Homework 3

Wenjun Zhang

A53218995

Problem 1: Job scheduling (KT 7.41)

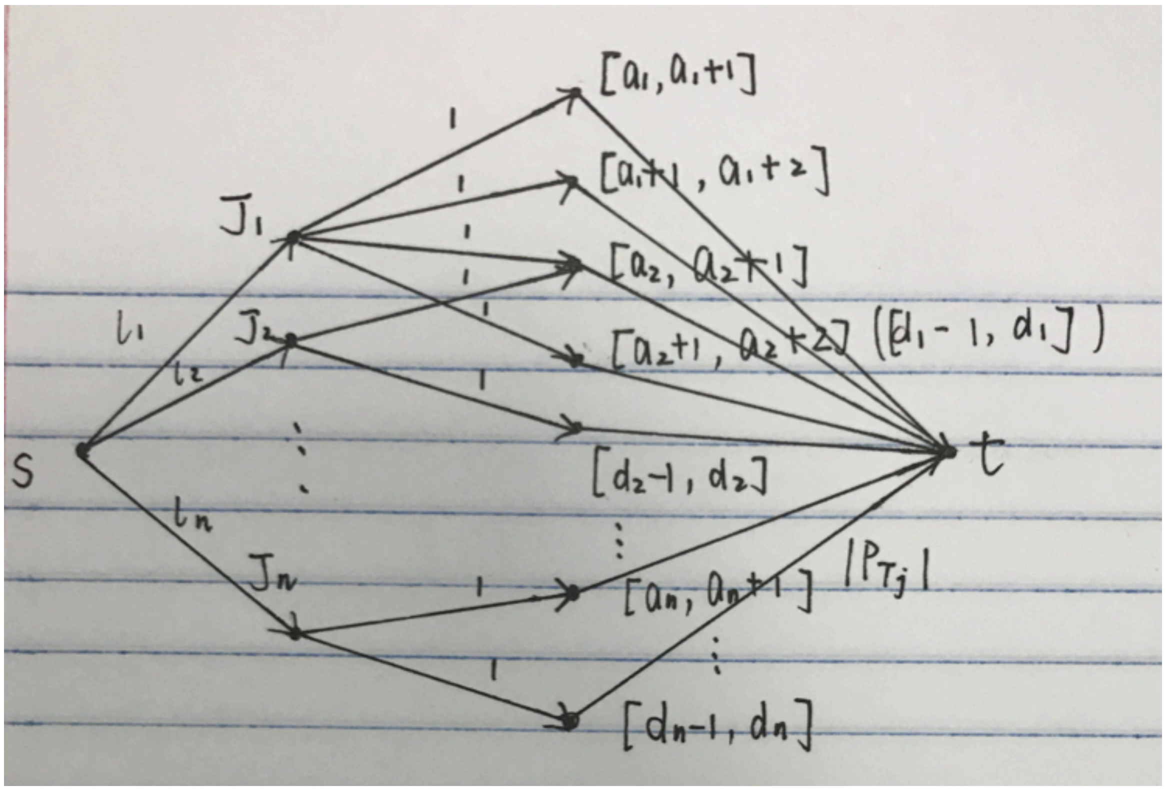
Algorithm description:

We are given a set of jobs and their corresponding arrival times, lengths and deadlines, and we also have a collection of processors which are not available all the time. We are asked to manage a job scheduling to check whether the jobs can all be completed or not.

We can change this problem into a max flow problem. Suppose the jobs set we have is $J = \{J_1, J_2, \dots, J_n\}$. For each J_i, its corresponding arrival time is a_i, deadline is d_i and length is l_i. We also have a set of processors $P = \{P_1, P_2, \dots, P_m\}$ and for each processor it is available during $[t_i, t_i']$. We also define a set $T = \cup_{i\in[1,n]} [a_i, d_i]$, where the element in set *T* is unit length interval such as $T_1 = [a_1, a_1 + 1]$.

We can construct a network $G = (V, E)$, where $V = \{s, t\} \cup J \cup T$ and $E = \{s\} \times J \cup \{(J_i, T_k) | T_k \in [a_i, d_i]\} \cup T \times \{t\}$. $\{s, t\}$ in V are the source and sink, and for the edge set, we link the source *s* with every job and link every job with their available processing time $T_k \in [a_i, d_i]$, and we also link all the available time interval of every job to the sink *t*.

For the capacity of each edge, we define $C(s, J_i) = l_i$, $C(J_i, T_j) = 1$ and $C(T_j, t) = |P_{T_j}|$, where $|P_{T_j}|$ is number of available processors during T_j. The network we construct is shown as below:



The capacities are all integer, so we can use the Labeling algorithm to find the max flow. If the max flow with value $\sum_{i=1}^{n} l_i$ exists, we can manage a job scheduling that the jobs can all be completed before their deadlines. Otherwise, no such scheduling exists.

Proof of correctness:

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem.

If there is a flow with value $\sum_{i=1}^{n} l_i$, then the capacities of all edges between the source *s* and every job J_i are used up, meaning that every job J_i is assigned l_i time intervals to process. Next, since every job only links to the source *s*, or links to their available process time intervals with 1 capacity respectively. So every job is finished during their available process time duration. For each time interval, they link to the sink *t* with the capacity equal to the number of available processors during that interval, so during every time interval the number of working processors is bounded by the number of available processors.

If no such flow exists, it means the flow value is smaller than $\sum_{i=1}^{n} l_i$. Since the capacity of edge is l_i, and the flow value of the network cannot exceed $\sum_{i=1}^{n} l_i$. This means that there will be a part of the jobs that cannot be finished, so there isn't such job scheduling.

Therefore, if and only if such max flow with value $\sum_{i=1}^{n} l_i$ exists, there will be a job scheduling ensuring all jobs be completed before their deadlines.

Time complexity:

We first need to construct the network, which has at most vertices $|V| = 2 + n + \sum_{i=1}^{n}(d_i - a_i)$ and edges $|E| = n + 2\sum_{i=1}^{n}(d_i - a_i)$. So the construction takes $O(|V| + |E|) = O(\sum_{i=1}^{n}(d_i - a_i))$. We are using Labeling algorithm to find the max flow, and the time complexity is $O(|E|C)$, where the max flow has the value no greater than $\sum_{i=1}^{n} l_i$. So the total complexity of the algorithm is

$$T(n) = O(\sum_{i=1}^{n}(d_i - a_i)\sum_{i=1}^{n} l_i)$$

Problem 2: Graph cohesiveness (KT 7.46)

Question 1

Algorithm description:

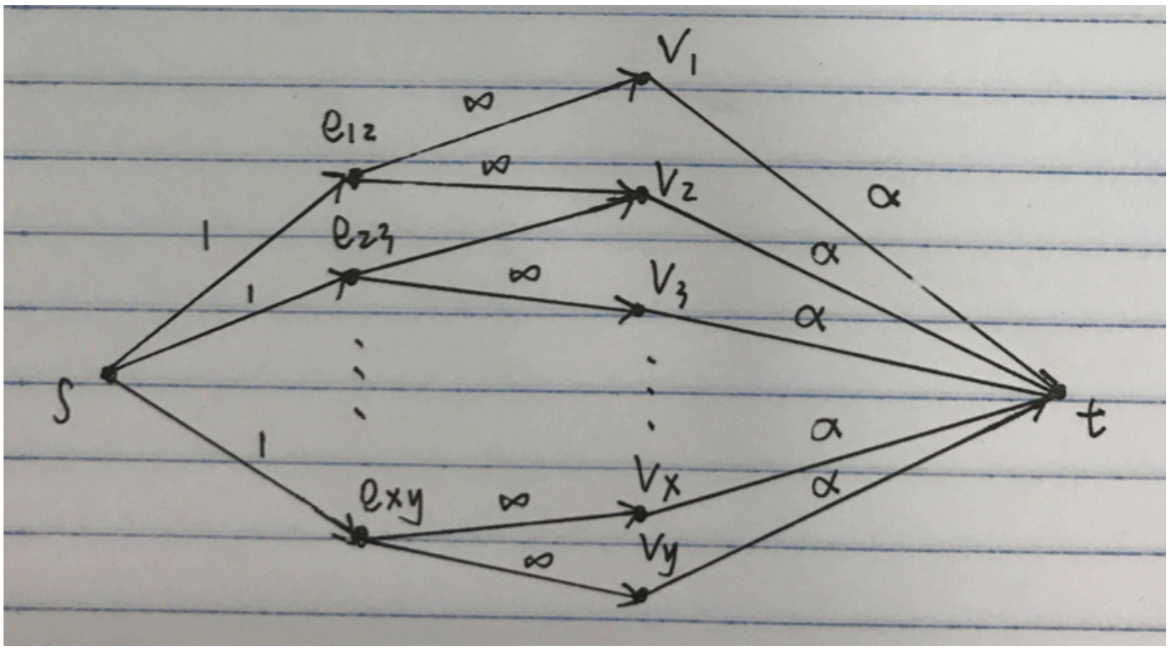
We are given a graph $G = (V, E)$ where nodes present people and edges represent friendship and we are asked to give a polynomial-time algorithm that takes a rational number α and determines whether there exists a set S with cohesiveness at lease α.

We can construct a flow network G' based on the graph G provided in the problem. The vertices and edges of the network G' are described as below:

For the vertices set, we first include a source and sink as well as the vertices of the provided graph, meaning $V' = \{s, t\} \cup V$. Then for every edge $(x, y) \in E$, we add a node e_{xy} into V'.

For the edges set, we link the source *s* to every e_{xy}, and then we link every e_{xy} node to its corresponding vertices v_x and v_y. Besides, we also link every $v_i \in V$ to the sink t.

For the capacities of edges, we define $C(s, e_{xy}) = 1$, $C(e_{xy}, v_x) = C(e_{xy}, v_y) = \infty$ and $C(v_i, t) = \alpha$. The network we construct is shown as below:



Given α is a rational number, we can apply the Preflow-push method to find max flow. If there exists max flow with value smaller than $|E|$, which is the number of edges linked to the source *s*, then there is a set *S* with cohesiveness at lease α. Otherwise, no such set S exists.

Proof of correctness:

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem.

We assume the min cut (S, T) where $s \in S$ and $t \in T$. We first prove that e_{xy}, v_x and v_y must be included in the same set. Since $C(e_{xy}, v_x) = C(e_{xy}, v_y) = \infty$, if we do not include v_x and v_y in the set which e_{xy} belongs to, it will lead to infinite capacity. Similarly, if v_x and v_y are in a particular set, then including e_{xy} will only decrease the cut capacity.

Based on the truth mentioned above, the capacities only consist of 1 and α, which can be expressed as $Capacity(S, T) = \sum_{e_{xy} \notin S} 1 + \sum_{v_x \in S} \alpha$, where there are $|E|$ kinds of e_{xy}, so we can also write it as $Capacity(S, T) = \sum_{(x,y) \in E} 1 - \sum_{e_{xy} \in S} 1 + \sum_{v_x \in S} \alpha$ or $Capacity(S, T) = |E| - e(S) + \alpha|S|$.

Under the max-flow min-cut theorem, when max flow with value smaller than $|E|$, $Capacity(S, T) < |E|$, which means

$$Capacity(S, T) = |E| - e(S) + \alpha|S| < |E|$$

and thus,

$$e(S) > \alpha|S| \quad \rightarrow \quad \frac{e(S)}{|S|} > \alpha$$

On the other hand, we can prove that when max flow with value $|E|$ (since there are only $|E|$ edges linked to the source, the max flow is bounded by $|E|$), there won't exist such a set.

If the set S is still the one of min cut (S, T), from the previous description we will have

$$Capacity(S, T) = |E| - e(S) + \alpha|S| = |E| \quad \rightarrow \quad \frac{e(S)}{|S|} = \alpha$$

If the set S is not the one of min cut (S, T), let's assume it is S', to achieve

$$Capacity(S', T') = |E| - e(S) + \alpha|S| < |E|$$

we will have to make $Capacity(S', T') < Capacity(S, T)$, which disobey the fact that $Capacity(S, T)$ being the min cut.

Therefore, if and only if max flow with value smaller than $|E|$ exists, there will be a set *S* with cohesiveness at lease α. Otherwise, there won't.

Time complexity:

We first need to construct the network, which has vertices $|V'| = 2 + |E| + |V|$ and edges $|E'| = |E| + 2|V|$. So the construction takes $O(|V'| + |E'|) = O(|V| + |E|)$. The time complexity for FIFO Preflow-push algorithm is $O(|V'|^3)$ So the total complexity of the algorithm is

$$T(n) = O((|V| + |E|)^3)$$

Question 2

Algorithm description:

We are asked to give a polynomial time algorithm to find a set S of nodes with maximum cohesiveness.

From the expression of cohesiveness $\frac{e(S)}{|S|}$, for the denominator $|S|$ there are $|V|$ choices of value and for the numerator $e(S)$ there are $C_2^{|V|} = \frac{|V|(|V|-1)}{2}$ choices. So there are $O(|V|^3)$ choices of rational number in total.

For every rational number derived above, we can start from the largest one to the smallest one and apply the algorithm mentioned in Question 1. Then we can find the maximum cohesiveness.

Proof of correctness:

The rational numbers we derive all are the candidate maximum cohesiveness. Since we start from the larger one to the smaller one, we go through all the possible maximum cohesiveness and we can stop after we obtain the maximum cohesiveness.

Time complexity:

There are $O(|V|^3)$ choices of cohesiveness number and time complexity for the algorithm in Question 1 is $O((|V| + |E|)^3)$. So the total complexity of this algorithm is

$$T(n) = O(|V|^3(|V| + |E|)^3)$$

Problem 3: Number puzzle

Algorithm description:

We are given sums for each row and column of a $n \times n$ matrix and we are asked to reconstruct the matrix.

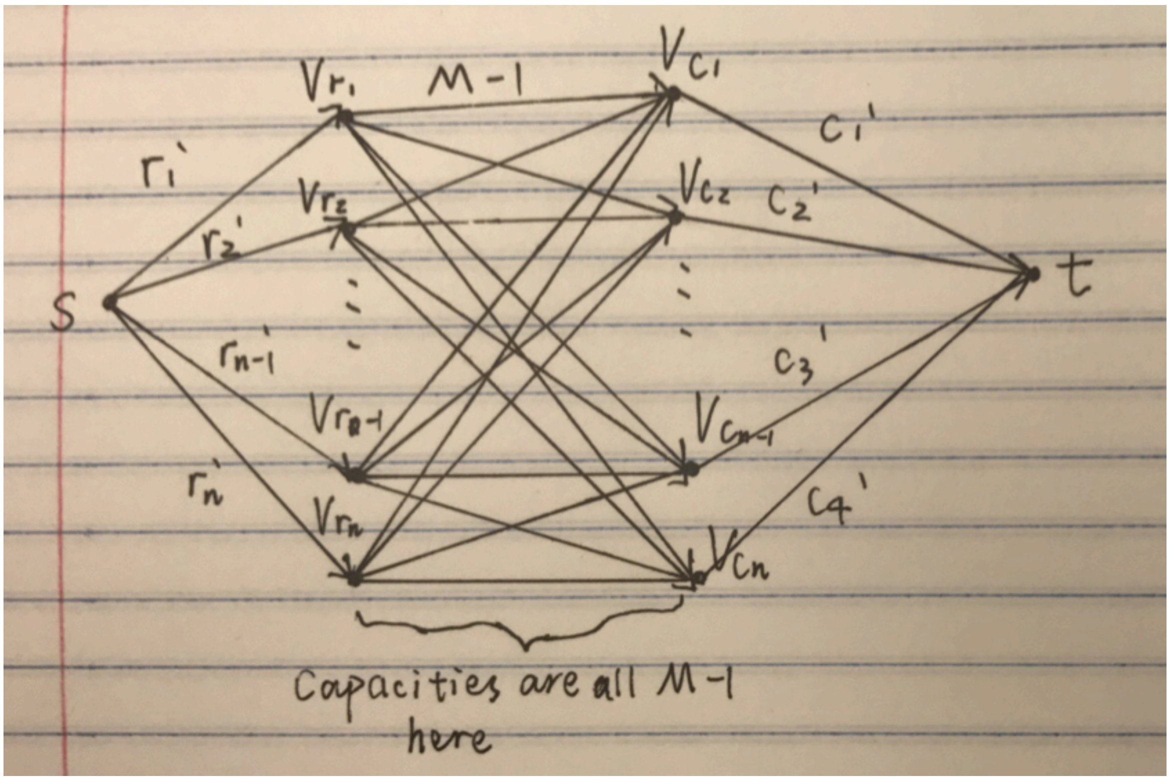
We can convert this problem into a max flow problem. But to begin with, we need to check if $\sum_{i=1}^{n} r_i = \sum_{j=1}^{n} c_j$ and $r_i \geq n$ and $c_j \geq n$ for $1 \leq i, j \leq n$.

Since the integers are in the range $1, \dots, M$, to convert this into a max flow network, we first need to reduce every r_i and c_j by n for $1 \leq i, j \leq n$, which means $r_i' = r_i - n$ and $c_j' = c_j - n$ for $1 \leq i, j \leq n$. Then we can start to construct the network.

The vertices set of the network can be expressed as $V = \{s, t\} \cup V_r \cup V_c$, where $\{s, t\}$ are the source and sink, V_r represents the *n* reduced row sums and V_c represents the *n* reduced column sums.

For the edges set, $E = \{s\} \times V_r \cup V_r \times V_c \cup V_c \times \{t\}$. We first link the source with every row sum, and every row sum links to every column sum since there is always a mutual element between them. And we also link every column sum to the sink.

For the capacities of edges, we define $C(s, V_{r_i}) = r_i'$, $C\left(V_{r_i}, V_{c_j}\right) = M - 1$ and $C\left(V_{c_j}, t\right) = c_j'$. The network we construct is shown as below:



The capacities are all integer, so we can use the Labeling algorithm to find the max flow. If the max flow with value $\sum_i^n r_i'$ exists, we can reconstruct the matrix whose element a_{ij} is equal to the flow on edge $\left(V_{r_i}, V_{c_j}\right)$ plus one. Otherwise, we cannot reconstruct the matrix.

Proof of correctness:

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem. But to begin with, we need to prove that making $r_i' = r_i - n$ and $c_j' = c_j - n$ for $1 \leq i, j \leq n$ in this problem will not influence the correctness of the solution.

The elements of the matrix are in the range $1, \ldots, M$. But as we all know in the max flow problem the flow on an edge can be zero, so we need to make each element in the matrix minus 1 to make them in the range $0, \ldots, M-1$. If a matrix whose elements are in the range $0, \ldots, M-1$ exists, then by adding 1 to each of its element will lead to the result we want for this problem.

If there is a flow with value $\sum_{i=1}^n r_i'$, then the capacities of all edges between the source *s* and V_{r_i} are used up, meaning that for every row i, the sum of i row is exactly r_i'. Next, since every V_{r_i} links to every V_{c_j}, it means there will always be a non-negative value passing through V_{r_i} to V_{c_j}, and it's the mutual element of i row and j column, and we can write it as a_{ij}'. And since the capacity of edge $\left(V_{r_i}, V_{c_j}\right)$ is $M-1$, we guarantee that the value of a_{ij}' is within the range $[0, M-1]$. Every V_{c_j} links to the sink *t* and the capacity for every edge $\left(V_{c_j}, t\right)$ is c_j', because $\sum_{i=1}^n r_i' = \sum_{j=1}^n c_j'$, to achieve a flow with value $\sum_{i=1}^n r_i'$, the capacity of every edge $\left(V_{c_j}, t\right)$ is used up as well, meaning the column sum of j column is exactly c_j'.

If no such flow exists, it means the flow value is smaller than $\sum_{i=1}^n r_i'$, since the capacity of edge is r_i', and the flow value of the network cannot exceed $\sum_{i=1}^n r_i'$. This means that there will be a part of the rows that don't sum up to their corresponding r_i', the required matrix cannot be constructed.

Finally, we make $a_{ij} = a_{ij}' + 1$, which can ensure that $a_{ij} \in \{1, \ldots, M\}$, and since $\sum_{j=1}^n a_{ij}' = r_i'$ for $1 \leq i \leq n$ and $\sum_{i=1}^n a_{ij}' = c_j'$ for $1 \leq j \leq n$, we have $\sum_{j=1}^n a_{ij} = r_i$ for $1 \leq i \leq n$ and $\sum_{i=1}^n a_{ij} = c_j$ for $1 \leq j \leq n$.

Therefore, if and only if such max flow with value $\sum_{i=1}^{n} r_i{}'$ exists, we can reconstruct the matrix.

Time complexity:

We first need to construct the network, which has vertices $|V| = 2n + n$ and edges $|E| = n^2 + 2n$. So the construction takes $O(|V| + |E|) = O(n^2)$. We are using Ford-Fulkerson algorithm to find the max flow, and the time complexity is $O(EC)$, where the max flow has the value no greater than $\sum_{i=1}^{n} r_i{}'$, so it takes $O(n^2 \sum_{i=1}^{n} r_i{}')$. Adding one to every element in the matrix takes $O(n^2)$. So the total complexity of the algorithm is

$$T(n) = O(n^2 \sum_{i=1}^{n} r_i{}')$$

Problem 4: Database projections (KT 7.38)

Algorithm description:

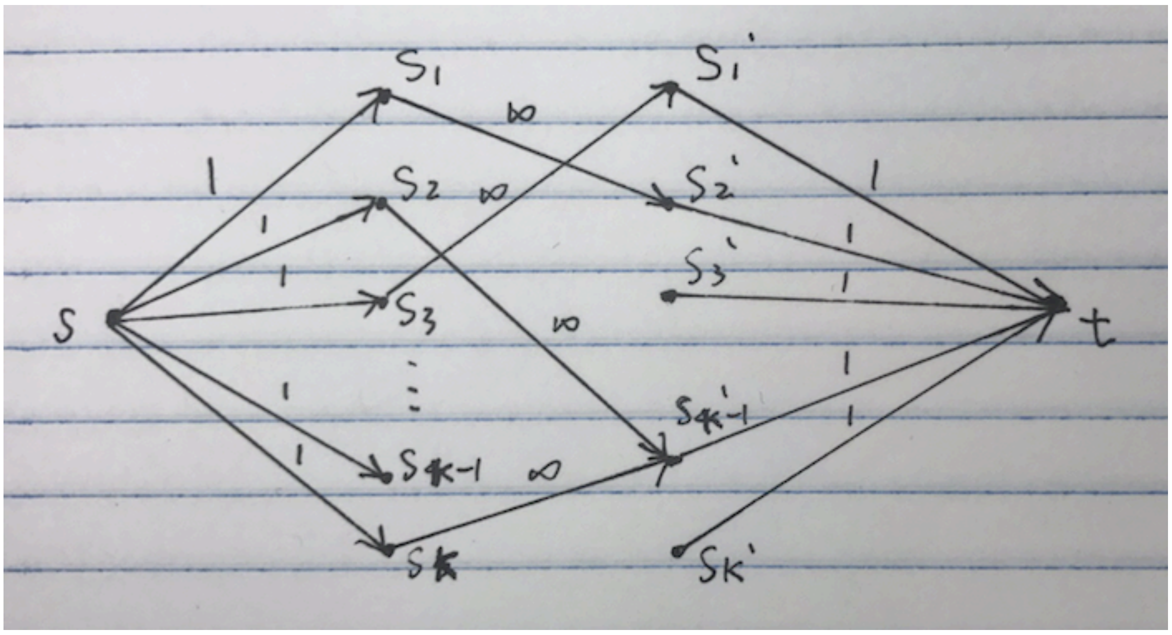
We are given subsets S_i (for $i = 1, 2, \ldots, k$) and a parameter $l < k$ and we are asked to find l permutations of the columns that for every one of the given subsets, it's the case that the column in S_i constitute a prefix of at least one of the permuted tables.

We can convert this problem to a max flow problem. We first duplicate a set S' of size k, where $S_i' = S_i$ for $1 \le i \le k$. Then we can construct a network described as below:

For the vertices set, $V = \{s, t\} \cup S \cup S'$, where $\{s, t\}$ are the source and sink, and the S_i (for $i = 1, 2, \ldots, k$) and their replicas are included so as to restore the prefix relation.

For the edges set, we first link the source *s* to all subsets S_i (for $i = 1, 2, \ldots, k$) and link all the replicas S_i' (for $i = 1, 2, \ldots, k$) to the sink *t*. Next, if $S_i \subseteq S_j'$ for $1 \le i, j \le k$ and $i \neq j$, we add an edge (S_i, S_j) to the edges set E.

For the capacities of the edges, we define $C(s, S_i) = 1$, $C\big(S_i, S_j'\big) = \infty$ and $C\big(S_j', t\big) = 1$. The network we construct is shown as below:



The capacities are all integer, so we can use the Labeling algorithm to find the max flow. If the max flow value is greater than or equal to $k - l$, there ia a set of valid permutations $p_1, p_2, \ldots, p_l$. Otherwise, no l such permutations exist.

Proof of correctness:

To prove the correctness of the algorithm, we need to show that the max flow in the graph corresponds to the solution of the original problem.

Let's first assume the max flow of the network is f. Based on the network we construct, there are f edges $\big(S_i, S_j'\big)$, meaning there are f pairs of $\big(S_i, S_j'\big)$ where $S_i \subseteq S_j'$ with

$1 \le i, j \le k$ and $i \neq j$. Since $S_i \subseteq S_j{}'$, for every pair, we can use the permutation of $S_j{}'$ to represent both of the subsets S_i and $S_j{}'$. Note that in the example network, we have $S_3 \subseteq S_1 \subseteq S_2 \subseteq S_{k-1}$, there are 3 pairs and we can use one permutation of S_{k-1} to represent all of them, which is exactly four minus three. There are k subsets in total, to represent all the subsets we will require $k - f$ permutations. Given we can construct a permuted table of size l, if the max flow value is greater than or equal to $k - l$, this means we can represent all the subsets using no more than l permutations, which is the same as the requirements of the problem.

If the max flow value is smaller than $k - l$, let's assume the max flow is again f and we have $k - f > l$. We want to construct a valid permuted table with less than $k - f$ permutations, suppose we can use $k - f'$ permutations where $f' > f$. To do this we will need f' pairs of $(S_i, S_j{}')$ such that $S_i \subseteq S_j{}'$ with $1 \le i, j \le k$ and $i \neq j$. If we have that many pairs, the flow of the network will be f', which contradicts that fact that the max flow value is f. So it's impossible to use less than $k - f$ permutations to represent the all the subsets, let alone l permutations.

Therefore, if and only if the max flow value is greater than or equal to $k - l$, there are a set of valid permutations $p_1, p_2, \dots, p_l$.

Time complexity:

We first need to construct the network, which has vertices $|V| = 2k + 2$ and edges at worst $|E| = O(k^2)$. So the construction takes $O(|V| + |E|) = O(k^2)$. We are using Ford-Fulkerson algorithm to find the max flow, and the time complexity is $O(EC)$, where the max flow has the value no greater than k, so it takes $O(k^3)$. So the total complexity of the algorithm is

$$T(n) = O(k^3)$$

Problem 5: Maximum likelihood points of failure

Algorithm description:

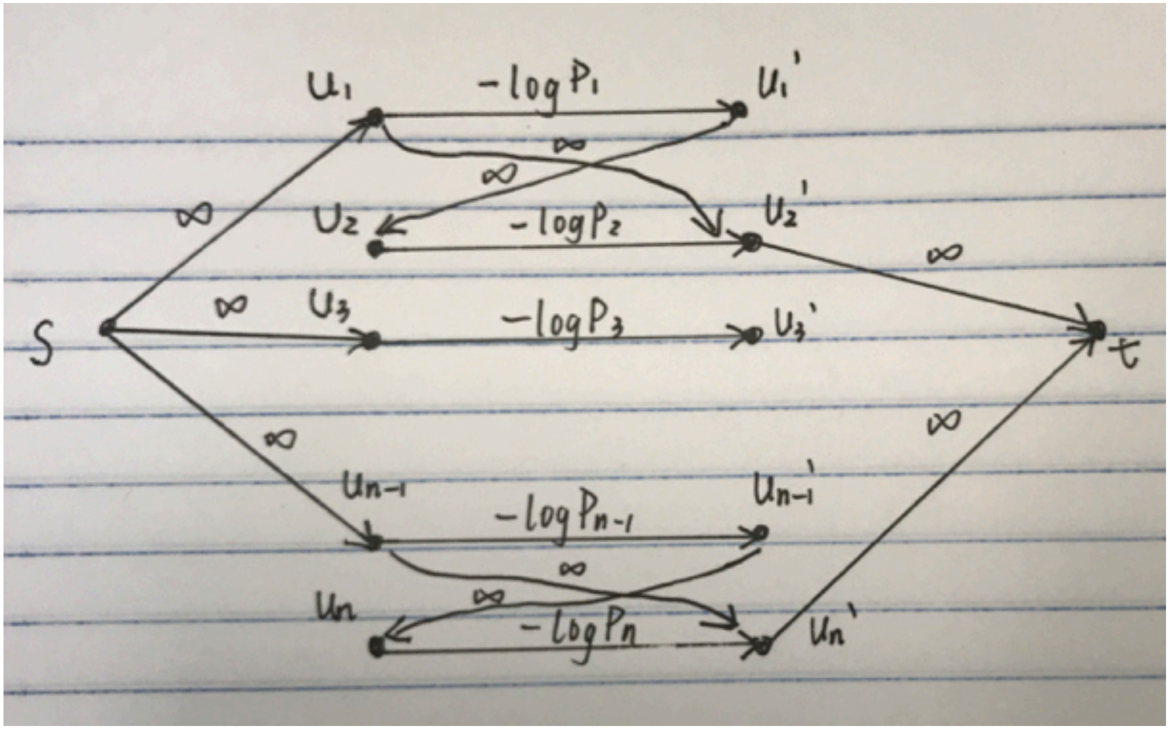
We are given a network and we are asked to find a subset F of failure point with the maximum failure probability.

We can convert this problem into a flow problem. We first define a vertices set U, which includes the same vertices of V except for s and t, meaning $U \cup \{s,t\} = V$ and $U \cap \{s,t\} = \emptyset$. Then we duplicate U and make $U_i{}' = U_i$. Then we can construct a network described as below:

For the vertices set, $V' = \{s,t\} \cup U \cup U'$, where $\{s,t\} \cup U = V$ is the vertices set of the provided network. And we add a replica of the vertices set except for the source and sink to the vertices set.

For the edges set, we first link the source *s* to U_i and link $U_j{}'$ to the destination *t* if there is an edge (s, U_i) or (U_j, t) for $1 \le i, j \le n$ and $i \ne j$ in the provided edges set E. Next, we link U_i to $U_j{}'$ and link $U_k{}'$ to U_l if there is an edge (U_i, U_j) or (U_k, U_l) for $1 \le i, j, k, l \le n$ and $i \ne j$, $k \ne l$ in the provided edges set E. In addition, we also link every $(U_i, U_i{}')$ for $1 \le i \le n$.

For the capacities of the edges, we define every $C(s, U_i) = \infty$, $C(U_i{}', t) = \infty$, $C(U_i, U_j{}') = \infty$ and $C(U_k{}', U_l) = \infty$ for $1 \le i, j, k, l \le n$ and $i \ne j$, $k \ne l$. Since we are looking for a maximum of $\prod_{i \in F} p_i$, to convert it into a min cut problem, we can take the log form. So we assign $C(U_i, U_i{}') = -\log(p_i)$ for $1 \le i \le n$. The network we construct is shown as below:



The capacities are probably irrational, so we need to apply Preflow-push method to find the min cut. Suppose the min cut (S, T) where the source *s* belongs to S and the destination *t* belongs to T, we assume the set of endpoints of the cutting border edges is F'. The nodes included in F' corresponds to a set F of original nodes in the provided vertices set V, which is the subset of failure point with the maximum failure probability.

Proof of correctness:

To prove the correctness of the algorithm, we need to show that the min cut solution corresponds to the solution of the original problem.

First we need to prove that the min cut disconnect the *t* from *s*. This is actually quite obvious due to the definition of a cut, which is a set of edges whose removal will divide the network into two separate parts.

Next, since only the capacities of edges (U_i, U_i') have finite value, the cut only includes edges among (U_i, U_i'). Otherwise, the cut will have value infinity. Let's assume the min cut value is m, and this means the capacity of the min cut is m, which can be expressed as

$$m = \sum_{U_i \in F} -\log(p_i) = \log\left(\frac{1}{\prod_{U_i \in F} p_i}\right)$$

Since m is the minimum and log function is monotonically increasing, term $\prod_{U_i \in F} p_i$ is the maximum, and F is the subset of failure point with the maximum failure probability.

Time complexity:

We first need to construct the network, which has vertices $|V'| = O(|V|)$ and edges $|E'| = O(|V| + |E|)$. So the construction takes $O(|V| + |E|)$. We use FIFO preflow-push algorithm, whose time complexity is $O(|V'|^3)$, so it takes $O(|V|^3)$. Thus, the total complexity of the algorithm is

$$T(n) = O(|V| + |E|) + O(|V|^3) = O(|V|^3 + |E|)$$