of them is by simply defining our weight function as:

$$w(u, v) = -\log(r(u, v))$$

and then we can use any suitable using any **single-pair shortest path** algorithms which are basically the same as **single-source shortest path** like **Dijkstra's algorithm** using s as source which finds shortest path to every vertex in the graph, and then we can simply traverse the parent (π) pointers of the destination vertex (t) till we hit the source to get get the required shorted path from s to t. We can do this modification of the weight function because the logarithm is a monotonic function and it's property of changing internal product to external summation as mentioned in the hint.

The other option would be to modify the comparison statement in the relax function by reversing the comparison operator and using multiplication instead of addition to the following and then do Dijkstra's algorithm and other steps as stated in previous paragraph:

```
1 def relax(u, v, r):
2     if v.d < u.d * r(u,v):
3         v.d = u.d * r(u,v) # Actually Decrease-key in Q.
4         v.p = u
```

Running Time:
The total running time is equivalent to that of Dijkstra's algorithm (i.e. $O((V + E)\log(V))$)as both the above stated modifications take $O(1)$.

□

Problem 2. Tucker, who lives in a node s of a weighted graph G (with non-negative weights), is invited to an exciting party located at a node h, where he will meet a girl of his dreams, Sharona. Naturally, Tucker wants to get from t to h as soon as possible, but he is told to buy some beer on the way over. He can get beer at any supermarket, and the supermarkets form a subset of the vertices $B \subset V$. Thus, starting at s, he must go to some node $b \in B$ of his choice, and then head from b to h using the shortest total route

possible (assume he wastes no time in the supermarket). Help Tucker to meet Sharona as soon as possible, by solving the following sub-problems. Note that the graph can be either directed or undirected, so either give solutions for both cases separately, or justify why your solution works for both cases.

We can apply all of the following solutions to both directed and undirected graphs as Dijkstra's algorithm can be applied to both directed and undirected graph, and we're just using Dijkstra's.

Problem 2.a. Compute the shortest distance from s to all supermarkets $b \in B$.

Solution.

As we did in Q1, using Dijkstra's algorithm with source s to solve single source Shortest problem, we find shortest distance from s to all nodes including all supermarkets $b \in B$. □

Problem 2.b. Compute the shortest distance from every supermarket $b \in B$ to h. Can one simply add a new "fake" source s' connected to all supermarkets with zero-weight edges and run Dijkstra from s'?

Solution.

Now, our problem is a basic Single Destination Shortest Path problem which we can solve by reversing all the edges in the graph and then solving Single Source Shortest Path problem using Dijkstra's algorithm with source h, we find shortest distance from h to all nodes including all supermarkets $b \in B$ in the reversed graph. Due to the commutative nature of the distance, we can get the shortest from every supermarket $b \in B$ to h in the original graph.

Adding a a new "fake" source s' connected to all supermarkets $b \in B$ with zero-weight edges and running Dijkstras from s' would not help. This is because the distance from s' to h could go through any other b, which may not be the same b with the shortest distance from h, or thus, the path distances would be incorrect as they could form the incorrect paths. □

Problem 2.c. Combine parts (a) and (b) to solve the full problem.

Solution.
From (a) and (b), we get the distances for from s to all $b \in B$ (let's say d_1) and from all $b \in B$ to h (let's say d_2). the required answer would the corresponding path and sum of path distance which minimises $d_1(s, b) + d_2(b, h)$ for all $b \in B$. □

Problem 2.d. Your solution in part (c) used two calls to the Dijkstras algorithm (one in part (a) and one in part (b). Define a new graph G' on $2n$ vertices and at most $2m + n$ edges (and appropriate weights on these edges), so that the original problem can be solved using a single Dijkstra call on G'.

Solution.
This can be achieved by cloning the original graph G twice, let's call the clones G_1 and G_2. Now, we connect G_1 and G_2 to form the new require graph G' in the following manner:

- Let's assume Z to be the set of nodes which have an incoming edge from any $b \in B$, i.e.

$$b \to z, \text{ where } z \in Z$$

- Now we connect all b_1 i.e. $b \in B$ in G_1 to their respective adjacent outgoing node's clone, i.e $z_2 \in Z$ in G_2 and keep the same weights as original graph.

- Now running Dijkstra's algorithm with s_1 as source, we get the required distance and path to the final destination's clone, h_2.

This will work because the only connection from G_1 to G_2 is via supermarkets $b \in B$ in G_1 to their outgoing clones $z \in Z$ in G_2 and thus, the minimum distance returned by Dijkstra's will be a path going through one of the beer shops b. □

Problem 2.e. Assume now that, in addition to beer, Tucker also needs to buy flowers for the beautiful Sharona, and the set of flower shops is $F \subset V$. As with supermarkets, Tucker can choose any flower shop $f \in F$, and it does not matter if he buys first beer, then flowers or vice versa, so he will naturally choose the best among all these options (which supermarket, which flower shop, and in what order). Show how to solve this problem for the poor Tucker.

Solution.
This can be achieved in a very similar manner as above by cloning the original graph G 5 times and connecting in very similar manner as above but choosing flower shops or supermarket according to the connection. The total graph can be visualised as follows:

$$G_{fb} \leftarrow G_f \leftarrow G \rightarrow G_b \rightarrow G_{bf}$$

where connections are made with outgoing clones. Now, we run the Dijkstra's algorithm with source s in G and compare the distance between h in G_{bf} and G_{fb} choosing the one with lower distance and the corresponding path. □

Problem 3. Let G = (V, E) be a directed graph with weighted edges; edge weights could be positive, negative or zero.

Problem 3.a. Describe an $O(n^2)$ algorithm that takes as input $v \in V$ and returns an edge weighted graph $G' = (V', E')$ such that $V' = V \setminus \{v\}$ and the shortest path distance from u to w in G' for any $u, w \in V'$ is equal to the shortest path distance from u to w in G.

Solution:

In order to be able get the same shortest path distance after removing a vertex v from G, we would need to preserve the distances and paths that go through v.

- First, we copy all vertices except v and edges which dont have vertex v as one of the endpoints to a new graph G.
- Then we connect for every adjacent vertices u with incoming edges to v (i.e. $u \rightarrow v$) with adjacent vertices z with outgoing edges from v (i.e. $u \rightarrow z$) such that the weight of the new edge $u \rightarrow v$ is the sum of the weights of previous edges $w(u, v) + w(v, z)$. This would preserve the (u, z) distances.

Running Time:
As we need to traverse all vertices z for all vertices u, this would run in $O(n^2)$. □

Problem 3.b. Now assume that you have already computed the shortest distance for all pairs of vertices in G'. Give an $O(n^2)$ algorithm that computes the shortest distance in G from v to all nodes in V' and from all nodes in V' to v.

Solution:
Since we have already calculated the distance from u to w, for all $u, w \in G'$.

Let's consider finding the shortest distance from v to all other nodes first. As incoming edges to v won't effect this, we only consider outgoing edges from v. Therefore, for every vertex z in G', we traverse all the vertices u with an outgoing edge from v in G (i.e. $v \rightarrow u$) and find the minimum distances $w(v, u) + d(u, z)$ and update the distance (v, z) corresponding to it.

Now, let's consider finding the shortest distance to v from all other nodes. As outgoing edges from v won't effect this, we only consider incoming edges to v. Therefore, for every vertex z in G', we traverse all the vertices u with an incoming edge to v in G (i.e. $u \rightarrow v$) and find the minimum distances $d(z, u) + w(u, v)$ and update the distance (z, v) corresponding to it.
Running Time:
As for every adjacent vertex of v, we traverse all the vertices of the graph G',the algorithm runs in $O(n^2)$. □

Problem 3.c. Use part (a) and (b) to give a recursive $O(n^3)$ algorithm to compute the shortest distance between all pairs of vertices $u, v \in V$.

Solution:
To compute the shortest distance between all pairs of vertices $u, v \in V$, we can use the algorithms for part (a) and (b) above, i.e, for each $v \in V$ we can first remove the vertex (part a) and then find distance to and from it , v,for all other vertices (part b) to give us an all pairs shortest path algorithm.
Running Time:
As we do this for all vertices, the overall running time is $O(n^3)$. □

Problem 4. Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 US dollar buys 0.5 British pounds, 1 British pound buys 10.0 Hong Kong dollars, and 1 Hong Kong dollars buys 0.21 US dollars. Then, by converting currencies, a clever trader can start with 1 US dollar and buy 0.5 * 10.0 * 0.21 = 1.05 US dollars, making a profit of 5 percent.

Your bank gives you a list of m tuples, each tuple of the form (x, y, r) denoting that they will exchange 1 unit of currency x for r units of currency y. Assume there are n currencies in total. Give the fastest algorithm you can to determine if arbitrage is possible, and prove the running time and correctness.

Solution:
Similar to Q1, first we convert the product maximization problem (i.e. to see if profit can be generated) to sum minimization problem using $-\log(r)$ as the weight function where the nodes of the directed graph are currencies and r is the conversion value for an edge $x \to y$. Now, we need to detect if a negative exists in the graph, which can be easily done by applying the Bellman-Ford algorithm. Thus, the running time is also same as Bellman-Ford. □