CSCI 5454 Problem Set 5
Robert Werthman
No Collaborators

---

1. *Formulate Gru's problem as an integer programming problem and write its linear programming relaxation.*

Integer Programming Problem

Assume each drone can communicate with any number of the k stations. The constraint for this problem is that we have to use every drone but we don't have to use every station. For this problem we want to minimize the sum of the cost of activating a station a_j and the cost c_{ij} of each of the drones communicating with that station. In mathematical terms we are trying to

$$\text{minimize} \sum_{j=1}^{k} a_j x_j + \sum_{j=1}^{k} x_j \sum_{i=1}^{n} c_{ij}$$

subject to the constraints

(a)
$$x_j = \begin{cases} 1 & \text{if we activate station } j \\ 0 & \text{if we don't activate station } j \end{cases}$$

(b) Let $d_{ij} = 1$ if drone i is communicating with station j then the mathematical notation for ensuring all drones are communicating with at least one station is

$$\forall d \quad \sum_{j=1}^{k} d_{ij} x_j \geq 1$$

Linear Programming Relaxation

The linear programming relaxation of this problem is still trying to

$$\text{minimize} \sum_{j=1}^{k} a_j x_j + \sum_{j=1}^{k} x_j \sum_{i=1}^{n} c_{ij}$$

but instead of $x_j \in \mathbb{Z}$ and being only 1 or 0 it can now be any number between 1 and 0. x_j is now $\in \mathbb{R}$. The question now though is how do you round x_j to make sure the constraint

$$\forall d \quad \sum_{j=1}^{k} d_{ij} x_j \geq 1$$

is still satisfied. As was discussed in lecture you should use randomized rounding.

2. *Find an $s - t$ flow of amount at least d that minimizes the total cost, or report that such a flow does not exist.*

   (a) *Show how to solve this problem using linear programming.*

   In this problem we want to send the target amount of flow d from s to t. At the same time we want to minimize the total cost of the units of flow f_e for all $e \in E$. Written mathematically this is

   $$\text{minimize} \sum_{(u,v)\in E} p_{u,v} f_{u,v}$$

   subject to the contraints

   i.
   $$f_{u,v} \leq c_{u,v}$$
   The flow of an edge is less than or equal to the capacity of the edge.

   ii.
   $$\sum_{v\in V} f_{s,v} = \sum_{v\in V} f_{v,t} = d$$
   The sum of the flow from the source vertex s to other vertices must equal the target flow d and the sum of the flow from other vertices to the sink vertex t must equal the target flow d.

   iii.
   $$\sum_{(w,u)\in E} f_{w,u} = \sum_{(u,z)\in E} f_{u,z}$$
   The amount of flow entering u equals the amount of flow leaving u. Flow is conserved [2].

   If these constraints are satisfied by a graph with a particular flow then an $s - t$ flow of amount at least d that minimizes the total cost exists, otherwise no such flow exists.

   (b) *Show that the $s - t$ shortest path problem and the classic network flow problem are both special cases of this problem.*

   The classic network flow problem seeks to maximize the flow from s to t. This is the same as the problem $2a$ but in this case we want to maximize f_e for all $e \in E$ by letting d be the maximum flow possible in the graph. We also no longer suffer

a penalty $p_{u,v}$ per unit flow for the classic network flow problem as we do in the problem from $2a$.

The $s-t$ shortest path problem seeks to minimize the sum of the cost of the edges involved in the flow $s-t$. In this case we no longer have a target flow d as in the problem $2a$ but were are still trying to minimize the penalty/cost $p_{u,v}$ per edge per unit flow in the graph.

3. *Give a polynomial time 3-approximation algorithm that, given an undirect graph G with nonnegative edge weights, removes a set of edges of minimum total weight from G so the remaining graph is triangle free. Return the maximum-weight subgraph which is triangle free.*

Algorithm

I believe you can solve this problem with linear programming specifically integer linear programming. We wish to minimize the total weight of the edges we remove from the graph. This can written as

$$\text{minimize} \sum_{(u,v)\in E} w(u,v)x_{u,v}$$

subject to the constraints

(a)

$$x_{u,v} = \begin{cases} 1 & \text{if we choose to remove edge } (u,v) \\ 0 & \text{if we don't choose to remove edge } (u,v) \end{cases}$$

(b)

$$\forall\{(u,v),(v,x),(u,x)\} \quad x_{u,v} + x_{v,x} + x_{u,x} \geq 1$$

This says that for all edges that form a triangle at least one edge has to be removed in order for there not to be a triangle anymore.

First, we need to find all of the triangles in the graph G and then remove an edge so the traingle no longer exists. This can be done by comparing the adjacency list of vertex u with the adjacency lists of the vertices in u's adjacency list. For example if u's adjacency list was $[v,x]$, v's adjacency list was $[x,u]$, and x's adjacency list was $[u,v]$ there would be a triangle in the G between (u,v), (v,x), (x,u). The psuedocode would look like

```
def FindAndRemoveTraingles(G):
  for each u in V:
    for a in u's adjacency list:
      for b in a's adjacency list:
        # first iteration: a = v, b = x
        if u in b's adjacency list:
```

```
            # There is a triangle between (u,a), (a,b), (b,u)
            # Compare weight of edges and remove lowest weight edge
            # Update adjacency lists
    return G
```

Correctness and Runtime

The worst case graph G for this problem is a graph that is connected which means all vertices have edges to all other vertices in the graph. This maximizes the number of triangles a graph will have.

The base case is when there are 3 vertices in G which forms a single triangle. FindAndRemoveTriangles will remove a single edge from the triangle that has the lowest weight. Once that edge is removed there is no longer a triangle and the graph is triangle free.

Inductively we can say that since the algorithm removes a single triangle it will remove all of the triangles in a graph. This is true because FindAndRemoveTriangles visits every vertex and finds every triangle in the graph. For every one of these triangles an edge is removed thus removing the triangle. This will leave any graph with any number of triangles triangle free.

Because the graph is connected the size of each adjacency list would be $|V| - 1$. The runtime of FindAndRemoveTraingles will then be $O(|V|^3)$.

Approximation Proof

Since we are trying to minimize for this problem in order to prove that this algorithm is a 3-approximation algorithm, we have to show that the value of the solution x which is the total weight of all the edges removed by the algorithm is at most 3 times the value of the optimal solution. This statement shown by the inequality

$$\frac{ALG}{OPT} \leq 3$$

As stated above we found a solution with integer linear programming. But what if we relaxed the ILP problem to just a LP problem. This means that $x_{u,v}$ is now $\in \mathbb{R}$

and could be any real number between and including 0 and 1. What if we also said that the solution to the ILP problem was the optimal solution OPT. This means that $LP \leq ILP = OPT$ [1].

The constraints we have for the LP problem are

(a) $0 \leq x_{u,v} \leq 1$

(b)
$$\forall\{(u,v),(v,x),(u,x)\} \quad x_{u,v} + x_{v,x} + x_{u,x} \geq 1$$

This says that for all edges that form a triangle at least one edge has to be removed in order for there not to be a triangle anymore.

In the LP problem though we are now dealing with possible fractions for $x_{u,v}$. This means we will have to round those fractions to a whole number in order for the LP problem solution to still remain a solution to the ILP problem. To do this we define $x'_{u,v} = 1$ if $x^*_{u,v} \geq 1/3$ and $x'_{u,v} = 1$ if $x^*_{u,v} < 1/3$ [1].

Using this rounding scheme we can show that the LP is a 3-approximation algorithm. The minimal weight produced by the LP solution is

$$\begin{aligned}
&\sum_{(u,v)\in E} w(u,v)x'_{u,v} \\
&\leq \sum_{(u,v)\in E} 3w(u,v)x^*_{u,v} \\
&= 3\sum_{(u,v)\in E} w(u,v)x_{u,v} \\
&= 3\,OPT
\end{aligned}$$

This shows that

$$\frac{LP = ALG}{ILP = OPT} \leq 3$$

which means the LP algorithm is a 3-approximation algorithm.

References

[1] http://www.eecs.berkeley.edu/ luca/cs261/lecture07.pdf

[2] http://beust.com/algorithms.pdf