HW08

Jiaxin Li, Sunjun Gu

November 8, 2020

1 Question 1

1.1 a:

Input: n,m,k,and S_i for i-1,..,m. Explanation: n topics; m friends; k questions about each of the n topic; a set S_i of topics i-th friend is capable of asking about,i = 1,2,...,m.Each friend asks at most three questions each.(Constraint:$k \leq m; 3m \geq kn$)
Output: Binary "yes" if there exists a solution to ask k questions about each of the n topics, while each friend asks at most three questions each. Otherwise, binary "no".

Algorithm:

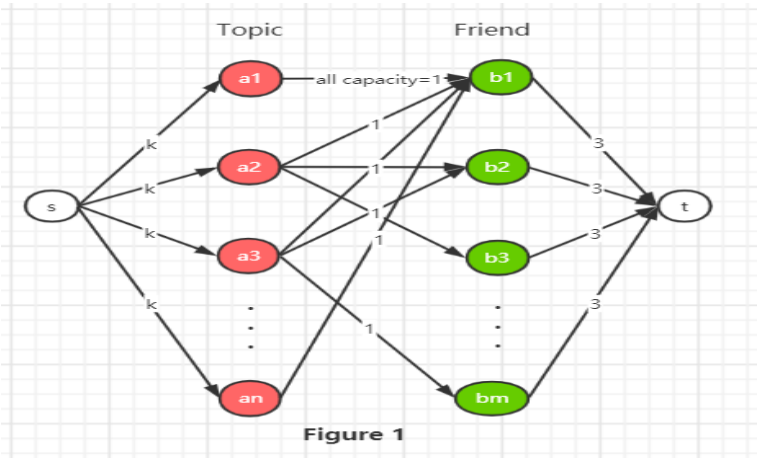

Figure 1: Network flow G for 1a
On G, vertex s is the source, vertex t is the sink. vertex a_i for i = 1, ..., n is the i-th topic, vertex b_j for j=1, ..., m is the j-th friend.
Set one edge from s$\rightarrow a_i$ for each a_i with capacity k
Set exactly one edge with capacity 1 between a_i and b_j, if $a_i \in S_j$.
Set one edge from $b_j \rightarrow t$ with capacity 3

1. Build network G with vertices and edges like the Figure 1.
2. Run the MaxFlow algorithm on G getting the max-flow f and its value v(f).
3. If $v(f) < kn$,return binary "No".Else $v(f) = kn$,return binary "Yes"

Sanity Check: The network G meets all restrictions on the problem.
1. Each friend can ask about a specific topic at most once. (By capacity constraints, $f(e) \leq c(e) = 1$, where $e \in E : e = (a_i, b_j)$.
2. In this network, each topic will be asked about at most k times (By conservation and capacity constraints, $f_{out}(a_i) = f_{in}(a_i) \leq c(S, a_i) = k$)
3. Each friend can only ask up to 3 total questions. (By capacity constraints,)
Claim:
The value of a maximum flow in G is kn if and only if there is a valid way to ask k questions about each of the n topics, with each friend asking at most 3 questions (each on a different topic).
Proof of Correctness:
By the claim, we need to prove two implications
(1).Maximum flow in G is kn ⇒ Exists a valid assignment of questions.
(2).Exists a valid assignment of questions⇒ Maximum flow in G is kn

(1)Let f be the max-flow, consider the max-flow f's value v(f) = kn. Since edges from s to each topic a_i for i=1,..,n have capacity k, this means that all the edges from vertex s to a_i for i=1,..,n are used at full capacity k on f.Thus, we ask k questions about each topic for total n topics(Sanity Check 2). Since the edges coming into friend b_i for i = 1,...,m have unit capacity and from the topic vertex only once,and the topics belong to S_j, therefore each friend can only ask about a given topic once(Sanity Check 1).As the edge from each friend vertex to vertex t all have capacity 3, by the capacity constraint, flow on these edge can not exceed 3. Therefore, each friend asks at most 3 questions (Sanity Check 3). The assignment solution that friend b_j ask one question on a_i topic and totally no more than 3 questions given by edge $a_i \to b_j$ on the max-flow f exists and is valid.

(2)For each pair(i,j) consisting of a friend b_j asking about a topic a_i, route one unit of flow along the path s→ $a_i \to b_j \to$ t(a friend asks one question on one topic). The conservation constraints and integrality are met as we are considering a superposition of unit flows along st-paths. The capacity constraint of k on the edge from s to a_i is met as we ask k questions about each of the n topics. The capacity of 1 on the edge from a_i to b_j is met as friend b_j only ask one questions for topics on S_j. The capacity of 3 on the edge from b_j to t is met as each friend could ask at most 3 questions. The value of the flow equals $\sum_{i=1}^{k} \sum_{j=1}^{n} 1$=kn as that is the sum of the flow coming into t. For s and t constraint, v(f)≤kn,v(f)≤3m,3m≥ kn. Therefore, kn is also the max-flow value.

Combine the two claim proof above, we show the correctness of the claim also the correctness of our algorithm.

Proof of Time Complexity :
For algorithm step 1, we are setting s,t,a_i for i=1,...,n,b_j for j=1,..,m vertices, total O(1+n+m+1)=O(n+m); Besides, there are edges s→ a_i for i=1,...,n, total n edges;$b_j \to t$ for j=1,..,m, total m edges; $a_i \to b_j$ for j=1,..,m where $a_i \in S_j$, at most nm edges. Therefore, setting edges costs O(n+m+nm) = O(nm) edges. For step 2, we run the MaxFlow algorithm on G whose runtime is O(number(vertex)×number(edge)) by lecture,therefore, runtime here is O(nm)×O(n+m)=O(n^2m+nm^2).For step 3, it does comparison and return value with runtime O(1).Add these steps together, runtime for this algorithm is O(n^2m+nm^2). Thus, it is a polynomial-time algorithm.

1.2 b:

Input: n,m,k,and S_i for i-1,..,m,each friend is Optimist or Pessimist. Explanation: n topics; m friends; k questions about each of the n topic; a set S_i of topics i-th friend is capable of asking about,i = 1,2,...,m.

Output: Binary "yes" if there exists a solution to ask k questions about each of the n topics, while each friend asks at most three questions each Each friend asks at most three questions each and there is no topic for which all k questions come from people with the same attitude. Otherwise, binary "no".

Algorithm: We do modification on Algorithm 1a step 1, build a different network G'. On G', we add vertex O and P after each topic vertex with edge capacity k-1 and before each friend matching their attitude with capacity 1.The other vertex/edge/capacity keep the same as 1a. The detailed algorithm is as the following figure 2 and description.

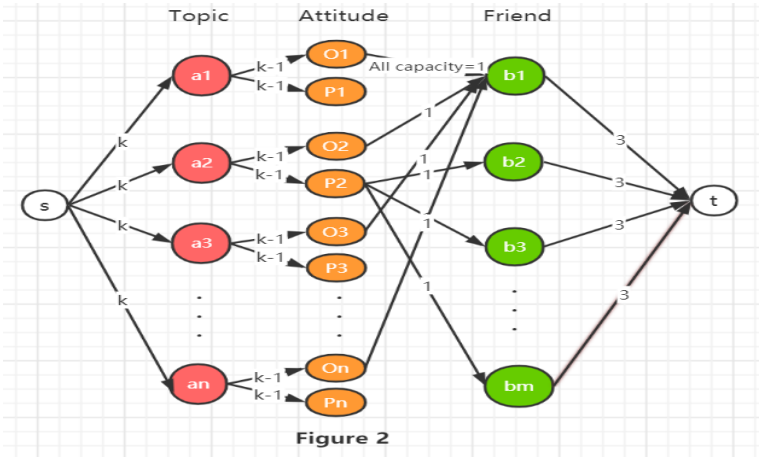

Figure 2: Network flow G' for 1b
On G', vertex s is the source, vertex t is the sink. vertex a_i for i = 1, ..., n is the i-th topic, vertex b_j for j=1, ..., m is the j-th friend. vertex O,P stand for Optimist or a Pessimist attitude.
Edges s$\rightarrow$ a_i has capacity k,Edges $b_j \rightarrow t$ has capacity 3,Edges $a_i \rightarrow O_i$,$a_i \rightarrow P_i$ has capacity 1, Edges O_i or P_i matching b_j attitude with capacity 1 and $a_i \in S_j$,and only one edge between $O_i/P_i, b_j$(There is a path among a_i,O_i,b_j if j-th friend is Optimist and are capable of asking about the i-th topic; a path among a_i,OP_i,b_j if j-th friend is Pessimist and are capable of asking about the i-th topic)

1. Build network G' with vertices and edges like the Figure 2.
2. Run the MaxFlow algorithm on G getting the max-flow f and its value v(f).
3. If $v(f) < kn$,return binary "No".Else $v(f) = kn$,return binary "Yes"

Sanity Check: The network G' meets all restrictions on the problem.
1. Each friend can only ask about a given topic once
2. We need to ask k questions about each topic
3. Each friend can only ask up to 3 total questions
4.There is no topic for which all k questions come from people with the same attitude.
Claim:
The value of a maximum flow in G' is kn if and only if there is a valid way to ask k questions about each of the n topics come from people with the same attitude, with each friend asking at most 3 questions (each on a different topic).
Proof of Correctness:

By the claim, we need to prove two implications
(1).Maximum flow in G is kn ⇒ Exists a valid assignment of questions.
(2).Exists a valid assignment of questions⇒ Maximum flow in G is kn

(1)Consider the max-flow f's value v(f) = kn. Since edges from s to each topic a_i for i=1,..,n have capacity k, this means that all the edges from vertex s to a_i for i=1,..,n are used at full capacity k on f.Thus, we ask k questions about each topic for total n topics(Sanity Check 2). Edges from a_i to O_i, P_i have capacity k-1, this means neither optimist or pessimist can ask all the k question for a topic, one of them could ask at most k-1 questions for a topic a_i(Sanity Check 4).Since the edges from O or P coming into friend b_i for i = 1,...,m have unit capacity and from the topic vertex only once,and the topics belong to S_j, therefore each friend must be optimist or pessimist and can only ask about a given topic once(Sanity Check 1).As the edge from each friend sink into vertex t all have capacity 3, by the capacity constraint, flow on these edge can not exceed 3. Therefore, each friend asks at most 3 questions (Sanity Check 3). The assignment solution that friend b_j(optimist or pessimist) ask one question on a_i topic and totally no more than 3 questions given by edge $a_i \to b_j$ on the max-flow f exists and is valid.

(2)For each pair(i,j) consisting of a friend b_j(optimist or pessimist) asking about a topic a_i, route one unit of flow along the path s→ $a_i \to O_i \to b_j \to$ t or s→ $a_i \to P_i \to b_j \to$ t (a optimist or pessimist friend asks one question on one topic). The conservation constraints and integrality are met as we are considering a superposition of unit flows along st-paths. The capacity constraint of k on the edge from s to a_i is met as we ask k questions about each of the n topics.The capacity of k-1 on edge from a_i to O_i or P_i is met as no topic for which all k questions come from people with the same attitude. The capacity of 1 on the edge from O_i or P_i to b_j is met as friend b_j must be optimist or pessimist and only ask one questions for topics on S_j. The capacity of 3 on the edge from b_j to t is met as each friend could ask at most 3 questions. The value of the flow equals $\sum_{i=1}^{k}\sum_{j=1}^{n} 1 =$ kn as that is the sum of the flow coming into t. For s and t constraint, v(f)≤kn,v(f)≤3m,3m≥ kn. Therefore, kn is also the max-flow value.

Combine the two claim proof above, we show the correctness of the claim also the correctness of our algorithm.

Proof of Time Complexity :
Similar with 1a, for algorithm 1b step 1, there are total O(1+n+2n+m+1)=O(n+m) vertices where 2n is for attitude vertices; Besides, there are at most total O(n+2n+m+nm) = O(nm) edges where 2n is for topic to attitude edges, attitude to friend edges are at most O(nm) for each friend has at most one question on at most each question. For step 2, we run the MaxFlow algorithm on G whose runtime is O(number(vertex)×number(edge)) by lecture,therefore, runtime here is O(nm)×O(n+m)=O(n^2m+nm^2). For step 3, it does comparison and return value with runtime O(1).Add these steps together, runtime for this algorithm is O(n^2m+nm^2). Thus, it is a polynomial-time algorithm.

2 Question 3

2.1 a:

Input: n,a_i,b_i for i = 1,...,n, c(i,j) for i,j = 1,...,n,i≠j and c(i,j)=c(j,i). Explanation: n projects; a_i A company(Abed) charges a_i to do job i; b_i B company(Britta) charges b_i to do job i; $c(i,j)$ incompatibility cost to get jobs i and j done by different companies. For convenience to read, we

only write c(i,j) for all i, j and i<j
Output: Projects assignment for A and B landscapers so as to minimize the sum of the purchase costs and incompatibility costs.

Algorithm:

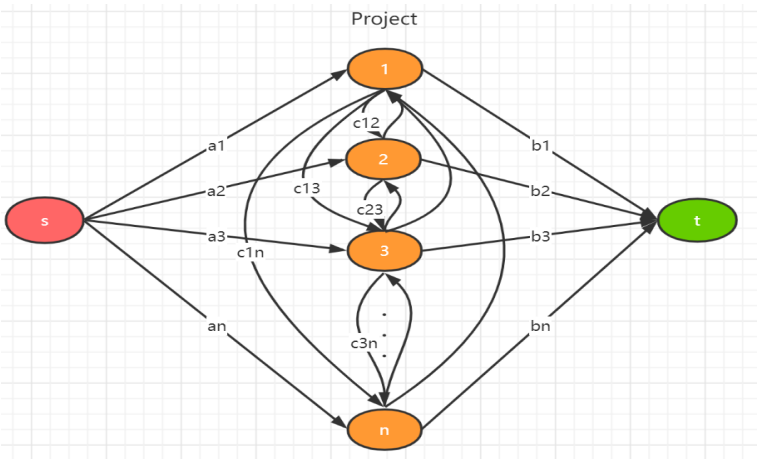

Figure 3: Network flow N for 3a
On N, vertex s is the source representing for B company, vertex t is the sink representing company A. vertex i for i = 1, ..., n is the i-th project.
Edges A$\rightarrow$ i has capacity a_i,Edges $i \rightarrow B$ has capacity b_i for i=1,...,n,Edge $i \rightarrow j$ has the same capacity c(i,j) as edge $j \rightarrow i$

1. Build network N with vertices and edges like the Figure 3.
2. Construct min-cut (S,T) in N:
   Run the MaxFlow algorithm on N getting the max-flow f and its value v(f).
   Build residual graph N_f using N and f.
   Find min st-cut (S,T) using N_f and BFS.
3. According to min-cut(S,T), assign all projects in S(exclude source vertex) to B company, assign all projects in T(exclude sink vertex) to A company.

Sanity Check: The network N meets all restrictions on the problem.
1. Every job should be done exactly once
2. We need to account for the incompatibilities between the styles of the different suppliers
3. Abed charges a_i to do job i, and Britta charges b_i for the same job.
4.Minimize the sum of purchase costs and incompatibility costs
Proof of Correctness:
We want to Minimize the sum of purchase costs and incompatibility costs, write the formula:

$$\min(\sum_{i \in A} a_i + \sum_{j \in B, B = I \setminus A} b_j + \sum_{i \in A, j \in B} c(i,j)) \tag{1}$$

. By algorithm, we get the min st-cut (S,T) and its value c(S,T). We want to show that the value of equation(1) is exactly the capacity of the min-cut in the network we build. Projects j contained on S are assigned to B company, projects i contained on T are assigned to A company, each vertex could only be on one side in the min-cut(Sanity Check1). Since c(S,T) sum up all S$\rightarrow$T edges' capacity,

saying project j is contained on S, project i is contained on T. S→T edges include edges from j to B and edges from A to i. Sum up the capacity of edges from A to i equals to $\sum_{i\in A} a_i$, sum up the the capacity of edges from j to B equals to $\sum_{j\in B, B=I\setminus A} b_j$(Sanity Check3). They meet the first and second term on equation(1). S→T edges also include edges from j in S to i in T, these edges' capacity are the incompatibility costs between i and j. Since we draw arrows in different directions on the i, j, c(i,j)=c(j,i), it only add one incompatibility cost. Sum up the capacity of edges from j to i equals to $\sum_{i\in A, j\in B} c(i,j)$ accounting for the third term on equation(1).(Sanity Check2)
Combine these three pluses, the min st-cut capacity equals to the minimum sum of purchase costs and incompatibility costs(Sanity Check4). And we successfully assign all n tasks to two landscaper by vertex on S,T.

Proof of Time Complexity :
For algorithm step 1 in which we set the network, there are A, B, i for i=1,...,n vertices, totally costs O(1+n+1)=O(n) time to set these vertices;

Besides, there are edges A→ i for i=1,...,n, total n edges; $i \to B$ for i=1,..,n, total n edges; $i \to j$ for i,j=1,..,n and i≠j, total $n(n-1)$ edges. Therefore, on N total O(n+n+n(n-1)) = O(n^2) edges.

For step 2, we run the MaxFlow algorithm on N whose run-time is O(number(vertex)×number(edge)) by lecture. Therefore, run-time in this step is O(n)×O(n^2)=O(n^3). Build residual graph need to go over all edges and copy all vertices with run-time O(n+n^2)=O(n^2). Using BFS to find the min ct-cut takes O(n+n^2)=O(n^2) time. Step 2, runtime is O(n^3+n^2 + n^2)= O(n^3).

For step 3, it does search and assignment on S,T with run-time O(n).

Add these steps together, the overall run-time complexity for this algorithm is O(n^3). Thus, it is a polynomial-time algorithm.

2.2 b:

Input: n,a_i,b_i for i = 1,...,n, c(i,j) for i,j = 1,...,n,i≠j and c(i,j)=c(j,i), d. Explanation: n projects; a_i A company charges a_i to do job i; b_i B company charges b_i to do job i; $c(i,j)$ incompatibility cost to get jobs i and j done by different companies. For convenience to read, we only write c(i,j) for all i, j and i<j; d discount when purchase job i from one company, the competing company will subtract d from their price to complete job (i + 1)
Output: Projects assignment for A and B landscapers so as to minimize the sum of the purchase costs and incompatibility costs accounting for discount.

Algorithm: We do modification on Algorithm 1a step 1, build a different network N'. On N', we change the edge $i \to j$ capacity c(i,j) to c(i,j)-d, $j \to i$ capacity c(j,i) to c(j,i)-d where i and j are adjacent projects. The other vertex/edge/capacity keeps the same. The detailed algorithm is as the following figure 4 and description.

1. Build a new network N' with vertices and edges like the Figure 4.
2. Construct min-cut (S,T) in N:
   Run the MaxFlow algorithm on N' getting the max-flow f and its value v(f).
   Build residual graph N_f using N and f.
   Find min st-cut (S,T) using N_f and BFS.

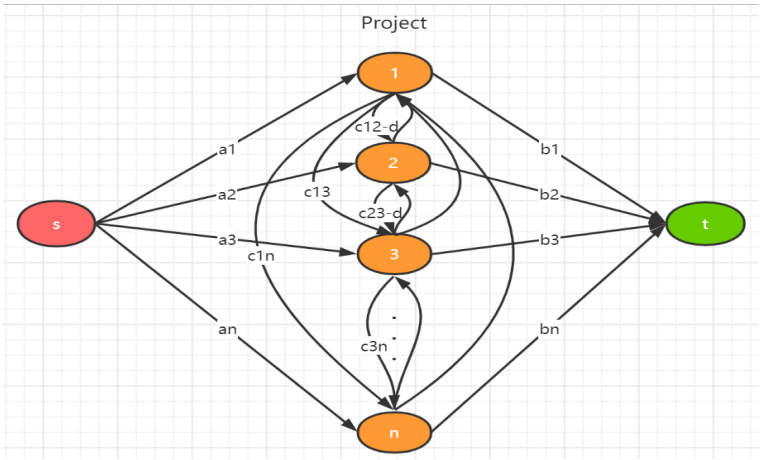

Figure 4: Network flow N' for 3b
On N', vertex s is the source representing company B, vertex t is the sink representing company A. vertex i for i = 1, ..., n is the i-th project.
Edges A$\rightarrow i$ has capacity a_i,Edges $i \rightarrow B$ has capacity b_i for i=1,...,n,Edge $i \rightarrow j$ has the same capacity c(i,j) as edge $j \rightarrow i$ when i and j is not adjacent project. Edge $i \rightarrow j$ has the same capacity c(i,j)-d as edge $j \rightarrow i$ c(j,i)-d when i and j is not adjacent project.

3. According to min-cut(S,T), assign all projects in S(exclude source vertex) to B company, assign all projects in T(exclude sink vertex) to A company.

Sanity Check: The network N' meets all restrictions on the problem.
1. Every job should be done exactly once
2. We need to account for the incompatibilities between the styles of the different suppliers
3. Abed charges a_i to do job i, and Britta charges b_i for the same job.
4. Minimize the sum of purchase costs and incompatibility costs accounting for discount
5.If purchase job i from one company, the competing company will offer discount d on job (i + 1).
Proof of Correctness:
We want to Minimize the sum of purchase costs and incompatibility costs, write the formula:

$$\min(\sum_{i\in A} a_i + \sum_{j\in B, B=I\setminus A} b_j + \sum_{(i,j)\in P} c(i,j) - \sum_{(i,j)\in D} d)(2)$$
$$= \min(\sum_{i\in A} a_i + \sum_{j\in B, B=I\setminus A} b_j + \sum_{(i,j)\in D} (c(i,j) - d) + \sum_{(i,j)\in P\setminus D} c(i,j))(3)$$

Set I={all projects from 1 to n}, P={(i,j) $|i \in A$,j∈B}. D={(i,j) | i∈A,j∈B, $|i-j|$=1}
By algorithm, we get the min st-cut (S,T) and its value c(S,T). We want to show that the value of equation(3) is exactly the capacity of the min-cut in the network we build. Projects j contained on S are assigned to B company, projects i contained on T are assigned to A company, each vertex could only be on one side in the min-cut(Sanity Check1). Since c(S,T) sum up all S$\rightarrow$T edges' capacity, saying project j is contained on S, project i is contained on T. S$\rightarrow$T edges include edges from j to B and edges from A to i. Sum up the capacity of edges from A to i equals to $\sum_{i\in A} a_i$, sum up the the capacity of edges from j to B equals to $\sum_{j\in B, B=I\setminus A} b_j$(Sanity Check3). They meet the first and second term on equation(3). S$\rightarrow$T edges also include edges from j in S to i in T, these edges' capacity are the incompatibility costs between i and j. And we modify the two direction adjacent

project edge capacity to (c(i,j)-d) accounting for the discount. Since we draw arrows in different directions on the i, j, c(i,j)=c(j,i),c(i,j)-d= c(j,i)-d, it only add one incompatibility cost for not adjacent project and one incompatibility cost minus discount on adjacent project. Sum up the capacity of adjacent project edges from j to i equals to $\sum_{i\in A, j\in B, i=i+1}(c(i,j)-d)$ (Sanity Check5) and sum up the capacity of the left no adjacent project edges from j to i equals to $\sum_{left(i\in A, j\in B)} c(i,j))$ accounting for the third and forth term on equation(3).(Sanity Check2,4)
Combine these four pluses, the min st-cut capacity equals to the sum of purchase costs and incompatibility costs accounting for discount(Sanity Check4). And we successfully assign all n tasks to two landscaper by vertex on S,T.

Proof of Time Complexity :
For algorithm step 1 in which we set the network, there are A, B, i for i=1,...,n vertices, totally costs O(1+n+1)=O(n) time to set these vertices;

Besides, there are edges A$\rightarrow i$ for i=1,...,n, total n edges; $i \rightarrow B$ for i=1,..,n, total n edges; $i \rightarrow j$ for i,j=1,..,n and i$\neq$j, total $n(n-1)$ edges. Therefore, on N total O(n+n+n(n-1)) = O(n^2) edges.

Next will be different from 3a algorithm, subtracting d from c(i,j) for every adjacent i and j in the job list, which costs O(n) time. So the total time complexity for setting the graph is still $O(n^2)$

For step 2, we run the MaxFlow algorithm on N whose run-time is O(number(vertex)$\times$number(edge)) by lecture. Therefore, run-time in this step is O(n)$\times$O(n^2)=O(n^3). Build residual graph need to go over all edges and copy all vertices with run-time O(n+n^2)=O(n^2). Using BFS to find the min ct-cut takes O(n+n^2)=O(n^2) time. Step 2, runtime is O(n^3+n^2 + n^2)= O(n^3).

For step 3, it does search and assignment on S,T with run-time O(n).

Add these steps together, the overall run-time complexity for this algorithm is O(n^3). Thus, it is a polynomial-time algorithm.