Algorithms: CSE 202 — Homework 3 Solutions

Problem 1: Graph cohesiveness (KT 7.46)

In sociology, one often studies a graph G in which nodes represent people and edges represent those who are friends with each other. Let's assume for purposes of this question that friendship is symmetric, so we can consider an undirected graph.

Now suppose we want to study this graph G, looking for a "close-knit" group of people. One way to formalize this notion would be as follows. For a non-empty subset S of nodes, let $e(S)$ denote the number of edges in S-that is, the number of edges that have both ends in S. We define the *cohesiveness* of S as $e(S)/|S|$. A natural thing to search for would be a set S of people achieving the maximum cohesiveness.

1. Give a polynomial-time algorithm that takes a rational number α and determines whether there exists a set S with cohesiveness at least α.

2. Give a polynomial-time algorithm to find a set S of nodes with maximum cohesiveness.

Solution: Graph cohesiveness (KT 7.46)

Determining whether graph cohesiveness is strictly larger than a rational number α: Let $G = (V, E)$ be the friendship graph. For $S \subseteq V$, let $e(S)$ be the number of edges in E with both ends in S. Define the *cohesiveness* of S as $\frac{e(S)}{|S|}$.

Let α be a rational number. We design an efficient algorithm to determine whether there exists a vertex set S with cohesiveness strictly *larger* [1] than α.

Flow network: We construct a flow network $\mathbb{G} = (\mathbb{V}, \mathbb{E}, c)$ to solve the problem. $\mathbb{G}$ contains a source node s and a sink node t. For each vertex $x \in V$, $\mathbb{G}$ contains a node v_x. For each edge $(x, y) \in E$, $\mathbb{G}$ also contains a node $u_{x,y}$. $\mathbb{G}$ contains the following edges:

- source node s connects to each vertex of the form $u_{x,y}$ with an edge of capacity 1 [$s \longrightarrow u_{x,y}$ with capacity 1],
- each node of the form $u_{x,y}$ connects to nodes v_x and v_y with infinite capacity [$u_{x,y} \longrightarrow v_x$ and $u_{x,y} \longrightarrow v_y$ with capacity ∞],
- each node v_x connects to the sink node t with capacity α [$v_x \longrightarrow t$ with capacity α]

The flow network $\mathbb{G}$ is shown in Figure 1 schematically. It has the following properties

1. Since the source node s has exactly $|E|$ outgoing edges, each with capacity 1, the capacity of the min-cut($\mathbb{G}$) is less or equal to $|E|$.

2. Since α is a rational number, we can scale the capacities to integers.

3. The complexity of running the Preflow-Push maximum-flow algorithm on $\mathbb{G}$ is $O((|V| + |E|)^3)$.

[1] We will show later that cohesiveness of all subsets of G is a finite set of discrete rational numbers D. For an arbitrary rational number α, let β be the largest rational number in D such that $\beta < \alpha$. Then determining whether graph cohesiveness is *at least* α is equivalent to determining whether graph cohesiveness is strictly *larger* than β.

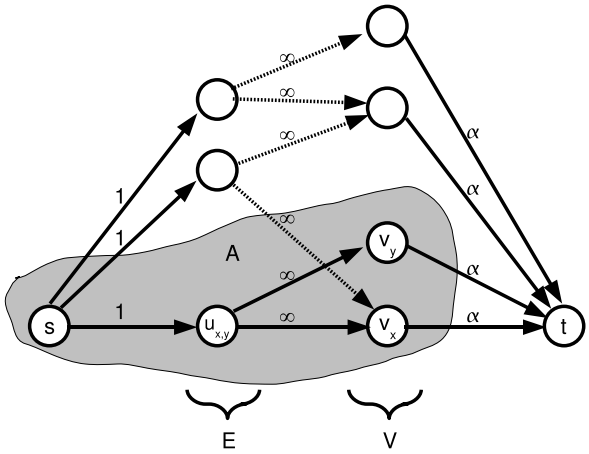

Figure 1: Flow network $\mathcal{G}$ to determine if graph cohesiveness is strictly larger than α

Theorem 0.1. *There exists a vertex set S in G with cohesiveness strictly larger than α if and only if the maximum flow in $\mathbb{G}$* cannot *saturate all the edges leaving the source s.*

Proof. Let (A, B) be a min-cut of the flow network $\mathbb{G}$ obtained by running the maximum flow algorithm. Let S^* be the set of vertices in A of the form v_x. Observe that $u_{x,y} \in A$ if and only if both $v_x, v_y \in S^*$. Indeed, if $u_{x,y} \in A$ and v_x or v_y were in B, then (A, B) will have infinite capacity, which is not possible since the cut (A, B) has finite capacity. If $v_x \in A$ and $v_y \in A$ but $u_{x,y} \notin A$, then adding $u_{x,y}$ to A decreases the cut capacity, contradicting the fact that (A, B) is a minimum cut.

The capacity of the minimum cut (A, B) is

$$\begin{aligned}\text{capacity}(A, B) &= \sum_{u_{x,y} \notin A} 1 + \sum_{v_x \in A} \alpha \\ &= (\sum_{(x,y) \in E} 1 - \sum_{u_{x,y} \in A} 1) + \sum_{x \in S^*} \alpha \\ &= |E| - e(S^*) + \alpha |S^*|\end{aligned}$$

If maximum flow cannot saturate all the edges leaving the source s, then by the max-flow min-cut theorem, capacity$(A, B) < |E|$, so $e(S^*) > \alpha |S^*|$, i.e., the set S^* has cohesiveness strictly larger than α.

Suppose there exists a set S with cohesiveness strictly larger than α and the maximum flow is $|E|$. This implies capacity$(A, B) = |E|$. We get a contradiction by constructing a cut of capacity smaller than $|E|$. Let $A' = \{s\} \cup \{v_x | x \in S\} \cup \{u_{x,y} | (x, y) \in E, x, y \in S\}$. Let B' be the remaining nodes of the flow network $\mathbb{G}$. We then have

$$\text{capacity}(A', B') = |E| - e(S) + \alpha |S| < |E|,$$

which leads to the desired contradiction since (A, B) is a minimum cut.

□

Finding a set S with maximum cohesiveness

There are $|V|$ choices for the size of $S \subseteq V$. There are $\binom{|S|}{2} = \frac{|S|(|S|-1)}{2}$ choices for $e(S)$. We thus have a total of $O(|V|^3)$ possible values for α. We apply binary search to determine the set with largest cohesiveness by using the algorithm that checks if there exists a subset with cohesiveness greater than a given value.

By applying the Preflow-Push algorithm to determine the minimum cut for a given α, we can compute the set with maximum cohesiveness in time $O((|V| + |E|)^3 \lg |V|)$.

Problem 2: Number puzzle

You are trying to solve the following puzzle. You are given the sums for each row and column of an $n \times n$ matrix of integers in the range $1 \ldots, M$, and wish to reconstruct a matrix that is consistent. In other words, your input is M, $r_1, \ldots, r_n$, $c_1, \ldots, c_n$. Your output should be a matrix $a_{i,j}$ of integers between 1 and M so that $\sum_i a_{i,j} = c_j$ for $1 \leq j \leq n$ and $\sum_j a_{i,j} = r_i$ for $1 \leq i \leq n$; if no such matrix exists, you should output, "Impossible". Give an efficient algorithm for this problem.

Solution: Number puzzle

Idea: The idea is to view the problem as that of distributing the sum of each row among the n columns while respecting the sum of every column.

Since we know that each entry should be at least 1, we allocate 1 to each entry of the matrix and distribute each row sum $r_i - n$ to the columns so the column sums turn out to be $c_j - n$. Now the entries in the matrix will be in the range $\{0, 1, \ldots, M-1\}$. If any of the the $2n$ values r_i and c_i are less than n or $\sum_i (r_i - n) \neq \sum_j (c_j - n)$, then output "impossible". Observe that if r_i and c_i were indeed row and column sums respectively, then $\sum_i r_i = \sum_i c_i$ must be equal to the sum of the entries of the matrix.

Flow network: We will construct a flow network which contains a node for every row and column in such a way that the flow corresponds to the distribution of row sums.

Let the flow network $G = (V, E, c)$ where c is capacity function. V contains a node for each row and a node for each column in addition to the source node s and sink node t. More specifically, $V = \{s, t\} \cup R \cup C$ where $R = \{u_1, \ldots, u_n\}$ and $C = \{v_1, \ldots, v_n\}$. The nodes in R represent rows and the nodes in C represent columns. The source node s is connected by a directed edge to each node u_i with capacity $r_i - n$. Each node in C is connected to the sink node t by a directed edge with capacity $c_i - n$. Each node u_i is connected to each node v_j with a directed edge of capacity $M - 1$. Formally,

$$\begin{aligned} V &= \{s, t\} \cup R \cup C \\ E &= \{s\} \times R \cup \{(u_i, v_j) \mid 1 \leq i, j \leq n\} \cup C \times \{t\} \end{aligned}$$

and the edge capacities are

$$\begin{aligned} c(s, u_i) &= r_i - n \\ c(u_i, v_j) &= M - 1 \\ c(v_j, t) &= c_j - n \end{aligned}$$

where c is the capacity function on edges.

We now run the Preflow-Push algorithm on the flow network. If the maximum flow is not equal to $\sum_i (c_j - n)$, then output "impossible". Otherwise, output the matrix M with $M_{i,j} = f(u_i, v_j) + 1$ where f is the flow function on the edges of G.

Complexity: The flow network has $O(n)$ vertices and $O(n^2)$ edges and it can be constructed in $O(n^2)$ time. One can apply the Preflow-Push algorithm for determining the maximum flow on this network to get an overall time bound of $O(n^3)$.

Correctness: Let $\sum_i (r_i - n) = \sum_j (c_j - n) = x$. The following two claims will establish the correctness.

Claim 0.2. *If there is a matrix A satisfying the constraints, then the max-flow is equal to x.*

Proof. We prove the claim by constructing a flow for G with value x. Define the following flow function for G:

$$\begin{aligned} f(s, u_i) &= r_i - n && \text{for } 1 \leq i \leq n \\ f(v_j, t) &= c_j - n && \text{for } 1 \leq j \leq n \\ f(u_i, v_j) &= A[i, j] - 1 && \text{for } 1 \leq i, j \leq n \end{aligned}$$

It is easy to check that f is a valid flow and its value is x. □

Claim 0.3. *If the max-flow of G is equal to x, then the algorithm outputs a matrix A satisfying the constraints.*

Proof. Let f be a maximum flow in G with value x. Let $A[i,j] = f(u_i, v_j) + 1$ for $1 \leq i, j \leq n$. We will argue that A has the desired properties.

Since the value of f is x, it must be that $f(s, u_i) = r_i - n$, $f(t, v_j) = c_i - n$. Since the flow is conserved at u_i,

$$\begin{aligned}\sum_{j=1}^{n} A[i,j] &= \sum_{j=1}^{n} f(i,j) + 1 \\ &= r_i.\end{aligned}$$

Similarly, since the flow is conserved at v_j,

$$\begin{aligned}\sum_{i=1}^{n} A[i,j] &= \sum_{i=1}^{n} f(i,j) + 1 \\ &= c_j.\end{aligned}$$

Clearly, $1 \leq A[i,j] \leq M$ for $1 \leq i, j \leq n$. A satisfies all the constraints and the claim is proved. □

Problem 3: Database projections (KT 7.38)

You're working with a large database of employee records. For the purposes of this question, we'll picture the database as a two-dimensional table T with a set R of m rows and a set C of n columns; the rows correspond to individual employees, and the columns correspond to different attributes.

To take a simple example, we may have four columns labeled

`name, phone number, start date, manager's name`

and a table with five employees as shown here. Given a subset S of the columns, we can obtain a new, smaller

Table 1: Table with five employees.

name	**phone number**	**start date**	**manager's name**
Alanis	3-4563	6/13/95	Chelsea
Chelsea	3-2341	1/20/93	Lou
Elrond	3-2345	12/19/01	Chelsea
Hal	3-9000	1/12/97	Chelsea
Raj	3-3453	7/1/96	Chelsea

table by keeping only the entries that involve columns from S. We will call this new table the *projection* of T onto S, and denote it by $T[S]$. For example, if $S = \{$`name, start date`$\}$, then the projection $T[S]$ would be the table consisting of just the first and third columns.

There's a different operation on tables that is also useful, which is to *permute* the columns. Given a permutation p of the columns, we can obtain a new table of the same size as T by simply reordering the columns according to p. We will call this new table the *permutation* of T by p, and denote it by T_p.

All of this comes into play for your particular application, as follows. You have k different subsets of the columns $S_1, S_2, \ldots, S_k$ that you're going to be working with a lot, so you'd like to have them available in a readily accessible format. One choice would be to store the k projections $T[S_1], T[S_2], \ldots, T[S_k]$, but this would take up a lot of space. In considering alternatives to this, you learn that you may not need to explicitly project onto each subset, because the underlying database system can deal with a subset of the columns particularly efficiently if (in some order) the members of the subset constitute a *prefix* of the columns in left-to-right order. So, in our example, the subsets {`name, phone number`} and {`name, start date, phone number,`} constitute prefixes (they're the first two and first three columns from the left, respectively); and as such, they can be processed much more efficiently in this table than a subset such as {`name, start date`}, which does not constitute a prefix. (Again, note that a given subset S_i does not

come with a specified order, and so we are interested in whether there is *some* order under which it forms a prefix of the columns.)

So here's the question: Given a parameter $\ell < k$, can you find ℓ permutations of the columns $p_1, p_2, \ldots, p_\ell$ so that for every one of the given subsets S_i (for $i = 1, 2, \ldots, k$), it's the case that the columns in S_i constitute a prefix of at least one of the permuted tables $T_{p_1}, T_{p_2}, \ldots, T_{p_\ell}$? We'll say that such a set of permutations constitutes a valid solution to the problem; if a valid solution exists, it means you only need to store the ℓ permuted tables rather than all k projections. Give a polynomial-time algorithm to solve this problem; for instances on which there is a valid solution, your algorithm should return an appropriate set of ℓ permutations.

Example. Suppose the table is as above, the given subsets are

$$S_1 = \{\texttt{name, phone number}\},$$
$$S_2 = \{\texttt{name, start date}\},$$
$$S_3 = \{\texttt{name, manager's name, start date}\},$$

and $\ell = 2$. Then there is a valid solution to the instance, and it could be achieved by the two permutations

$$p_1 = \{\texttt{name, phone number, start date, manager's name}\},$$
$$p_2 = \{\texttt{name, start date, manager's name, phone number}\}.$$

This way, S_1 constitutes a prefix of the permuted table T_{p_1}, and both S_2 and S_3 constitute prefixes of the permuted table T_{p_2}.

Solution: Database projections (KT 7.38)

Flow network: Let $S_1, \ldots, S_k$ be the given subsets of columns. We construct a bipartite graph $G = (U \cup V, E)$ where $U = \{u_1, \ldots u_k\}$ and $V = \{v_1, \ldots, v_k\}$. u_i and v_i represent the set S_i. E consists of exactly those edges (u_i, v_j) where $S_i \subset S_j$.

We construct a flow network $\mathbb{G} = (\mathbb{V}, \mathbb{E}, c)$ based on the bipartite graph.

- $\mathbb{V} = \{s, t\} \cup U \cup V$.
- $\mathbb{E}$ consists of the following edges with capacities as indicated:
  - (s, u_i) with capacity 1
  - (u_i, v_j) with capacity ∞
  - (v_j, t) with capacity 1

For the example given above, the graph is shown in Figure 2.

Output: Suppose the maximum flow of $\mathbb{G}$ is m. You will later see that $0 \leq m \leq k-1$. We assert that we need at least $k - m$ permutations. Therefore, if $k - m \leq \ell$, there is a valid solution.

Claim 0.4. *If the maximum flow of $\mathbb{G}$ is m, we can cover all subsets with $k - m$ permutations.*

Proof. Given a flow f of value m, we want to construct $k - m$ permutations that together cover all subsets. We assume without loss of generality that our flow is integral. Since the capacities of the outgoing edges of s and the incoming edges of t are 1, we conclude that all the flow values are either 0 or 1.

We use the fact that if a family of subsets form a chain, then they can be covered by the same permutation, e.g., if $S_1 \subset S_2 \subset S_3$, then we can construct a permutation that starts with all elements in S_1 (in any order), followed by all elements in $S_2 - S_1$, followed by all elements in $S_3 - S_2$, followed by the rest. This permutation has prefixes for S_1, S_2, and S_3.

We now show that if the value of the flow is m, we can construct $k - m$ chains such that each set is in exactly one chain.

Consider the set P of all paths of the form $s \to u_i \to u_j \to t$ in $\mathbb{G}$ where the flow along each of the edges in the path is 1. Since the value of the flow is m, $|P| = m$. Let P' be the set of pairs of subsets of the form (S_i, S_j) such that there is a path in P with the edge (u_i, v_j). Since the flow leaving u_i is at most 1, we have

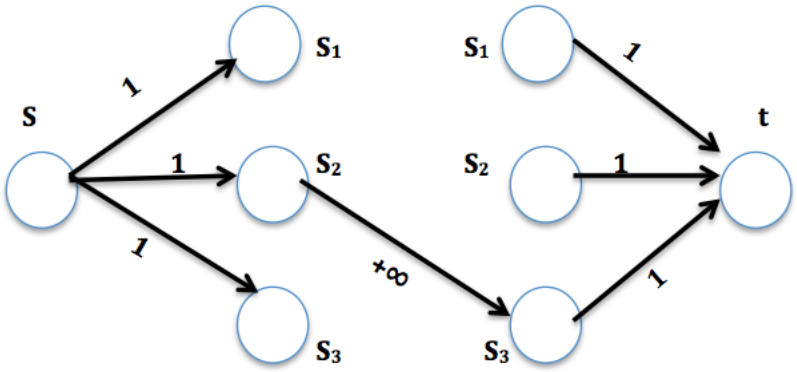

Figure 2: Flow network $\mathbb{G}$ for database projection

at most one pair of the form $(S_i, X) \in P'$. Similarly, since the flow leaving v_j is at most 1, we have at most one pair of the form $(X, S_j) \in P'$. P' can be viewed as a graph H on $S_1, \ldots, S_k$ with exactly m edges where the indegree and outdegree of each node is at most 1. Furthermore H is acyclic. We can partition H into maximal chains where each chain is formed by starting with a node of indegree 0 and following its outgoing edge as long as one exists. In this partition, a chain covers i sets if and only if it employs $i-1$ edges. Since we have k vertices and m edges in the graph, the partition consists of exactly $k-m$ disjoint chains.

Therefore, with a flow of m, we can cover all subsets with $k-m$ permutations. □

Claim 0.5. *If the maximum flow of $\mathbb{G}$ is m, we cannot cover all subsets with fewer than $k-m$ permutations.*

Proof. We prove the claim using a proof by contradiction. Suppose there is a set of $k-d$ permutations with $d > m$ that covers all subsets. We then construct a flow of value d in $\mathbb{G}$, contradicting the fact that the maximum flow is m.

Suppose we can cover all subsets with $k-d$ permutations where $d > m$. By assigning each set to a permutation that covers it, we partition the sets into $k-d$ groups. If a set is covered by more than one permutation, choose one arbitrarily. Let σ be one of the permutations. The subsets covered by σ must form a chain. Let $S_1^\sigma \subseteq S_2^\sigma \subseteq \cdots \subseteq S_h^\sigma$ be the chain covered by σ for some $h \geq 1$. We call $S_i^\sigma \subseteq S_{i+1}^\sigma$ a segment. Since the $k-d$ permutations partition k sets into chains, there are exactly d segments in all chains together. We use these segments to define a flow of value d in $\mathbb{G}$ which leads to the desired contradiction.

Each segment corresponds to a unique augmenting path $s \to u_i \to v_j \to t$ in the flow network $\mathbb{G}$. So we can construct d augmenting paths in $\mathbb{G}$ which leads to a flow of value d. □

From the two claims, we conclude that we need at least $k-m$ permutations to cover all subsets where m is the maximum flow of $\mathbb{G}$ which implies that there is a valid solution if $\ell \geq k-m$.

Time complexity: We have k subsets. The construction of $\mathbb{G}$ takes $O(k^2)$. It takes $O(k^3)$ time to run the Preflow-Push algorithm on $\mathbb{G}$ which leads to a total time complexity of $O(k^3)$.

Problem 4: Maximum likelihood points of failure

A network is described as a directed graph (not necessarily acyclic), with a specified *source* s and *destination* t. A set of nodes (not including s or t) is a *failure point* if deleting those nodes disconnects t from s. For each node of the graph, i, a *failure probability* $0 \leq p_i \leq 1$ is given. It is assumed that nodes fail independently, so the failure probability for a set $F \subseteq V$ is $\prod_{i \in F} p_i$. Give an algorithm which, given G and p_i for $i \in V$, finds the failure point F with the maximum failure probability.

Solution: Maximum likelihood points of failure

Flow network: Let $G = (V, E)$ be the input graph. We construct a flow network $\mathbb{G} = (\mathbb{V}, \mathbb{E}, c)$. The vertex set $\mathbb{V} = \{s', t'\} \cup V_{\text{in}} \cup V_{\text{out}}$ consists of the following nodes:

- s' and t' where s' is the source node and t' is the sink node.
- $V_{\text{in}} = \{v_{\text{in}} |\ v \in V - \{s, t\}\}$ and $V_{\text{out}} = \{v_{\text{out}} |\ v \in V - \{s, t\}\}$

The edges in $\mathbb{E}$ are as follows:

- For every $(s, v) \in E$, we have a directed edge (s', v_{in}) with capacity ∞.
- For every $(v, t) \in E$, we have a directed edge (v_{out}, t') with capacity ∞.
- For every other $(u, v) \in E$, we have a directed edge $(u_{\text{out}}, v_{\text{in}})$ with capacity ∞.
- For every $v \in V$ other than s and t there is a directed edge $(v_{\text{in}}, v_{\text{out}})$ with capacity $-\log p_v$.

Output: Compute a minimum cut of $\mathbb{G}$ using Preflow-Push maximum flow algorithm. Let (A, B) be a minimum cut of $\mathbb{G}$ determined by the algorithm. Define $F = \{v |\ v_{\text{in}} \in A \text{ and } v_{\text{out}} \in B\}$. F is exactly the set of vertices v in G corresponding to the cut edges in $\mathbb{G}$ of the form $(v_{\text{in}}, v_{\text{out}})$. We output F as the failure point with maximum failure probability.

Correctness: We claim that there is a one-to-one correspondence between finite cuts in the flow network and points of failure in the input graph. From the claim, we conclude that a minimum cut would correspond to a failure point with maximum failure probability.

We first establish the following correspondence between G and $\mathbb{G}$.

Claim 0.6. *There is a one-one correspondence between s-t paths in G involving the vertex u and s'-t' paths in $\mathbb{G}$ involving the edge $(u_{\text{in}}, u_{\text{out}})$. More precisely, there is an s-t path in G which involves the vertex u if and only if there is an s'-t' path in $\mathbb{G}$ that involves the edge $(u_{\text{in}}, u_{\text{out}})$.*

Proof. Students are asked to provide the proof of the claim. □

Claim 0.7. *There is a cut in the flow network with capacity c if and only if there is a point of failure in the graph with probability 2^{-c}.*

Proof. Consider any cut with capacity c. Since the cut capacity is finite, all cut edges are of the form $(v_{\text{in}}, v_{\text{out}})$ for some $v \in V$. Let $F \subseteq V$ be the set of vertices v such that the corresponding edge $(v_{\text{in}}, v_{\text{out}})$ is a cut edge. We claim that F is a point of failure with probability 2^{-c}.

To argue that F is a point of failure, we notice that removing all cut edges of the form $(v_{\text{in}}, v_{\text{out}})$ in the flow network disconnects s' from t' (by the definition of a cut). However, by the construction of $\mathbb{G}$, it is easy to see that removing F from G would disconnect s from t in G.

The capacity c of the cut (A, B) is given by

$$\begin{aligned} c &= \sum_{v \in F} -\log p_v \\ &= -\log \left(\prod_{v \in F} p_v \right) \end{aligned}$$

Hence $\prod_{v \in F} p_v = 2^{-c}$.

For the other direction let $F \subseteq V$ be a point of failure with probability p. If we delete the vertices in F, G will not have any s-t paths. Let $F_{\text{in}} = \{v_{\text{in}} \in \mathbb{V} |\ v \in F\}$ and $F_{\text{out}} = \{v_{\text{out}} \in \mathbb{V} |\ v \in F\}$. We define

$$\begin{aligned} A &= \{s'\} \cup V_{\text{in}} \cup (V_{\text{out}} - F_{\text{out}}) \\ B &= \mathbb{V} - A \end{aligned}$$

We argue that (A, B) is a finite capacity cut for $\mathbb{G}$ with cut edges $\{(v_{\text{in}}, v_{\text{out}}) |\ v \in F\}$ (explain why).

The capacity of the cut is $\sum_{v \in F} -\log p_v = -\log \left(\prod_{v \in F} p_v \right) = -\log p$. □

Since the function 2^{-c} is monotonically decreasing with c we conclude that the maximum likelihood point of failure corresponds to the minimum cut.

Complexity analysis: Since we have $|\mathbb{V}| = O(|V|)$ we can find the minimum cut in time $O(|V|^3)$ using the Preflow-Push algorithm. Note that we have irrational capacities, hence Ford-Fulkerson is not be guaranteed to terminate. It is therefore important that we chose an algorithm with a runtime independent of the capacities.