6.046/18.410 Problem Set 6

Yijun Jiang
Collaborator: Hengyun Zhou, Eric Lau

October 25, 2015

1 Shipping to Save the World

1.1 Part (a)

1.1.1 Subpart (i)

Description:

Create a tripartite graph $G = (V, E)$, where $V = D \cup S \cup C$. D is the set of day vertices, S the set of ship vertices and C the set of city vertices. $|D| = 7, |S| = m$ and $|C| = n$. If on day d_i a ship s_j can operate, create a directed edge (d_i, s_j). If a ship s_j can reach the city c_k, create a directed edge (s_j, c_k).

Construct a flow network by adding a source s and a sink t. For every $d_i \in D$, create a directed edge (s, d_i) with capacity $c(s, d_i) = v_i$. For every $c_k \in C$, create a directed edge (c_k, t) with capacity $c(c_k, t) = f_k$. Let V' and E' be the new sets of vertices and edges, and $G' = (V', E')$. All edges between D and S as well as between S and C have capacity ∞.

Run a maximum flow algorithm (e.g. Edmonds-Karp) on G'. If $|f| = \sum_k f_k$, a shipping plan exists such that each city c_k receives exactly f_k supplies.

Correctness:

A flow in G' is equivalent to a shipping plan subject to the constraints that (a) each ship can operate on certain days and can reach certain cities (b) on day d_i no more than v_i supplies are sent (c) for city c_k NO MORE THAN f_i supplies are received. Among these constraints, (a) is enforced by the structure of G', while (b) and (c) are enforced by the finite capacities of (s, d_i) and (c_k, t), respectively. If a flow exists such that $f(c_k, t) = c(c_k, t) = f_k$ for all c_k, each city c_k gets exactly f_k supplies. This requirement is equivalent to $|f| = \sum_k f_k$. Such a flow must be a maximum flow because it saturates the cut $(V'/\{t\})$-$\{t\}$. Therefore, to determine the existence of a shipping plan, where the constraint (c) is strengthened to "for city c_k EXACTLY f_i supplies are received", we can solve the maximum flow problem of G' and see if $|f| = \sum_k f_k$. If so, such a shipping plan exists and can be determined by f (see the following subpart). Otherwise, such a shipping plan does not exist. This proves the correctness of the algorithm.

Runtime:

$|V'| = O(|D| + |S| + |C|) = O(m + n)$. Notice that $|D| = 7$ and each s_j can only reach $|Y_j| \leqslant 5$ cities. Therefore, $|E'| = O(7|S| + 5|S| + |D| + |C|) = O(m + n)$. Construction of G' costs $O(V' + E') = O(m + n)$ time. Solving the maximum flow problem costs $O(V'E'^2) = O((m + n)^3)$ by Edmonds-Karp, or $O(V'^3) = O((m + n)^3)$ by relabel-to-front. Finally, checking if $|f| = \sum_k f_k$ costs $O(n)$. Therefore, the total runtime is limited by the maximum flow solver. Since $m < n$, the runtime is simplified as $T = O(n^3)$.

1.1.2 Subpart (ii)

Description:

The algorithm described above (named IfPlanExists) not only determines whether a valid shipping plan exists, but also returns the maximum flow f in the graph G'. Suppose a valid shipping plan exists. On

day d_i, ship s_j delivers in total $f(d_i, s_j)$ supplies. We determine this plan by specifying how the $f(d_i, s_j)$ supplies are distributed to c_k.

For a given s_j, construct a flow network $G_j = (V_j, E_j)$ as the following. Create 7 vertices corresponding to d_i, and $|Y_j| \leqslant 5$ vertices corresponding to all the cities c_{k_α} that are reachable from s_j, where $\alpha = 1, 2, \cdots, |Y_j|$. For every i and every α, create a directed edge (d_i, c_{k_α}) of capacity ∞. We also introduce a source s and a sink t. For every i, create a directed edge (s, d_i) whose capacity is $c^j(s, d_i) = f(d_i, s_j)$. For every α, create a directed edge (c_{k_α}, t) whose capacity is $c^j(c_{k_\alpha}, t) = f(s_j, c_{k_\alpha})$. Solve the maximum flow problem in this network. Let the resulting flow be f^j. Then $f^j(d_i, c_{k_\alpha})$ gives the amount of supplies delivered on day d_i by ship s_j to city c_{k_α}.

Iterating over all s_j produces a valid shipping plan.

Correctness:

For the flow f determined by IfPlanExists, $\sum_i f(d_i, s_j) = \sum_k f(s_j, c_k) = \sum_\alpha f(s_j, c_{k_\alpha})$ due to flux conservation at s_j.

In the flow network G_j corresponding to s_j, no capacity constraint is exerted on any (d_i, c_{k_α}). Therefore, the only cuts with finite capacities are $\{s\}$-$(V_j/\{s\})$ and $(V_j/\{t\})$-$\{t\}$. The former has capacity $\sum_i f(d_i, s_j)$ and the latter has capacity $\sum_\alpha f(s_j, c_{k_\alpha})$. These two values are equal and thus determine the minimal cut capacity, which is also the maximum flow. Therefore, all edges coming out of s and all edges going into t are saturated.

Because of this, $\sum_\alpha f^j(d_i, c_{k_\alpha}) = f^j(s, d_i) = c^j(s, d_i) = f(d_i, s_j)$ and $\sum_i f^j(d_i, c_{k_\alpha}) = f^j(c_{k_\alpha}, t) = c^j(c_{k_\alpha}, t) = f(s_j, c_{k_\alpha})$. We can then think of $f^j(d_i, c_{k_\alpha})$ as the flow in the original network going from d_i to c_{k_α} via s_j. So $f^j(d_i, c_{k_\alpha})$ gives the amount of supplies delivered on day d_i by ship s_j to city c_{k_α}. It follows that iterating over all s_j produces a valid shipping plan.

Runtime:

Suppose IfPlanExists returns a valid f. Notice that for every s_j, the flow network G_j is of constant size: at most $7 + 5 + 2 = 14$ vertices. Therefore, solving the maximum flow problem in G_j costs $O(1)$ time. Since we iterate over all s_j, the total runtime is $T = O(m)$.

We see that determining a shipping plan is of linear time once IfPlanExists is already executed. If not, or if somehow f is discarded, we have to run IfPlanExists first and altogher $O(n^3)$ time is required.

1.2 Part (b)

Description:

WLOG, let s_1 be the ship that cannot be operated. We know the previously valid maximum flow f_{old}. Now create a new flow in the network without s_1 as well as edges to and from s_1. Call this network G_{new}. Let $f_{new}(s, d_i) = f_{old}(s, d_i) - f_{old}(d_i, s_1)$ for all i and $f_{new}(c_k, t) = f_{old}(c_k, t) - f_{old}(s_1, c_k)$ for all k (it is sufficient to consider only c_k reachable from s_1). And $f_{new}(e) = f_{old}(e)$ if e goes to or from some s_i other than s_1.

It is proved in the correctness part that G_{new} is a flow network and f_{new} is a flow in G_{new}. But f_{new} may not be the maximum flow. So we augment it by Ford-Fulkerson method. If at the end of augmentation $|f| = \sum_k f_k$, then it is possible to make a shipping plan without s_1. Otherwise, such a plan is impossible.

Correctness:

The new network G_{new} is a flow network because we only delete s_1 as well as all (d_i, s_1) and (s_1, c_k). No backward edges are introduced.

f_{new} still satisfies flux conservation at all d_i, because $f_{new}(s, d_i) = f_{old}(s, d_i) - f_{old}(d_i, s_1) = \sum_j f_{old}(d_i, s_j) - f_{old}(d_i, s_1) = \sum_{j \neq 1} f_{old}(d_i, s_j) = \sum_{j \neq 1} f_{new}(d_i, s_j)$. Similarly, flux conservation holds at all c_k. Flux conservation at s_j other than s_1 is also maintained because all flows into and out of such an s_j are not changed. Finally, capacity constraints are satisfied because $f_{new}(e) \leqslant f_{old}(e)$ for all e. Moreover, for those $f_{new}(e)$ that are reduced, they are still non-negative. This is because $f_{new}(s, d_i) = \sum_{j \neq 1} f_{old}(d_i, s_j) \geqslant 0$, and the same is true for $f_{new}(c_k, t)$. In conclusion, f_{new} is a valid network flow on G_{new}.

Therefore, we can apply Ford-Fulkerson method to augment f_{new}. At the end we get a maximum flow f. If $|f| = \sum_k f_k$, then from the previous analysis, the demands of all cities can be satisfied. So a valid shipping

plan still exists. Otherwise, we cannot find a valid shipping plan.

Runtime:

To avoid confusion, I use capital K to replace the lowercase k in the problem set. Suppose s_1 delivered in total K supplies. Then the loss of $|f_{new}|$ is $\sum_i (f_{old}(s, d_i) - f_{new}(s, d_i)) = \sum_i f_{old}(d_i, s_1) = K$. Therefore, $|f_{new}| = \sum_k f_k - K$. By augmentation, we can at most improve the value of flow by K. Since each iteration of Ford-Fulkerson augments the flow by at least 1, $O(K)$ iterations are required. Each runs in $O(E) = O(m + n) = O(n)$ time. Therefore, the algorithm costs $T(K) = O(nK)$ time in all.

2 Career Fair

Description:

We create a bipartite graph $G = (V, E)$, where $V = L \cup R$. L is the set of student vertices and R the set of employer vertices. $|L| = n$ and $|R| = m$. For every $l_i \in L$ and every $r_j \in R$, create a directed edge (l_i, r_j). There are in total mn edges.

Construct a flow network by adding a source s and a sink t. For every $l_i \in L$, create a directed edge (s, l_i). For every $r_j \in R$, create a directed edge (r_j, t). Let V' and E' be the new sets of vertices and edges, and $G' = (V', E')$. Let the capacity of every edge in E' be 1.

Sort $t_{i,j}$ (break ties arbitrarily) and store them in an array T. Recursively reduce the size of T by bisection to get rid of large $t_{i,j}$ until a student does not have any employer to talk to. We need to keep tract of the "active subarray" of T that is being bisected. Call this subarray T'. In reality we just store the start point and the end point of T'. Initially, set $T' = T$. If $n > m$, raise an exception saying that some students are unable to talk to any employer. Otherwise, do the following recursion.

Bisect $T' = T'_1 \cup T'_2$, where T'_1 contains the smaller half of T'. For all the edges whose $t_{i,j}$ are stored in T'_2, set their capacities to 0, thus killing them in the flow network. Find a maximum bipartite matching of G by solving the maximum flow problem in G'. If $|M| = n$, let $T' = T'_1$ and recurse. If $|M| < n$, for all the edges whose $t_{i,j}$ are stored in T'_2, set their capacities back to 1. Let $T' = T'_2$ and recurse. $|M| > n$ is impossible since the cut $\{s\}$-$(V'/\{s\})$ has capacity n.

The recursion goes on until T' contains only one entry t_{i_0,j_0}. Run the maximum flow solver on G' (where the capacities of all edges beyond (i_0, j_0) are already set to 0). The maximum bipartite matching assigns students to employers such that each student has an employer to speak with and the last student's travel time is minimized to t_{i_0,j_0}.

Correctness:

Matching students with employers such that any two students do not go to the same employer and any student does not go to two different employers is a maximum-bipartite-matching problem between L and R. This can be solved as a network flow problem by adding a source and a sink and setting all edge capacities to be 1. The rule that each student must speak with exactly one employer further requires that every vertex in L is matched up with a vertex in R. Therefore, the maximum bipartite matching, i.e. the value of the maximum flow, must be exactly n. This can be achieved when $n \leqslant m$, which is checked at the beginning.

Suppose $n \leqslant m$ and let the minimized maximum of $t_{i,j}$ in a solution be $t^* = t_{i*,j*}$. The corresponding network flow forbids any edge (i, j) whose $t_{i,j} > t^*$. Moreover, if we further forbid (i^*, j^*) from carrying a flow, we do not get a solution with a smaller time because t^* is already optimal. This means that we either get the same t^* (which may happen if there are multiple edges with this time cost), or we cannot find any maximum bipartite matching such that $|M| = n$. In the first case, we can always keep on killing the most time-consuming edge until $|M| = n$ does not hold (which must happen no later than all edges of t^* being killed). Therefore, a valid strategy is to sort $t_{i,j}$, break ties arbitrarily, and find the frontmost one (call it t^{**}) beyond which all edges can be killed without affecting $|M| = n$. This is achieved by bisection in the algorithm.

In the bisection process it is guaranteed that $t^{**} \in T'$. This claim is proved by induction as the following. At the beginning the claim holds because $T' = T$. Suppose $t^{**} \in T'$ in the i-th recursion. In the $(i + 1)$-th recursion, if killing all edges in T'_2 preserves $|M| = n$, it must be that $t^{**} \in T'_1$. Otherwise $t^{**} \in T'_2$. By setting $T' = T'_1$ or T'_2 respectively, it is guaranteed that $t^{**} \in T'$ in this recursion. This means $t^{**} \in T'$ always holds. After finite recursions, T' will be reduced to one element t_{i_0,j_0}. Then $t^{**} = t_{i_0,j_0}$ is the minimized maximum

of travel time. The corresponding maximum bipartite matching M gives a valid assignment of students to employers.

Runtime:

$|V'| = O(m + n)$ and $|E'| = O(mn)$. Construction of G' costs $O(V' + E') = O(mn)$. $|T| = mn$, so bisection requires $O(\log(mn))$ recursions. In each recursion, reseting capacities costs $O(mn)$ time, and solving a maximum-bipartite matching problem by a network flow algorithm (e.g. Ford-Fulkerson) costs $O(V'E') = O(mn(m+n))$ time. Other work (e.g. comparisons) costs constant time. In conclusion, the total time cost is $T = O(mn(m+n)\log(mn))$. Notice that $n \leqslant m$ is required for a solution to exist. This simplifies the time cost to $T = O(m^2 n \log m)$.