Problem Set 8 Solutions

Problem 8-1. Arbitrage and Exchange Rates

This problem is analagous to the various shortest-path problems, with the following differences:

1. Instead of *summing* weights along paths, the exchange rate from u to v along a path is given by the *product* of the edge weights along that path. As such, we are interested in *largest-product paths*.
2. Instead of negative-weight cycles causing shortest paths to be undefined, cycles with product greater than 1 cause largest-product paths to be undefined.

A similar optimal substructure property is maintained, by a cut-and-paste argument: on any largest-product path from u to v, every sub-path is also a largest-product path between its endpoints. In both parts of this problem, we will simply modify existing shortest-path algorithms to correspond with this substructure property.

(A clever mathematical trick allows us to convert directly from a largest-product paths problem to a shortest paths problem: on a graph with edge weights w, construct new edge weights w' such that $w'(i,j) = -\log w(i,j)$. By properties of logarithms (monotonicity and $\log(ab) = \log a + \log b$), a shortest path under w' is a largest-product path under w. This is because if the length of a path under w' is ℓ, then the product of that path under w is 10^{-ell}. Minimizing ℓ maximizes $10^{-\ell}$. *Caveat:* This technique (which would work in practice) technically requires an infinite-precision $O(1)$-time algorithm for taking logs, which doesn't actually exist. However, a solution employing these ideas will receive full credit.)

(a) We modify the Bellman-Ford algorithm to detect cycles having product greater than 1 (such a cycle is necessary and sufficient for performing arbitrage). This requires us to change the code of INITIALIZE-SINGLE-SOURCE to initialize $d[v] \leftarrow 0$ for all v, then to set $d[s] \leftarrow 1$. We also modify RELAX to test if $d[v] < d[u] \cdot w(u,v)$, and if so, to set $d[v] \leftarrow d[u] \cdot w(u,v)$. Similarly, at the end of the Bellman-Ford algorithm, we check for each edge (u,v) whether $d[v] < d[u] \cdot w(u,v)$, and if so, output "ARBITRAGE." The running time of this algorithm is same as that of Bellman-Ford, i.e. $\Theta(VE)$.

(For the log-trick described above, we merely run Bellman-Ford with w', and output "ARBITRAGE" if there is a negative-weight cycle.)

(b) We suitably modify any all-pairs shortest path algorithm, e.g. Floyd-Warshall: simply replace line 6 with $d_{ij}^{(k)} \leftarrow \max\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} \cdot d_{kj}^{(k-1)}\right)$. Also, modify the definition of the original $W[i,j]$ to be 1 if $i = j$, $w(i,j)$ if $i \neq j$ and $(i,j) \in E$, and 0 otherwise ($(i,j) \notin E$).

We could also modify Johnson's algorithm to find a constant *multiple* of each exchange rate (instead of an additive term) so that every rate is *at most 1* (instead of non-negative), then run a modification of Dijkstra's algorithm.

(For the log-trick described above, we merely run any all-pairs shortest path algorithm with w' as the edge weights.)

To determine the best exchange paths, we simply use the predecessor matrix π in a similar way as for shortest paths.

Problem 8-2. Minimum Path Covers

(a) Suppose we construct G' from G as described in the problem statement, where all the edges have capacity 1. We first note that there is a correspondence between path covers in G and integral flows in G' (those for which the flow through every edge is an integer): for each path $(i_1, i_2, \ldots, i_m)$ in a cover, for each $j = 1, \ldots, m-1$ we can push a unit of flow from x_0 to x_{i_j}, to $y_{i_{j+1}}$, to y_0. This is an allowable flow because the paths are node-disjoint (and therefore edge-disjoint as well), so at most one unit of flow goes through each vertex and edge in G'.

Conversely, for any integral flow in G', every edge is either fully saturated or unused (because capacities are 1). For each (x_i, y_j) that is saturated, take the edge (i, j) in G. Now for every vertex i, there is at most one selected edge entering i, and at most one exiting i (this is because only one unit of flow can go through x_i, and similarly for y_i). Because G is acyclic, the set of edges has no cycles, and forms a path cover (where a vertex that is not incident to any of the selected edges belongs to its own path of length 0). The number of separate paths (components) is $|V| - f$, where f is the number of selected edges (also the flow that we started with).

Therefore we have established that every integral flow of f units corresponds to a path cover having $|V| - f$ paths, and vice versa. Then by finding a maximum flow and selecting the edges as above, we end up with a minimum path cover.

(b) The above algorithm does not work on graphs that have cycles. The problem is that some of the saturated edges in the flow may correspond to a cycle in the original graph, which cannot be part of a path cover. For concreteness, take a cycle on two nodes and run the above algorithm. The flow will be 2 units, but there clearly cannot be a path cover of $|V| - f = 0$ paths.

(In fact, this problem is known to be NP-hard, meaning that a polynomial-time algorithm for it would be a very surprising result.)