Homework 3

Prateep Mukherjee

November 12, 2013

1.
a) A "winning order" can be generated for a finite set of residents given the problem statement, can be proved through *principle of induction.*

Let, n denote the number of Pongville residents.

Base case: If $n = 2$, the "winning order" is the relative ordering of the two players. So, if 1^{st} player defeats 2^{nd}, then the order is $(1, 2)$, else the order is $(2, 1)$. If $n = 3$, there are two possible scenarios. The two cases are shown in Fig. 1. (i) **Without cycle:** The order can be written as $(3, 1, 2)$, (ii) **With cycle:** The order can be written as $(3, 1, 2, 3)$.

Inductive case: Assume n residents of Pongville has been arranged in the "winning order". Next we add another resident, say A_{n+1} to the graph. Finally, we need to prove that the sequence generated from $n+1$ residents is also a "winning order". There can be 3 different positions for A_{n+1}.

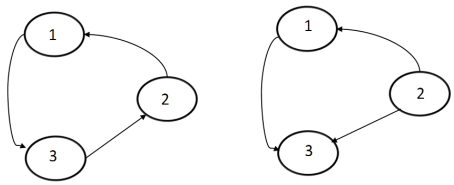

Figure 1: The two scenarios. **Left** with cycle; **Right** without cycle.

i A_{n+1} defeats everyone else. So, we can add A_{n+1} to the end of the list.

ii A_{n+1} loses to everyone else. So, we can add A_{n+1} to the beginning of the list.

iii A_{n+1} wins some and loses some. In this case, we find a position, say i, in the existing order, such that player at $i-1$ loses to A_{n+1} while that at $i+1$ wins the match against A_{n+1}.

This completes the proof that a "winning order" can be generated given the graph.

b) The algorithm to compute a "winning order" is designed as follows:

Step 1:	Compute the in-degree and out-degree of every vertex in V. Choose the vertex(u) with maximum out-degree. Initialize $order = \phi$
Step 2:	For each node $v \in N(u)$, if v has not been visited, do one of the following. If: there exists edge $u \rightarrow v$ in G, append v to the beginning of the list. Else: traverse the rest of *order* to find an appropriate(as discussed in (a)) place for inserting v. Goto step 1 and reiterate with v.
Step 3:	If no such v exists, *terminate*, as we have reached the end of the sequence.

Proof of correctness:. The graph G is a strongly connected graph. So, there is an edge between every pair of nodes. Therefore, one iteration over the set of nodes visits every node in the graph. Hence, our algorithm is guaranteed to generate an optimal "winning order". The algorithm has two nested loops, one for iterating over each vertex, and another for finding an appropriate position for it. Thus, the overall algorithm is $\mathcal{O}(V^2)$. □

2.
a) The given algorithm does not always compute a maximum flow. The latter can be proved if we can show that the algorithm fails to compute the maximum flow for any given graph(Proof by exception).

Proof. An example is shown in Fig. 2. The maximum feasible flow computed using this algorithm on G is 1.

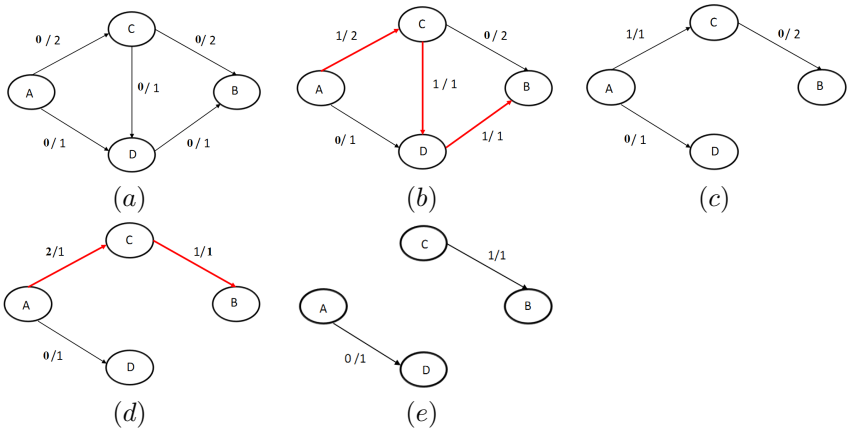

Figure 2: One iteration of the algorithm is shown. Node A is the source and B is the target. **(a)** A graph $G = (V, E), V = \{A, B, C, D\}$ is shown with each edge e labeled with flow(f) / maximum capacity(c); **(b)** A path π is chosen(edges marked in red); **(c)** The residual graph shown after removing the edges; **(d)** Path π is shown in red. Removing this path disconnects the graph **(e)** , therefore the algorithm terminates.

However, the maximum feasible flow in this graph is 3(2(AC), 0(CD), 1(AD), 2(CB), 1(DB)). Hence proved. □

b) The maximum flow along G can be computed using any maxflow algorithm, like *Ford-Fulkerson*. We can show that if the oracle picks a path from the set of augmenting paths used in the residual graph G_f of the Ford-Fulkerson algorithm, at each iteration of the given algorithm, we can compute the optimum maximum flow.

Proof. Let, M be the set of augmenting paths from source to target. The max-flow algorithm saturates an edge in each augmenting path in the residual graph G_f. So, if the oracle choses a path $p \in M$, saturating p and removing the

saturated edge ensures a maximum flow along that path. Therefore, choosing the path π from M always guarantees an optimum maximum flow. Hence proved. □

3. The problem statement can be restated as follows: Given a graph $G = (V, E)$, find the minimum number of vertices removing which disconnects the graph. This indicates a minimum cut solution to determine the minimum number of buildings in order to disconnect WEB & ML. However, there are two requirements that are needed in order to apply a standard max-flow min-cut algorithm.

1. The graph G must be directed acyclic.
2. The min-cut algorithm will output the minimum number of edges to disconnect the graph, and not the minimum number of vertices.

Note that the given graph in the problem is undirected. So, first we need to transform the undirected graph(G) to a directed acyclic one($\tilde{G} = (\tilde{V}, \tilde{E})$). An easy solution to this is an undirected edge $(u-v)$ can be replaced by two directed edges $u \to v$ and $v \to u$.

The next step is to transform the goal from finding the minimum number of vertices to finding the minimum number of edges, so that a standard max-flow min-cut algorithm can be applied. The solution here is split every vertex into two and introduce a "fake" directed edge($\tilde{e}$) between them, i.e. $\{\tilde{e} | \tilde{e} \in \tilde{E} \ \& \ \tilde{e} \notin E\}$. So, now the problem to finding the minimum set of vertices reduces to finding the minimum number of "fake" edges to disconnect the graph.

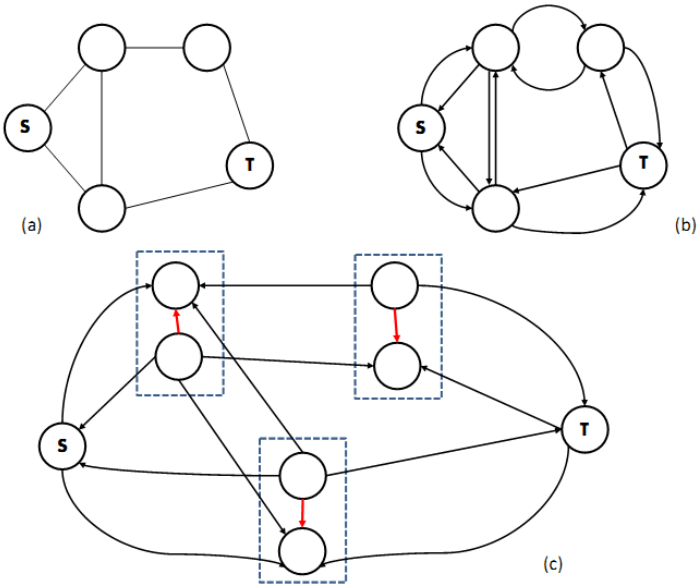

Figure 3: **(a)** original graph($G = (V, E)$); **(b)** the undirected graph converted to directed by introducing two directed edges for each edge; **(c)** augmented directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where $|\tilde{V}| = 2 \times |V|$ and $|\tilde{E}| = 2 \times |E| + |V|$. **S** represents WEB and **T** represents ML. "Fake" edges are denoted in red.

A transformation on a simple graph is shown in Fig. 3. Each vertex(v) is split into two vertices, namely *in-vertex* and *out-vertex*. All the incoming edges onto v are directed to *in-vertex*, while all the outgoing edges from v are directed to *out-vertex*. The "fake edge" for v is directed from *out-vertex* to *in-vertex*, as shown in Fig. 3. To ensure that only "fake" edges are in the min-cuts, we make the capacities of "real" edges to be ∞, and that of "fake" edges to be 1. Now, we apply the Dinits algorithm(Short-pipes) to find the minimum cut. The capacities ensure that only the fake edges are cut. The number of edges in the minimum cut is equal to the minimum number of nodes to be disconnected. The time complexity of the algorithm is

$$\begin{aligned}\mathcal{O}(\tilde{V}\tilde{E}^2) &= \mathcal{O}((2\times V).(2\times E+V)^2)\\ &= \mathcal{O}(8\times VE^2+8\times EV^2+2\times V^3)\\ &= \mathcal{O}(V^3+VE^2).\end{aligned}$$

Proof of correctness:. The graph $\tilde{G}$ is directed acyclic. Therefore, max-flow/min-cut algorithm successfully finds a minimum cut. Also, since capacities of real edges are ∞, they will never be saturated and thus not included in the minimum cut. Disconnecting the "fake" edges means removing the vertices. Therefore, the solution to the minimum cut problem is the optimum solution for our problem. Hence proved. □

4. The given problem is modeled as a max-flow algorithm. The maxflow graph $G=(V,E)$ can be designed as shown in Fig 4.
Now, the total number of countries is assumed to be $4k$. Therefore, the total flow that crosses barrier **B** is at most $4k$. According to the *conservation of flow* theorem, the total flow across barrier **A** is at most $4k$ too. The net outflow from the source S to the first layer, therefore, is $4k$. Also according to the question, the committee will have equal number of freshmen, sophomore, junior and seniors. Hence, the capacity of each edge $(S \to y_i)$ is k, where y_i denotes a node in the first layer. The year-student and student-country edges are all of same capacity, as they are all of same priority.
Each country can be represented by at most one student in the committee. Thus, the capacities of the edges in the final layer are also constant. Therefore, if $f(e)$ and $c(e)$ represent the flow and capacities of edge $e \in E$ respectively, then the above problem can be mathematically formulated as follows:

$$maximize \quad f \tag{1}$$

$$\begin{aligned} s.t. \qquad f(e) &\to feasible\ flow, \quad \forall e \in E,\\ c(S,y_i) &= k, \quad \forall y_i \in \{year\}\\ c(y_i,s_j) &= 1, \quad \forall y_i \in \{year\}, s_j \in \{students\}\\ c(s_i,c_j) &= 1, \quad \forall s_i \in \{students\}, c_j \in \{country\}\\ c(c_j,T) &= 1, \quad \forall c_j \in \{country\}\end{aligned}$$

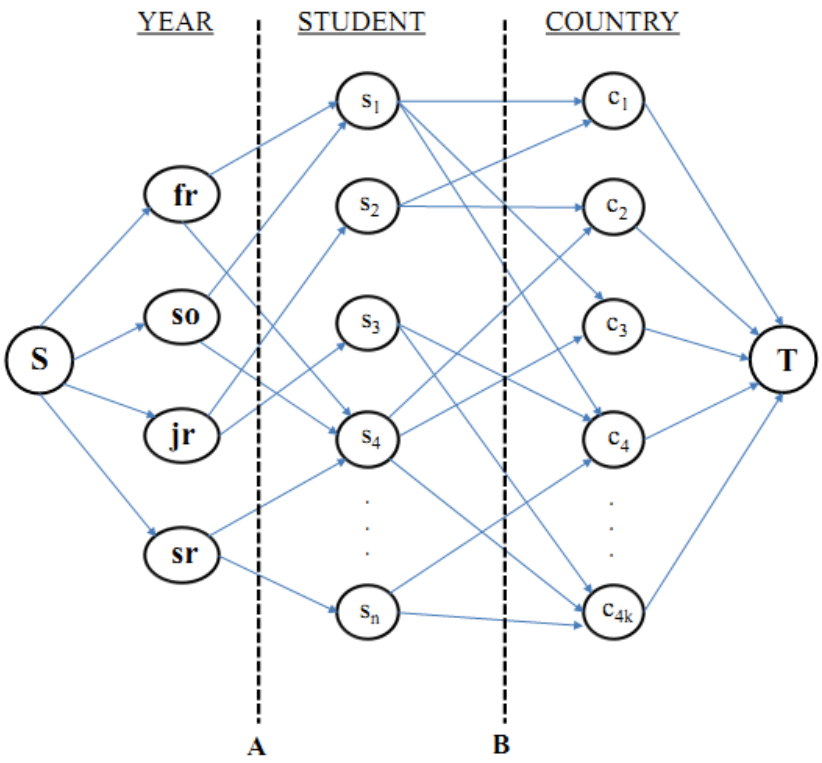

Figure 4: The maxflow graph G is shown. S is the source, $\{fr, so, jr, sr\}$ denotes the different years, s_i are the students, c_j are the countries, and T is the sink/target. An edge (s_i, c_j) indicates that student s_i is affiliated with country c_j. The dotted lines **A** and **B** show the barriers across which flow is transfered.

Eq. 1 can be solved using the standard algorithms for *max-flow*. For this example, we can use *Ford-Fulkerson* algorithm to obtain the maximum flow in $\mathcal{O}(E\mathbf{f})$, where $\mathbf{f}$ is the maximum flow.

Proof of correctness:. The capacities across the edges ensure that the maximum flow will guarantee optimal assignment of students to countries. However, **one might be able to generate an example graph which has no possible solution.** For example, a country is affiliated by only one student while the latter has more than one affiliations, and the max-flow algorithm saturates an edge connecting the student with some other country. In this case, the said country won't have a representative. Hence, the max-flow algorithm does not always generate a possible solution. □

5. The goal of this problem is to find the shortest path between *every* pair of vertices in a undirected weighted graph $G = (V, E, w), w : E \mapsto \Re$, such that the weight of the edges in that path is maximized. First, we define the value of

a path between a pair of nodes. Let, $P = e_1 e_2 \cdots e_k$ be a path between nodes i & j ($i, j \in V$, $e_i \in E$). Value of the path P, denoted as V(P), is defined as :

$$V(P) = \max_{i=[1,k], e_i \in P} w_{e_i}$$

Thus, our problem is equivalent to find, for every pair of nodes i and j, a minimum value path from node i to j. Let, T be the *minimum spanning tree*(MST) on G.

Claim: The unique path between nodes i and j in T is the minimax path between them in G.

Proof: Let P be the unique path between i and j in T. So, if (v_k, v_l) is the maximum weight edge in the path P, removing (v_k, v_l) from T creates two partitions S and S', such that $i \in S$ and $j \in S'$. This defines a cut (S, S').

According to the *cut property*[1] of a MST, for any edge (i', j') such that $i' \in S$ and $j' \in S'$, $w_{v_k, v_l} \leq w_{v'_i, v'_j}$.

Since any path P' between i and j must contain all the nodes from the cut (S, S'), w_{v_k, v_l} is the value of the path P, P must be the minimum value path. This establishes that the unique path in MST T is the minimax path between each pair of nodes in G. Extending this argument for all pairs of nodes in G completes our proof. □

The algorithm to compute the minimax path of G hence consists of two steps: (i) Compute the MST T in G, (ii) return the unique path between every pair of nodes in T. The first step takes $\mathcal{O}(|E| \log |V|) = \mathcal{O}(|E| \log |E|)$ using Kruskal's algorithm. The second step is $\mathcal{O}(|E|)$ worst case. Thus, the total time complexity of our algorithm is $\mathcal{O}(|E| \log |E|)$.

[1] Cut property: `http://en.wikipedia.org/wiki/Minimum_spanning_tree#Cut_property`