****

Alimulla Shaik
shal5122@colorado.edu
Algorithms problem set 5
No Collaborators
****

1. Sol.
We are given below variables,
‘k’ different transmission stations and ‘n’ drones.
Cost of activating station ‘j’ is a_j
Cost of having drone ‘i’ is communicating with station ‘j’ is $c_{i,j}$
We are asked to help Gru to find the minimum cost way to ensure that every drone is communicating with at least one station. This allows us to assume that we need to use every drone but we do not need to all the stations. We can consider this as constraint to convert our problem into integer programming problem.
As our problem focuses on finding minimum cost way, we need to minimize the both the costs ie, cost of activating a station and cost of each drone communicating with station.

I have expressed this problem in integer programming problem below.

Minimize $\sum_{j=1}^{k} a_j y_j + \sum_{j=1}^{k} y_j \sum_{i=1}^{n} c_{i,j}$

Subject to

$y_j = 0$ if station is not activated
$y_j = 1$ if station is activated

If drone ‘i’ is communicating with station ‘j’ then assume $Comm_{i,j} = 1$
And if all the ‘n’ drones are communicated with at least one station then,
For all $Comm_{i,j}$ $\sum_{j=1}^{k} Comm_{i,j}\, y_j >= 1$
According to the class lecture, here $y_j \in \{0, 1\}$ and to relax the above integer programming problem to linear programming we can consider $y_j \in [0, 1]$.
As mentioned in the class, we can use randomized rounding to round the y_j values.
With randomized rounding our constraints in the problem are still satisfied.
As required we have formulated given problem into integer programming problem and mentioned its linear programming relaxation.

2. a) Sol.
The goal in this problem is to find an s-t flow of amount at least ‘d’ that minimizes the total cost, or report that such a flow does not exist. Which means we need to allocate the target amount of flow ‘d’ from s to t and need to minimize the total cost of the units of flow f_e for all the edges $e \in E$.

For an edge (x, y) we are given that a per unit penalty is associated with edge as $p_{(x, y)}$ and capacity bound of each edge as $c_{(x, y)}$.
With help of network flow problem mentioned in the algorithm textbook, the problem is expressed in integer linear programming problem below.

$$\text{Minimize} \sum_{(x, y) \in E} p_{(x, y)} f_{(x, y)}$$

Subject to

i. The flow of any edge does not exceed capacity of any edge.
$f_{(x, y)} <= c_{(x, y)}$ for all edges $e \in E$.

ii. As mentioned in the network flow problem, The amount of flow entering vertex ‘u’ equals the amount of flow leaving ‘u’ to ensure flow is conserved.

$$\sum_{(x, y) \in E} f_{(x, y)} = \sum_{(y, z) \in E} f_{(y, z)}$$

iii. The sum of the flow from the source vertex ‘s’ to other vertices must be at least the target flow ‘d’ and at the same time the sum of the flow from other vertices to the sink vertex ‘t’ must be at least the target flow ‘d’.

$$\sum_{u \in V} f_{(s, u)} = \sum_{u \in V} f_{(u, t)} >= d$$

For a particular flow, if above all the constraints are satisfied for any given graph then there exist a flow ‘s-t’ of amount at least ‘d’ that minimizes the total cost otherwise above solution reports that no such flow exists.

b) Sol.

The classic network flow problem focuses on maximizing the flow from ‘s-t’ and is related to ‘s-t’ shortest path problem described in 2) a) which focuses on minimizing the total cost.

Let’s look into maximizing the flow and minimizing the flow cases below.
Maximizing the flow case:
Classic network flow problem already focuses on maximizing the flow from ‘s-t’. For this problem, we do not need to consider a penalty $p_{(x, y)}$ per unit as we do in s-t shortest path problem.
We can maximize the flow in s-t shortest path problem as well. In that case we need to maximize the f_e for all the edges $e \in E$.

Minimizing the flow case:
‘s-t’ shortest path problem described in 2) a) focuses on minimizing the total cost of the edges. In the classic network problem, we can minimize the flow from ‘s-t’ by minimizing the per unit penalty and cost of each edge in the graph.

s-t shortest path problem is an application of network flow problem which allows to send a particular amount of flow from source to destination at the lowest possible cost.
So from the above two cases we can consider that both classic network problem and s-t shortest path problem are related and special cases for this problem.

3. Sol.
In the class lecture, professor discussed linear programming relaxation providing an example of vertex cover problem. The given problem seems similar to this example. In the vertex cover problem, we need to find subset of vertices which covers every edge. But here we are asked to find maximum-weight subgraph which is triangle free and need to minimize the total weight of the edges removed from graph. So we can use same linear programming representation with relaxation of vertex cover problem here to solve this triangle free problem.

Based on integer programming representation of vertex cover problem, we can represent the given problem as below.

Minimize $\sum_{(x, y) \in E} w_{(x, y)}\, c_{(x, y)}$

Subject to

$c_{(x, y)} = 0$ if edge (x, y) is not removed from the graph
$c_{(x, y)} = 1$ if edge (x, y) is removed from the graph

If there exists triangle in given graph then we need to remove at least one edge to remove that triangle from the given graph as required. This gives us a constraint that at least one edge need to be removed in order to maintain triangle free property in the graph.
Let’s assume x,y,z vertices form triangle in the graph then for the below edges, at least one need to be removed.

For all {(x, y), (y, z), (z, x)} $c_{(x, y)}+c_{(y, z)} +c_{(z, x)} >= 1$

To determine triangle free subgraph, first we need to find all triangles possible in the given graph and then try to remove lowest weight edge so that triangle does not exist in the given graph. We can find the triangle in the graph by verifying adjacency list of vertices.
For example, consider below adjacency lists where vertex x has edges y and z, vertex y has edges z and x, vertex z has edges x and y. As you can see, triangle exists in the graph with vertices x, y and z. So if adjacency list of ‘x’ has ‘y’ and adjacency list of ‘y’ has ‘z’ then it is enough to say that triangle exists in the graph.

x	y z
y	z x
z	x y

Algorithm:

Steps:
1. Consider the graph G = (V, E).
2. For each ‘x’ in vertex list V, check adjacency list of vertex ‘x’.
3. For each ‘y’ in adjacency list of ‘x’, check adjacency list of ‘y’.
4. For each ‘z’ in adjacency list of ‘y’,
5. If ‘x’ in ‘z’ then there exist triangle in ‘G’.
6. Compare weight of edges which formed triangle and pick the lowest weight edge.
7. Update adjacency list with removed edge and continue step 2 so that all vertices in ‘G’ are traversed.

Correctness:
The basic case of this problem is that when it have one triangle in the graph ‘G’. In this case, our algorithm finds edges that formed triangle in ‘G’ and finds lowest weight edge which can be removed to ensure no triangle exists in ‘G’. This helps us to get subgraph which is triangle free. As our algorithm removes one triangle from the given graph then we can generalize it that it will find all triangles possible in any given graph and removes them all.

In the worst case scenario where all vertices are connected to all other vertices in the graph. In this case our algorithm finds maximum number of triangles possible in graph and still holds good to find lowest weight edge to remove which ensure triangle free subgraph.

Runtime:
In our algorithm, we are traversing each vertex and then verifying two adjacency lists. As the graph is undirected graph and adjacency list will take time |V| -1.
So our algorithm has running time of $O(|V|^3)$

Proof of approximation:
Above, we have represented given problem in integer linear programming.
And $c_{(x, y)} \in \{0, 1\}$. To relax the above integer programming problem to linear programming we can consider $c_{(x, y)} \in [0, 1]$.
In the linear programming we will deal with fractions of $c_{(x, y)}$ and as mentioned in the class, we can use randomized rounding to round the $c_{(x, y)}$ values.
With randomized rounding our constraints in the problem are still satisfied.

In the class, professor discussed about 2-approximation of vertex cover problem. As given triangle free problem is similar to vertex cover problem then we can relate both problems.
As discussed in the class, we can consider $c^*_{(x, y)}$ as feasible linear programming solution (LP) and $c'_{(x, y)}$ as feasible linear programming solution after rounding the fraction values using randomized rounding.
We can define $c'_{(x, y)} = 1$ if $c^*_{(x, y)} >= 1/3$
$c'_{(x, y)} = 0$ if $c^*_{(x, y)} < 1/3$

From the above considerations, we can express LP solution of given problem as below.

LP solution of the given problem is $\sum_{(x,y)\in E} w_{(x,y)}\, c'_{(x,y)} <= \sum_{(x,y)\in E} w_{(x,y)}\, (3)\, c^{*}_{(x,y)}$

$\sum_{(x,y)\in E} w_{(x,y)}\, c'_{(x,y)} <= 3 \sum_{(x,y)\in E} w_{(x,y)}\, c^{*}_{(x,y)}$

Here $c^{*}_{(x,y)}$ can be expressed as below ILP.

$\sum_{(x,y)\in E} w_{(x,y)}\, c'_{(x,y)} = 3 \sum_{(x,y)\in E} w_{(x,y)}\, c_{(x,y)}$

LP = 3 (ILP)

LP / ILP = 3

Which is in the form of ALG / OPT = 3 So we conclude that LP algorithm is 3 - approximation.

5) Sol.

We are given that the trench is extremely long and narrow, running East-West and we need to guide robot Ada to find an ancient artifact.

Basic algorithm:

1. Place Ada in the middle of the trench and Ada will travel from the east to west
2. If ancient artifact is present in the path then return from here.
3. Otherwise travel back from west till it reaches the end of east.
4. This will ensure Ada travelled the whole trench and if there is ancient artifact present then it will be found otherwise return from here.

Optimal Algorithm:

1. Place Ada at the one end of path either east or west and start traversing till it reached the other end.
2. If ancient artifact is present in trench then it will be found as we covered the whole trench.

Correctness:

In the basic algorithm, we placed Ada at the best guess which is at the middle of trench from east to west. If ancient artifact present in that direction then it will be found and it is the best case. In the worst case, let's say ancient artifact present in the other path which is in opposite direction then Ada need to travel back to find it. Which means Ada travelled one half path twice because it could not find it in the assumed first half path.

In the optimal algorithm, we placed Ada at the one end of the path either east or west and starts traversing to look for ancient artifact. If ancient artifact is present in trench then it will be found as we covered the whole trench. In the best case, it will be found in travelled direction with less steps. In the worst case it will be found with more steps meaning it present near to the opposite end. In all the cases our algorithm found artifact if it is present in the trench. Both basic and optimal algorithm guarantees to find the artifact.

Running Time:

In the basic algorithm, Ada will travel the half trench and the whole trench in the worst case. For the whole it costs O(n) and for the half it costs O(n/2). So total running time is O(n).

In the optimal algorithm Ada will travel the whole trench so running time is O(n).

References:
1. http://beust.com/algorithms.pdf
2. http://www.eecs.berkeley.edu/~luca/cs261/lecture07.pdf