Oliver Tonnesen
V00885732

Assignment 3

CSC 425
November 16, 2020

1

We construct a graph in polynomial time to which we can apply a network flow algorithm to solve the problem.

First calculate the distance between each pair of clients and stations. Add a vertex for each client and each station. Add an edge between any client-server pair within r of each other, and give it capacity 1. Finally add a source vertex adjacent to each client vertex with edge capacities of 1, and add a sink vertex adjacent to each station with edge capacities of L.

We run a network flow algorithm on this graph to obtain the value of its maximum flow, say v. If v is equal to the number of clients, then our algorithm returns true, otherwise false.

2

Our main idea is to construct an edge-capacited graph H whose maximum flow (in the original sense) is the same as G's maximum flow (in the vertex-capacited sense).

Roughly, we do this by "doubling up" all the vertices and transforming each vertex into an edge. For each vertex v, we add a vertex v' with an edge from v to v' with capacity c_v. Any edge going to v still do so, but any edges coming from v now instead come from v'. That is, if (i, j) is an edge in G, it becomes (i', j) in H. The edges between v and v' account for all the bottlenecks in G, so we can give the remaining edges infinite capacity (or more practically, $|v| \cdot \max\{c_v | v \in V\}$).

We define an $s - t$ cut in a vertex-capacited network analogously to an edge-capacited network: a partition of the edges into two sets.

Our cut-set analogue is the set of vertices with at least one edge in either partite set.

Theorem: *Max-flow min-cut for vertex-capacited graphs*:

$$\max\{f \mid f \text{ a feasible flow}\} = \min\{c(S, T) \mid (S, T) \text{ an } s - t \text{ cut}\}$$

We know the original max-flow min-cut theorem holds for H. Furthermore, a min-cut for H will certainly not use any of the edges with infinite capacity, and the rest of H's edges are in bijection with the vertices of G. This means that a min-cut in H can be translated into a min-cut for G. This – along with the fact that the max-flows in G and H have equal values – proves the max-flow min-cut theorem for vertex-capacited graphs.

3

We give a reduction from a known NP-complete problem: Vertex Cover $\leq_P$ Hitting Set.

Let (V, E, k) be an input to Vertex Cover. Let $A = V$, and for each $(v_i, u_i) \in E$, let $B_i = \{v_i, u_i\}$.

We show that $(A, B_1, \ldots, B_m, k)$ is an accepting input to Hitting Set if and only if (V, E, k) is an accepting input to Vertex Cover:

($\Longrightarrow$): Let H be a hitting set of size at most k for $B_1, \ldots, B_m$ Now consider the subset $C \subseteq V$ corresponding to H. For any edge e_i corresponding to B_i, $H \cap B_i \neq \emptyset$, so C covers that edge. Thus C is a vertex cover with size at most k, and so (V, E, k) is indeed an accepting input to Vertex Cover.

($\Longleftarrow$): Let $C \subseteq V$ be a vertex cover of size at most k for (V, E). Now consider the subset $H \subseteq A$ corresponding to C. For any subset B_i corresponding to edge e_i, at least one endpoint of e_i is in C, so at least one element of B_i is in H. Thus H is a hitting set with size at most K, and so $(A, B_1, \ldots, B_m, k)$ is indeed an accepting input to Hitting Set.

4

We give a reduction from a known NP-complete problem: Independent Set $\leq_P$ Path Selection:

Let $G = (V, E)$ be an input to Independent Set. We construct the input $(D, P_1, \ldots, P_c, k)$ (where $D = (V, E)$ is a directed graph) as follows:

Order the edges of G arbitrarily by $e_1, \ldots, e_m$, and for each edge e_i, add a vertex u_i to D. For each vertex v_i in G, add two more vertices to D: w_i and x_i. Draw a path from w_i to x_i, using (in ascending order) the vertices in D corresponding to the edges adjacent to v_i in G. Let $P_1, \ldots, P_n$ be the paths we constructed between each w_i and x_i.

We claim that G has an independent set of size k if and only if D has k nonintersecting paths. We prove our claim in two parts:

($\Longrightarrow$): Let S be an independent set of size k in G. Let $v_1, \ldots, v_k \in S$. None of $v_1, \ldots, v_k$ share any edges, so none of the k paths $w_1 - x_1, \ldots, w_k - x_k$ in D share any vertices. Thus they form a set of k nonintersecting paths in D.

($\Longleftarrow$): Let $P_1, \ldots, P_k$ be a set of k nonintersecting paths in D. We claim $S = \{v_1, \ldots, v_k\}$ is an independent set in G. Assume for a contradiction that this is not the case. Then there exist vertices $v_i, v_j \in S$ that share an edge. Let $e = (v_i, v_j)$, and let u be the vertex in D corresponding to e. By the construction of D, $u \in P_i$ and $u \in P_j$, a contradiction, since all k paths are nonintersecting. So indeed S is an independent set in G.

5

(1) Assume not. Then $u \notin C$, so each of u's $> k$ neighbours must be included in C to cover all the edges incident to u. This is a contradiction, since $|C| \leq k$, and so it must be the case that $u \in C$.

(2) A set of vertices can cover no more than k edges per vertex in the set, so any set of k' vertices in G covers at most kk' edges. Thus if G has more than kk' edges, a vertex cover cannot have fewer than $k' + 1$ vertices.

We use the above two facts in the design of an algorithm for the fixed parameter variant of Vertex Cover. The main idea is this: If m is the number of edges in G, and k' is the largest degree, then if $\frac{m}{k'} > k$, then there exists no vertex cover of size k or smaller. Otherwise, we can remove a maximum-degree vertex from G and check the above criteria again. We keep repeating this process until at most a constant number of vertices remain, at which point we can simply use a brute force search of the remaining graph to check if a vertex cover of the desired exists in constant time. A more concrete algorithm is given below:

Note: For brevity, we use the notation $\Delta(G)$ to denote the maximum vertex degree in G.

```
function VERTEX-COVER-FPT(G, k)
    C ← ∅
    while Δ(G − C) > k − |C| do
        v ← vertex with maximum degree
        C ← C ∪ {v}
    if (k − |C|)^2 < m then
        return false
    else
        check all subsets of G − C
        if G − C has a vertex cover of size at most k − |C| then
            return true
        else
            return false
```

If we're able to find a vertex cover of $G-C$ with size $k-|C|$, then we can add its vertices to C to get a vertex cover for all of G with size at most k, so our algorithm is correct.

In the while loop, we look at any edge or vertex $O(1)$ times, so the entire loop runs in $O(n+m)$. After the loop, $G-C$ has fewer than $(k-|C|)^2$ edges left, so no more than $2(k-|C|)^2$ vertices remain, and checking all subsets takes $O(2^{2(k-|C|)^2}) = O(2^{2k^2})$. Thus our algorithm runs in time $O(n+m+2^{2k^2})$, as desired.