COL 351

TUTORIAL SHEET 11

1. **[KT-Chapter7]** Give a polynomial time algorithm for the following minimization analogue of the max-flow problem. You are given a directed graph $G = (V, E)$, with a source s and sink t, and numbers (capacities) l_e for each edge $e \in E$. We define a flow f, and the value of a flow, as usual, requiring that all nodes except s and t satisfy flow conservation. However, the given numbers are lower bounds on edge flow, i.e., they require that $f_e \geq l_e$ for each edge e, and there is no upper bound on flow values on edges.

   (a) Give a polynomial time algorithm that finds a feasible flow of minimum possible value (Hint: Start with any flow from s to t which obeys the lower bounds, and try to send a maximum flow from t to s in a suitable modification of the graph G).

   (b) Prove an analogue of the max- flow min-cut theorem for this problem.

   Solution: For part (a), first send a flow from s to t such that all lower bounds are satisfied. We can easily do this as follows: for every edge $e = (u, v)$ find a path from s to t which goes through (u, v), and send l_e amount of flow on this path. Let f denote this flow. Now that we have a feasible flow, we will try to reduce the flow as much as possible, but we do not want to reduce the flow on an edge below l_e. Therefore, we reverse every edge, and make it's capacity equal to $f_e - l_e$. Now we send a max flow from t to s in this "reversed" graph.

   Can you guess the analogue of max-flow min-cut theorem? This will be the minimum over all cuts of the total lower bound on the edges leaving this cut. One can prove this by invoking max-flow min-cut theorem on the reverse graph.

2. **[KT-Chapter7]** You have a collection of n software applications, $\{1, \ldots, n\}$, running on an old system; and now you would like to port some of these to the new system. If you move application i to the new system, you expect a net (monetary) benefit of $b_i \geq 0$. The different software applications interact with one another; if applications i and j have extensive interaction, then the you will incur an expense if you move one of i or j to the new system but not both – let's denote this expense by $x_{ij} \geq 0$. So if the situation were really this simple, you would just port all n applications, achieving a total benefit of $\sum_i b_i$. Unfortunately, there's a problem. Due to small but fundamental incompatibilities between the two systems, there's no way to port application 1 to the new system; it will have to remain on the old system. Nevertheless, it might still pay off to port some of the other applications, accruing the associated benefit and incurring the expense of the interaction between applications on different systems. So this is the question: which of the remaining applications, if any, should be moved?

Give a polynomial-time algorithm to find a set $S \subseteq \{2, \ldots, n\}$ for which the sum of the benefits minus the expenses of moving the applications in S to the new system is maximized.

Solution: This can be solved by the min-cut algorithm. We first create an undirected graph as follows. It has a vertex for every software application – call these $v_1, \ldots, v_n$, where v_i corresponds to application i. We also have a special vertex t. If applications i and j have an associated value x_{ij}, then we have an edge between v_i and v_j of capacity x_{ij}. For every vertex v_i, $i \neq 1$, we have an edge (v_i, t) of capacity b_{v_i}. Now we argue that a v_1-t min-cut will give the desired solution. Indeed, let X be such a cut. So, $v_1 \in X, t \notin X$. What is the capacity of X? It is $\sum_{i,j:v_i \in X, v_j \notin X} x_{ij} - \sum_{i:v_i \notin X} b_{v_i} + \sum_v b_v$. Note that it is exactly the expense minus benefit of moving the applications which are not in the set X (plus a term which is fixed). Thus, finding a min-cut is same as finding the set of applications for which expense minus benefit is minimized, or benefit minus expense is maximized.

3. You are given a directed graph and special vertices s and t in the graph. Give an efficient algorithm to find the maximum number of vertex disjoint paths (i.e., no two paths should share a common vertex) from s to t. How will you solve this problem if the graph is undirected?

   Solution: Let G be the graph. We construct a new graph H from G as follows: for each vertex v of G, we add two new vertices v_1 and v_2 to H. Now all incoming edges into v now go into v_1 and all outgoing edges out of v in G now go out of v_2. Further, we add an edge from v_1 to v_2 in G. Now it is easy to check that a set of edge disjoint paths in H correspond to a set of vertex disjoint paths in H and vice versa.

   If the graph is undirected, we replace each edge by two parallel directed edges going in opposite directions and then use the algorithm for vertex disjoint paths in directed graphs. The main observation is that a set of vertex disjoint paths in the directed graph cannot use both copies of an original edge.

4. **[KT-Chapter7]** There is a set of k people $\{p_1, \ldots, p_k\}$ and for a sequence of N days, you are given a subset S_i of people who want to go work together in a car. You want to find a fair driving schedule. We say that the total driving obligation of a person p_j over a set of days is the expected number of times that p_j would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, the driving obligation for p_j can be written as

$$\Delta_j := \sum_{i:p_j \in S_i} \frac{1}{|S_i|}.$$

   Ideally, we would like to require that p_j drives at most Δ_j times; unfortunately, Δ_j may not be an integer. So lets say that a driving schedule is a choice of a driver for each day–that is, a sequence $p_{i_1}, \ldots, p_{i_n}$ with $p_{i_t} \in S_t$ for each day t - and that a fair driving schedule is one in which each p_j is chosen as the driver on at most $\lceil \Delta_j \rceil$ days. Prove that for any sequence of sets $S_1, \ldots, S_n$ there exists a fair driving schedule. Give an efficient algorithm to compute a fair driving schedule.

Solution: We first reduce this to an instance of the max-flow problem. Then we show that the desired flow exists but calculating the capacity of every cut. Construct a bipartite graph G with left hand side being people and right hand side being days. There is an edge between person p_j and day i if $p_j \in S_i$. All of these edges have infinite capacity. Now we add a source edge s and an edge from s to p_j for each j. The (s, p_j) edge has a capacity of $\lceil \Delta_j \rceil$. Similarly we have a vertex t and add an edge from each day i to t – each such edge has capacity 1. Now we wish to know if there is a flow of value N. This will happen iff the capacity of every cut is at least N. So take an s-t cut X – let P_X be the set of people in X and D_X denote the set of days in X. Because of the infinite capacity edges, we can assume that if $p_j \in P_X$, then all the days in which p_j travels are inside D_X. Now, the capacity of this cut is at least (by removing the ceiling on the numbers)

$$\sum_{p_j \notin P_X} \Delta_j + |D_X| = \sum_{j \notin P_X} \sum_{i : p_j \in S_i} \frac{1}{|S_i|} + |D_X| = \sum_i \frac{|S_i \cap (P - P_X)}{|S_i|} + |D_X|,$$

where P denotes the set of all people. Now notice that if a day i is outside D_X, then no member in S_i is in P_X (because of the infinite capacity edges). Therefore the RHS above is at least

$$\sum_{i \notin D_X} \frac{|S_i|}{|S_i|} + |D_X| = N - |D_X| + |D_X| = N.$$

Thus the capacity of every cut is at least N and hence, by max-flow min-cut theorem there is a flow of value N.

5. Prove the analogue of max-flow min-cut theorem in the circulation problem when we have lower and upper bounds on edge capacities and flow conservation needs to hold at every vertex.

   Solution: We showed that the circulation problem can be solved by reducing it to the problem where vertices have demands (which could be positive or negative) and edges have capacities only. We showed in class that for the latter (demand routing problem) there is a solution iff for every subset X of vertices, the total capacity of X is at least the demand outside X.

   Now let G be an instance of the circulation problem with l_e being the lower bound and u_e being the upper bound on the flow on an edge. We first sent l_e amount of flow along every edge and replaced all edge capacities by $u_e - l_e$ – let H denote this graph. We then showed that we need to solve the demand routing problem on H. Thus, G has a solution iff in H, for every subset X of vertices, the capacity of H is at least the demand outside X. What is the demand outside X? For this we only need to worry about the edges crossing X. For rest of the edges $e = (u, v)$, u gets positive demand and v gets negative demand, but the total demand outside X remains unchanged. Let E_1 denote the edges entering X and E_2 denote the edges leaving X. Then the capacity of X in H is equal to $\sum_{e \in E_1} (u_e - l_e)$. The demand outside X is given by $\sum_{e \in E_2} l_e - \sum_{e \in E_1} l_e$ – indeed, since we send l_e flow along each edge e, those leaving X carry some flow out of X resulting in a supply (and hence negative demand) outside X and similarly for E_2.

Now, we know that this problem has a solution iff for all X,

$$\sum_{e \in E_1} (u_e - l_e) \geq \sum_{e \in E_2} l_e - \sum_{e \in E_1} l_e.$$

Simplifying the above, this is same as

$$\sum_{e \in E_1} u_e \geq \sum_{e \in E_2} l_e.$$

This says that the circulation problem in G has a solution iff for every subset X of vertices in G, the total lower bound on the edges entering X is at most the total upper bound on the edges leaving X.