Pablo Acuna
CSCI 4020
Anshelevich

Computer Algorithms Homework 5

9. Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time.

Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now).

At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most $\lceil n/k \rceil$ people.

Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

Set-Up: There are nodes u_i $\forall$ i $\leq$ n that represent each person. Likewise, there are nodes v_j $\forall$ j $\leq$ k that respresent each hospital. There is an edge (u_i, v_j) if the patient i is 30 minutes away from hospital k. All these edges have capacity of 1. To convert this to a network flow problem, we will add a source node, s, which has an edge to every u_i. These edges will also have edge weight 1 each. And finally we will add a sink node, t, which has an edge to every v_j. These will have a capacity $\lceil n/k \rceil$ each.

Solution: To determine if there is a feasible solution to the scenario, if you get max flow of size n, then it is possible to transport the n people to the hospitals without overloading them.

Run Time: Feeding our graph in the set-up to the Ford Fulkerson algorithm we get a runtime of O((n+k)(nk))

Proof: The algorithm does not violate the capacity constraints, especially on edges (w_j, t) due to the load constraint. This means that no hospital can be overloaded. Conversely, if there is a flow of value n, then there is one with integer values, so that corresponds to some path for assignment of hospitals. We send patient i to hospital j if the edge (v_i,w_j) caries one unit of flow, and we observe that the capacity condition ensures that no hospital is overloaded.

Pablo Acuna
CSCI 4020
Anshelevich

Computer Algorithms Homework 5

14. We define the Escape Problem as follows. We are given a directed graph G = (V,E) (picture a network of roads). A certain collection of nodes X $\subset$ V are designated as populated nodes, and a certain other collection S $\subset$ V are designated as safe nodes. (Assume that X and S are disjoint.) In case of an emergency, we want evacuation routes from the populated nodes to the safe nodes. A set of evacuation routes is defined as a set of paths in G so that (i) each node in X is the tail of one path, (ii) the last node on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated nodes to "escape" to S, without overly congesting any edge in G.

(a) Given G, X, and S, show how to decide in polynomial time whether such a set of evacuation routes exists.

Set-Up: Given Graph G = (V,E), we will make each edge weight 1 because the problem states that evacuation routes should not overlap edges, therefore making the edge weights 1 solves this problem. Now to convert this to a Network Flow problem we will add a source node, s, and have a edge from s to all nodes in X each with an edge weight of 1. This is because we need to create $|X|$ escape routes, (s, X_i). We will finally add a sink node, t, this node will connect to each node in $|S|$, (S_j, t).

Solution: To determine if there is a feasible solution to the scenario, if you get max flow of size $|X|$, then there are $|X|$ unqiue routes from X to S that don't share any edges.

Run Time: Feeding our graph in the set-up to the Ford Fulkerson algorithm we get a runtime of O(VE^2)

Proof:

If evacuation routes exist, max-flow=$|X|$.

Use $|X|$ unit capacity edges from source to each of the nodes in X. From there, we have a path from each node in X to some node in S using unique edges. Thus the $|X|$ units of flow can reach nodes in S from where they have $|X|$ capacity paths to the sink.

If max-flow=$|X|$, evacuation routes exist.

If max-flow=$|X|$, each node in X is receiving unit capacity flow from s, and since all this flow is reaching t and each edge between X and S is unit capacity, all of this flow must be using unique edges. Thus we have a unique path from every node in X to some node in S.

(b) Suppose we have exactly the same problem as in (a), but we want to enforce an even stronger version of the "no congestion" condition (iii). Thus we change (iii) to say "the paths do not share any nodes." With this new condition, show how to decide in polynomial time whether such a set of evacuation routes exists. Also, provide an example with the same G, X, and S, in which the answer is yes to the question in (a) but no to the question in (b).

Set-Up: Given Graph G = (V,E), we want to make sure no two routes go through the same nodes. So, to achieve this we will split all the nodes not in X or S into two nodes, u_{i1}, u_{i2}. u_{i1} will have all the incoming edges that u_i had; and u_{i2} will have all the outgoing edges that u_i had. We will then create an edge (u_{i1}, u_{i2}) that will connect the two nodes. This ensures that only one route can go through a node.

Solution: Now we will use part a as a blackbox and we will have a possible escape route for all nodes in X if the max flow is equal to $|X|$.

Run Time: Feeding our graph in the set-up to the Ford Fulkerson algorithm we get a runtime of O(VE^2)

Proof: You can use the same proof as part one for this one because each path uses unique edges, therefore only one path can go through a node since to reach a node you have to cross a unique edge of weight 1.

Example: An example in which part a will give you a solution and not b. X = {a,b}, S = {d,e}, V= {a,b,c,d,e}. E = {(a,c),(b,c),(c,d)(c,e)}. This graph has 2 routes from X to S using unqiue edges, therefore a is yes, but they both use the node c to get to S, therefore b is no.