COL351: Analysis and Design of Algorithms

Tutorial Sheet - 10

November 7, 2022

Question 1 The basic rule for blood donation are: A patient of blood group A can receive only blood of group A or O. A patient of blood group B can receive only blood of group B or O. A patient of blood group O can receive only blood of group O. A patient of blood group AB can receive blood of any group.

Let s_O, s_A, s_B, s_{AB} denote the supply in whole units of the different blood types in a hospital for the coming week. Assume that the hospital knows the projected demand for each blood type d_O, d_A, d_B, and d_{AB} for the coming week. Give a max-flow based algorithm to check if the supply would meet the projected demand.

Solution Create a directed graph G with 4 layers as below:

1. First layer contains s
2. Layer two contains four vertices, namely, x_O, x_A, x_B, x_{AB}.
3. Layer three contains four vertices, namely, y_O, y_A, y_B, y_{AB}.
4. Last layer contains t.

The edge and their capacities are as follows:

1. For $i \in \{O,\ A,\ B,\ AB\}$, add edge (s, x_i) to G of capacity c_i.
2. For $j \in \{O,\ A,\ B,\ AB\}$, add edge (y_j, t) to G of capacity d_j.
3. Finally, add edge (x_i, y_j) to G, for $i, j \in \{O,\ A,\ B,\ AB\}$ iff a person of blood group j can receive blood from a person of blood group i. The edges (x_i, y_j) have infinite capacity.

Claim: The projected demand can be met iff the (s,t)-max-flow in G is $(d_O + d_A + d_B + d_{AB})$.

Question 2 Let $G = (V, E)$ be a directed graph, and (s, t) be a vertex pair. Two paths from s to t are said to be *internally-vertex-disjoint* if they do not share any vertex except end-points s and t. Present an $O(mn)$ algorithm to compute the maximum number of vertex disjoint paths from s to t. *Hint:* An (s, t)-max-flow on edges of unit capacity can be computed in $O(mn)$ time.

Solution Let G be the input graph on n vertices and m edges. Compute a new graph H from G by splitting each $v \neq s, t$ into an edge $(v_{in},\ v_{out})$ such that:

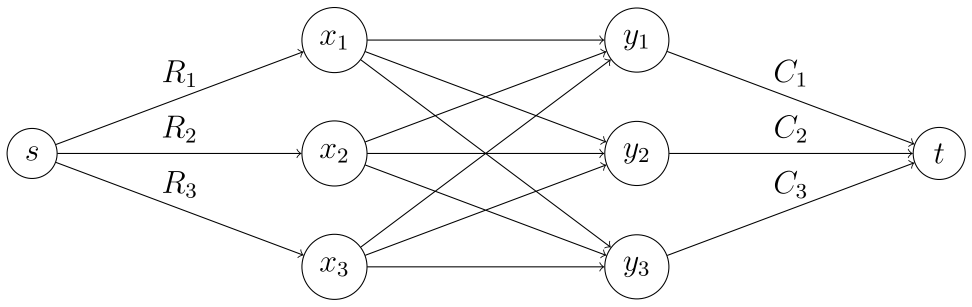
- The incoming edges of v_{in} in H corresponds to incoming edges of v in G.
- The outgoing edges of v_{out} in H corresponds to outgoing edges of v in G.

Note that $|V(H)| \leqslant 2n$ and $|E(H)| \leqslant m + n$. Compute $\alpha =$ the value of maximum flow from s to t in H assuming all edge capacities are one. This takes $O(mn)$ time.

The value α is equal to the maximum number of vertex disjoint paths from s to t in G as there is 1-1 correspondence between ‘edge-disjoint paths in H’ and ‘vertex-disjoint paths in G’.

Question 3 Let $M = (m_{ij})$ be a square matrix of size n storing positive real numbers. It is given that the sum of elements of each column as well as each row is a positive integer. Prove that elements of M can be replaced by integers without changing any column sum or row sum.

Solution Compute a directed graph $G = (V, E, c)$ on $2n + 2$ vertices as follows.



The edge capacities are as follows.

- For $i = 1$ to n: capacity of $(s, x_i) = R_i$ (sum of i^{th} row in M).
- For $j = 1$ to n: capacity of $(y_j, t) = C_j$ (sum of j^{th} column in M).
- For $1 \leqslant i, j \leqslant n$: capacity of $(x_i, y_j) = \infty$.

Compute maximum flow from s to t using Ford-Fulkerson algorithm. This will guarantee an integer flow along each edge.

Observe that the value of maximum flow will be $S := \sum_{i=1}^{n} R_i = \sum_{j=1}^{n} C_j$ (Why?). Furthermore, with respect to any maximum-flow out-edges of s, and in-edges of t will be fully saturated.

We can now compute new matrix M' by setting its $(i, j)^{th}$ entry as flow through (x_i, y_j) edge. It is easy to observe that the row-sums and column-sum remain same as that in the input matrix.

Question 4 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 a new system. If you move application i to the new system, you expect a net (monetary) benefit of $b_i \geqslant 0$. 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 $x_{ij} \geqslant 0$. So if the situation were really this simple, you would just port all n applications, achieving a total benefit of $\sum_{i=1}^{n} b_i$. 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.

Your task is the following: which of the remaining applications, if any, should be moved? Design an 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 Create an undirected graph $G = (V, E, c)$ on $n + 1$ vertices as follows. The vertex-set V is $\{s = v_1, v_2, \ldots, v_n, t\}$, where v_i corresponds to application i.

For $1 \leqslant i, j \leqslant n$, if applications i and j interact, then we have an edge between v_i and v_j of capacity x_{ij}. Further, for $1 \leqslant i \leqslant n$, we have an edge (v_i, t) of capacity b_i.

Claim: A (s, t)-min-cut will give the desired solution.[1]

Proof: Let (X, Y) be an (s, t)-cut. So, $v_1 \in X, t \in Y$. Here Y corresponds to the set of applications that we move. The capacity of this cut is

$$\begin{aligned} c(X, Y) &= \sum_{i,j: v_i \in X, v_j \in Y} x_{ij} + \sum_{i: v_i \in X} b_i \\ &= \Big(\sum_{i,j: v_i \in X, v_j \in Y} x_{ij} - \sum_{i: v_i \in Y} b_i \Big) + \sum_{i} b_i \end{aligned}$$

Note that $\big(\sum_{i,j: v_i \in X, v_j \in Y} x_{ij} - \sum_{i: v_i \in Y} b_i \big)$ is exactly the expense minus benefit of moving the applications. 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.

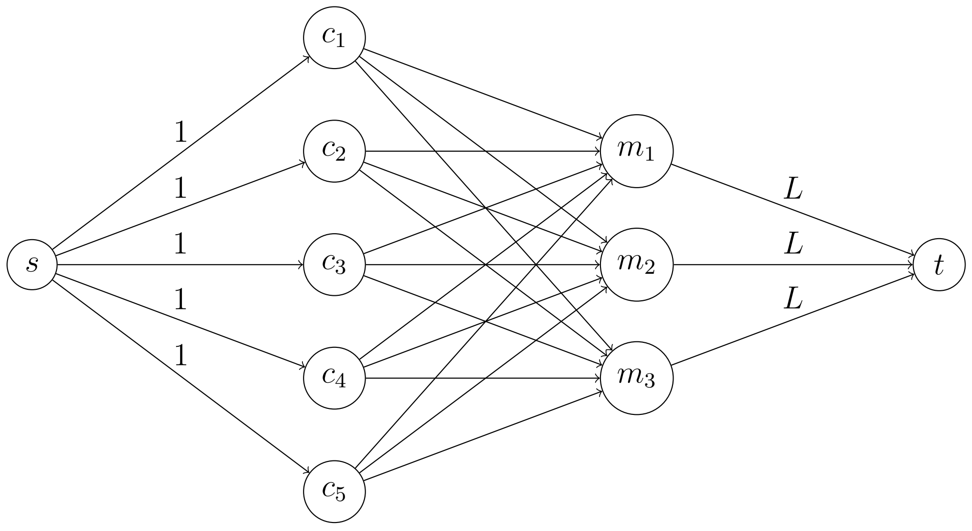
[1]Here $s = v_1$.

Question 5 There are n clients $(c_1, \dots, c_n)$ who want to be connected to one of the k mobile towers $(m_1, \dots, m_k)$ in a town. You are given the (x,y) coordinates of each client and each tower, a distance parameter d, and a load parameter L. Design a polynomial time algorithm to decide if every client can be connected simultaneously to some mobile tower subject to the following constraints.

1. Each client is connected with exactly one of the mobile towers, and a client can only be connected to tower that is within distance d.

2. No more than L clients can be connected to any single mobile tower.

Solution Compute a directed graph $G = (V, E, c)$ on $n + k + 2$ vertices as follows.



The edge capacities are as follows.

- For $i = 1$ to n: capacity of $(s, c_i) = 1$.
- For $j = 1$ to k: capacity of $(m_j, t) = L$.
- For $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant k$: capacity of $(x_i, y_j) = \begin{cases} 1 & \text{if } dist(c_i, m_j) \leqslant d; \\ 0 & \text{otherwise.} \end{cases}$.

<u>*Claim:*</u> It is possible to obtain a valid client-tower connection if and only if the value of (s, t)-max-flow in the above graph is exactly n.

The corresponding connections will be obtained by an (s, t)-max-flow f. We connect c_i to m_j if and only if $f(c_i, m_j) = 1$.