CSCI 570 Fall 2025

Homework 9

Due: April 11

Note: In the following questions we assume that the Ford-Fulkerson algorithm is provided. You do not need to rewrite the Ford-Fulkerson algorithm.

Question 1

Given a weighted, directed graph $G = (V, E)$, determine whether there exist k paths such that each path starts at a unique vertex in the set

$$A = \{a_1, a_2, \ldots, a_k\} \subset V,$$

and ends at a unique vertex in the set

$$B = \{b_1, b_2, \ldots, b_k\} \subset V,$$

with no two paths sharing any vertices or edges. (Note that $B \cap A = \emptyset$.)

Task: Design an algorithm to solve this problem and prove its correctness.

(Hint: The proof will have two directions. 1)Forward Direction: If there exist k paths, then [statement]. 2) Backward Direction: If [statement] then there exist k paths.)

Solution 1

Flow Network Construction.

1. *Vertex splitting*: For each $v \in V$, create v_{in} and v_{out} with an edge $(v_{\text{in}}, v_{\text{out}})$ of capacity 1.
2. *Edges*: For each $(u, v) \in E$, add an edge $(u_{\text{out}}, v_{\text{in}})$ of capacity 1.
3. *Super-source & super-sink*: Add a new source s with edges $(s, a_{i,\text{in}})$ of capacity 1 for each $a_i \in A$, and a new sink t with edges $(b_{j,\text{out}}, t)$ of capacity 1 for each $b_j \in B$.

Claim. There are k vertex- and edge-disjoint paths from A to B in G if and only if the maximum flow in the constructed network is k.

Proof (Forward Direction). If there exist k disjoint paths in G, send 1 unit of flow along each path in the transformed network. Because the paths do not share vertices or edges, no capacity exceeds 1. Hence, the total flow is at least k, and by capacity from s to each a_i and from each b_j to t, it is at most k. Thus, the maximum flow equals k.

Proof (Backward Direction). If the maximum flow is k, by flow decomposition we obtain k unit flows from s to t. Each $(v_{\text{in}}, v_{\text{out}})$ has capacity 1, ensuring no two flows share the same vertex. Likewise, original edges also have capacity 1, so no two flows share the same edge. Thus, we get k vertex- and edge-disjoint paths in G from A to B.

Rubrics 1 (15 pts)

- **(9 pts) Correct Construction of the Flow Network**
  - (3 pts) *Vertex Splitting:* Clearly describe splitting each vertex v into v_{in} and v_{out} with an edge $(v_{\text{in}}, v_{\text{out}})$ of capacity 1.
  - (3 pts) *Edge Transformation:* For every original edge $(u, v) \in E$, correctly add an edge $(u_{\text{out}}, v_{\text{in}})$ with capacity 1.
  - (3 pts) *Super-Source and Super-Sink:* Properly introduce a super-source s and super-sink t, with edges $(s, a_{i,\text{in}})$ and $(b_{j,\text{out}}, t)$ of capacity 1.
- **(6 pts) Accurate Proof of Correctness**
  - (3 pts) *Forward Direction:* Show that if there are k disjoint paths, one can send 1 unit of flow along each path to achieve a flow of value k.
  - (3 pts) *Backward Direction:* Prove that a max flow of k decomposes into k unit flows, each using distinct edges and vertices, hence yielding k disjoint paths.

Question 2

We have n traders, each possessing W_i Swiss Francs (for $i = 1, \ldots, n$). They wish to convert their Francs into m different currencies $c_1, \ldots, c_m$. However:

- Each currency c_j has a maximum conversion limit B_j, meaning the bank can convert at most B_j Francs into currency c_j (for $j = 1, \ldots, m$).
- Each trader i can convert at most $S_{i,j}$ Francs into currency c_j.

The question is: *Is it possible to convert all of the traders' Francs, i.e.,* $\sum_{i=1}^{n} W_i$*, subject to the given constraints?*

Design an algorithm to decide feasibility by formulating and solving a **maximum flow** problem. Clearly describe the construction of the flow network: specify all vertices, edges, and capacities, and explain how each constraint in the original problem is modeled within this network.

Solution 2

Flow Network Construction: We solve this by constructing a flow network and then running a maximum flow algorithm (e.g. Ford–Fulkerson):

1. **Vertices:**
   - A source vertex s.
   - One vertex t_i for each trader i ($1 \le i \le n$).
   - One vertex c_j for each currency j ($1 \le j \le m$).
   - A sink vertex τ.
2. **Edges and Capacities:**
   - An edge $(s \to t_i)$ with capacity W_i. This ensures that trader i cannot send more than W_i Francs into the network.
   - An edge $(t_i \to c_j)$ with capacity $S_{i,j}$. This captures the limit of how many Francs trader i can convert into currency c_j.
   - An edge $(c_j \to \tau)$ with capacity B_j. This enforces the bank's limit of converting at most B_j Francs into currency c_j.
3. **Algorithm:** Compute the maximum flow from s to τ. If the maximum flow value $|f|$ equals the total amount of Francs $\sum_{i=1}^{n} W_i$, then there exists a way to convert all the traders' Francs subject to the given constraints. Otherwise, it is impossible to do so.

Rubrics 2 (17 Points)

- **Vertices (4 points):** One point each for correctly including the vertices s, t, c, and τ.
- **Edges and Capacities (9 points):** There are three categories of edges. Each category is worth 3 points: 1 point for a correct construction/description, and 2 points for clearly explaining the rationale behind the capacity assignments involving W_i, $S_{i,j}$, and B_j.
- **Algorithm (4 points):** A clear and explicit statement that the bank can satisfy all conversion requests if and only if the maximum flow in the constructed network equals $\sum_{i=1}^{n} W_i$, along with a brief justification.

Question 3

We have m disjoint time intervals. There are n guards, and each guard is *available* for some subset of these intervals (i.e., the guard can only be assigned to intervals in which they are available). We need to assign guards so that:

- Each interval is covered by at least one and at most two guards.
- Each guard is deployed to *at most* Q intervals.
- Each guard is deployed to *at least* L intervals.

The goal is to decide whether there exists a valid assignment of guards to intervals satisfying all these conditions.

Task: Design an algorithm to decide feasibility by formulating and solving a maximum flow problem.
Clearly describe the construction of the flow graph: specify all vertices, edges, capacities, and constraints, and explain how each constraint in the original problem is modeled within this network. Reduce the problem to a standard maximum flow problem without lower bounds by introducing new nodes where necessary.

Solution 3

This is similar to a survey design problem. First, we design a circulation problem where all nodes have zero demand, but we have edge lower bounds.

Vertices

- A **source** vertex s.
- A **sink** vertex t.
- **Guard nodes:** For each guard i, create a vertex denoted by g_i, for $i = 1, \ldots, n$.
- **Interval nodes:** For each interval j, create a vertex denoted by I_j, for $j = 1, \ldots, m$.

Edges

1. **Edges from s to each guard g_i:**
   - **Lower bound:** L (ensuring guard i is assigned to at least L intervals).
   - **Capacity:** Q (ensuring guard i is assigned to at most Q intervals).
2. **Edges from a guard g_i to an interval I_j:**
   - An edge (g_i, I_j) is added if and only if guard i is available during interval j.
   - **Lower bound:** 0.
   - **Capacity:** 1, since a guard can cover an interval at most once.
3. **Edges from each interval I_j to t:**
   - **Lower bound:** 1 (ensuring that each interval is covered by at least one guard).
   - **Capacity:** 2 (ensuring that each interval is covered by at most two guards).
4. **Edge from t to s:**
   - **Lower bound:** 0
   - **Capacity:** ∞

Transforming the problem: Removing the lower bounds

1. **For each edge** $e = (u, v)$ with lower bound $l(e)$ and capacity $c(e)$, define a new capacity push flow $f_0(e)$ through the graph where $f_0(e) = l(e)$.

2. **For each edge** $e = (u, v)$ with lower bound $l(e)$ and capacity $c(e)$, define a new capacity:
$$c'(e) = c(e) - l(e).$$

3. **Adjust Node Balances:**[1] For every vertex v, define its imbalance as
$$b(v) = \sum_{\text{edges } e \text{ into } v} l(e) - \sum_{\text{edges } e \text{ out of } v} l(e).$$

   and update the demands to:
$$d'(v) = d(v) - b(v)$$

   In our network:

   - For the source s, it has n outgoing edges each with a lower bound L, so
$$b(s) = 0 - nL = -nL.$$
     So
$$d'(s) = nL$$

   - For each guard node g_i, since it has an incoming edge from s with lower bound L and outgoing edges with zero lower bound,
$$b(g_i) = L - 0 = L.$$
     So
$$d'(g_i) = -L$$

   - For each interval node I_j, it has an outgoing edge to t with lower bound 1 and incoming edges (from guards) with zero lower bound,
$$b(I_j) = 0 - 1 = -1.$$
     So
$$d'(I_j) = 1$$

   - For the sink t, it has m incoming edges each with a lower bound 1, so
$$b(t) = m - 0 = m.$$
     So
$$d'(t) = -m.$$

[1] This is denoted as $L(v)$ instead of $b(v)$ in the textbook, but we use this different notation here to avoid confusion with the lower bound parameter L.

Transforming the problem to Network flow

Now we convert this circulation problem (with no lower bounds) to a max flow problem:

1. **Introducing a Super Source and Super Sink:** Create a **super source** S and a **super sink** T. Then, for each vertex v:

    - If $d'(v) < 0$, add an edge (S, v) with capacity $d'(v)$.
    - If $d'(v) > 0$, add an edge (v, T) with capacity $|d'(v)|$.

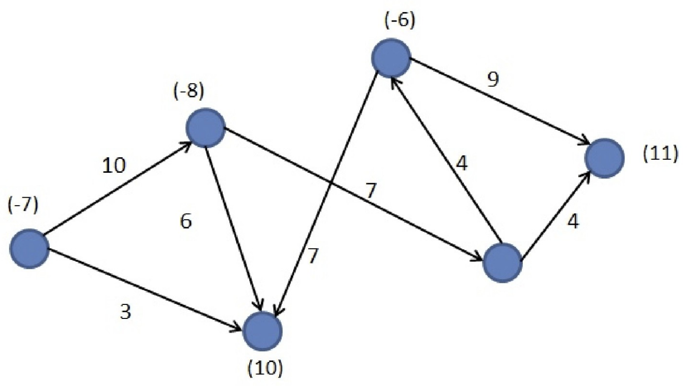
    In our network, this means:

    - For each guard node g_i, since $d'(g_i) = -L$, add edge (S, g_i) with capacity L.
    - For the sink t, since $d'(t) = -m$, add edge (S, t) with capacity m.
    - For the source s, since $d'(s) = nL$, add edge (s, T) with capacity nL.
    - For each interval node I_j, since $d'(I_j) = 1$, add edge (I_j, T) with capacity 1.

2. **Solve the Maximum Flow:** Solve the standard maximum flow problem from S to T. If the maximum flow saturates all edges emanating from S (or, equivalently, if the total flow equals the sum of the positive imbalances which is $m + nL$), then there exists a feasible circulation in the original network.

Question 4

Graph G is an instance of a circulation problem with demands. The edge weights indicate capacities, and the node weights (shown in parentheses) indicate demands. A negative demand indicates that a node acts as a source.



- Transform this graph into an instance of max-flow problem. *Just draw the the new graph.*
- Now, assume that each edge of G has a constraint of lower bound of 1 unit, i.e., one unit must flow along all edges. Find the new instance of max-flow problem that includes the lower bound constraint. *Just draw the the new graph.*

Solution 4

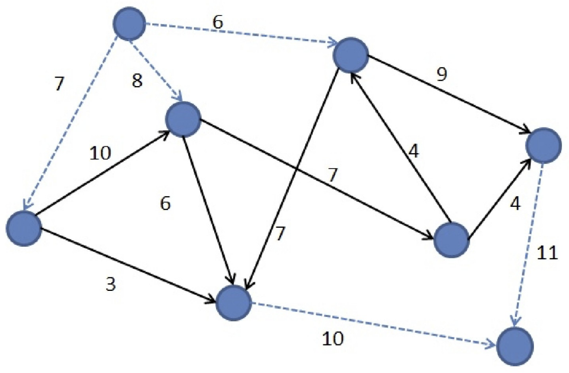


Figure 1: First part

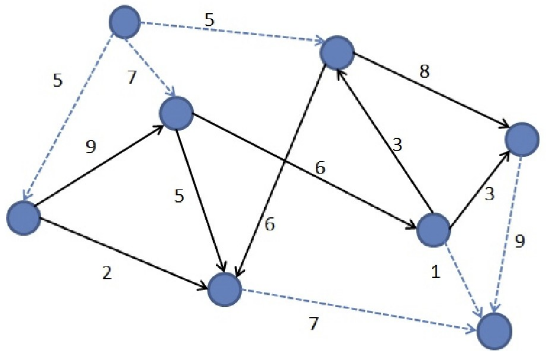


Figure 2: Second part

Rubrics (6 points)

Each graph: 3 points

Ungraded 1

Kleinberg and Tardos, Chapter 7, Exercise 9

Solution of Ungraded 1

We build the following flow network:

- For each patient i, create a node v_i.
- For each hospital j, create a node w_j.
- If patient i is within a half-hour drive of hospital j, add an edge (v_i, w_j) with capacity 1.
- Add a super-source s and connect it to each v_i with an edge of capacity 1.
- Add a super-sink t and connect each w_j to t with an edge of capacity $\lceil \frac{n}{k} \rceil$.

We claim that there is a feasible way to assign patients to hospitals if and only if there is an s–t flow of value n.

(⇒) Feasibility implies flow: If there is a feasible way to assign patients, then for each patient i assigned to hospital j, send one unit of flow along the path

$$s \;\to\; v_i \;\to\; w_j \;\to\; t.$$

Because each patient can be sent to only one hospital, we use at most one unit of flow per patient node. The edges (v_i, w_j) have capacity 1, so they are not exceeded. The edges (w_j, t) have capacity $\lceil \frac{n}{k} \rceil$, which ensures that no hospital j can exceed its maximum load. Thus, this construction yields an integer flow of value n.

(⇐) Flow implies feasibility: Conversely, if there exists an s–t flow of value n, then by the integrality property of maximum flows, there is an integral flow of the same value. In this integral flow, whenever (v_i, w_j) carries one unit of flow, we interpret it as assigning patient i to hospital j. Since the flow respects the capacities, no hospital is assigned more than $\lceil \frac{n}{k} \rceil$ patients, so no capacity constraints are violated.

Running time: The time complexity is dominated by solving a max-flow problem on a graph with $O(n + k)$ nodes and $O(nk)$ edges. Using a standard max-flow algorithm (e.g., Edmond–Karp, Dinic, Push–Relabel), the running time depends on these parameters but is polynomial in n and k.

Ungraded 2

Counter Espionage Academy instructors have designed the following problem to see how well trainees can detect SPY's in an $n \times n$ grid of letters S, P, and Y. Trainees are instructed to detect as many disjoint copies of the word SPY as possible in the given grid. To form the word SPY in the grid they can start at any S, move to a neighboring P, then move to a neighboring Y. (They can move north, east, south or west to get to a neighbor.) The following figure shows one such problem on the left, along with two possible optimal solutions with three SPY's each on the right.

Give an efficient network flow-based algorithm to find the largest number of SPY's.

Note: We are only looking for the largest **number** of SPYs not the actual location of the words. No proof is necessary.

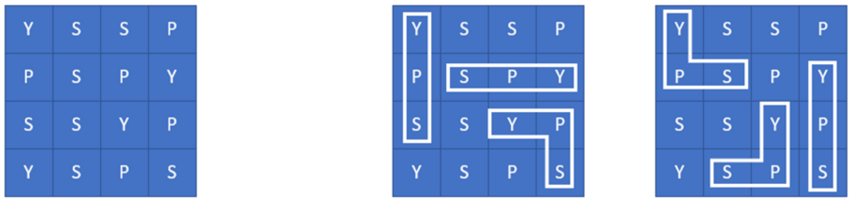


Figure 3: SPY detection problem and two optimal solutions

Solution to Ungraded 2

We construct a flow graph as follows, with all edges having capacity 1:

1. Create one layer of nodes for all the S's in the grid. Connect this layer to a source vertex s, directing edges from s to each S.

2. Create two layers of nodes for all the P's in the grid. Connect the first layer to the S's based on whether or not they are adjacent in the grid, directing edges from S to P. Also, connect the first layer to the second layer by connecting the nodes that correspond to the same location. This is similar to the vertex capacities problem.

3. Create a layer of nodes for all the Y's in the grid. Connect these to a sink vertex t, directing edges from Y to t. Also, connect them to the second layer of P's based if the P and Y are adjacent in the grid, directing edges from P to Y.

Claim: Maximum number of disjoint SPYs we can make corresponds to the value of max flow in this construction.