Assignment #2

Due: July 17, 2021, 11:59 pm (EDT)

SIWEI TANG(tangsiw3)
Yuewei Wang (wangyuew)

Q1:

a) Define a flow network where $u_1 ... u_n$ are all *n* days and $v_1 ... v_k$ are all doctors. We fix the network by adding *s* and *t* as source and sink respectively. Edges in the network are:

- (s, u_i):$c(s, u_i) = a_i$ for $i \in \{1, 2, ..., n\}$
- (v_j, t): $c(v_j, t) = |A_j|$ for $j \in \{1, 2, ..., k\}$
- (u_i, v_j): $c(u_i, v_j) = 1$ if $u_i \in A_j$

1:procedure HospitalSchedule($a_1 ... a_n, A_1 ... A_k$**)**
2: $sum := a_1 + a_2 + ... + a_n$
3: Define a flow network (*G*) with $u_1 ... u_n$, $v_1 ... v_k$, *s*, *t* and all edges.
4: *f* := **Edmonds-Karp**(*G, s, t*) *# to obtain the max-flow*
5: **if** |*f*| < *sum* **then**
6: **return** "no schedule is possible given the doctors' availabilities"
7: **else**
8: **for** *j* = 1:*k* **do**
9: $S_j := \{\}$
10: **for** *i*= 1:*n* **do**
11: **if** $f(u_i, v_j) = 1$ **then**
12: $S_j = S_j \cup \{v_j\}$
13: **return** $S_1, ..., S_k$

b) **Correctness:** In *G* we defined, for the number of doctors that can be scheduled on each day, $0 \le f(s, u_i) \le c(s, u_i) = a_i$ for $i \in \{1, 2, ..., n\}$ due to the capacity constraint. Similar to the number of days that each doctor is scheduled, $0 \le f(v_j, t) \le c(v_j, t) = |A_j|$ for $j \in \{1, 2, ..., k\}$. Each flow between u_i to v_j will be $0 \le f(u_i, v_j) \le c(u_i, v_j) = 1$. There are two cases:

- $f(u_i, v_j) = 0$: The doctor *j* is not scheduled on day *i*.
- $f(u_i, v_j) = 1$: The doctor *j* is scheduled on day *i*.

Since exactly a_i doctors should work on day *i*, a valid flow *f* in G must satisfy $|f| = \sum_{i=1}^{n} a_i$ so that generating a valid schedule $S_1, ..., S_k$. Edmonds-Karp algorithm gives the max-flow, line 5 checks the criteria: if the value of flow is less than the sum, then no valid schedule exists.

S_j will only schedule available days for each doctor (i.e. $f(u_i, v_j) = 1$ for $j \in \{1, 2, ..., k\}$). Line 8 to 12 iteratively checks the flow value for each doctor, which satisfies the criteria.

Runtime: Finding the sum takes $O(n)$. Defining all vertices in the network takes $O(n + k + 2)$. All edges take $O(n + k + \sum_{i=1}^{k} |A_i|)$, which is $O(nk + n + k)$ since $\sum_{i=1}^{k} |A_i|$ will no more than nk. Calling Edmonds-Karp algorithm takes $O((n + k + 2)(nk + n + k)^2)$. The nested loop in line 8 to 12 takes $O(nk)$. Therefore, the total complexity is $O((n + k + 2)(nk + n + k)^2)$.

c) Modify the flow network by adding vertices of $v'_1 ... 'v_k$ for c days such that:

- (v'_j, t): $c(v'_j, t) = c$ for $j \in \{1, 2, \dots, k\}$
- (u_i, v'_j): $c(u_i, v'_j) = 1$ if $u_i \notin A_j$

1:procedure ScheduleDays(c, $a_1 ... a_n$, $A_1 ... A_k$)
2: *sum*:= $a_1 + a_2 + ... + a_n$
3: Define a flow network (G) with $u_1 ... u_n$, $v_1 ... v_k$, $v'_1 ... v'_k$, s, t and all edges.
4: f := **Edmonds-Karp**(G, s, t) *# to obtain the max-flow*
5: **if** $|f| <$ *sum* **then**
6: **return** "no schedule is possible given the doctors' availabilities"
7: **else**
8: **for** j = 1:k **do**
9: $S_j := \{\}$
10: **for** i= 1:n **do**
11: **if** $f(u_i, v_j) = 1$ **then**
12: $S_j = S_j \cup \{v_j\}$
13: **else if** $f(u_i, v'_j) = 1$ **then**
14: $S_j = S_j \cup \{v'_j\}$
15: **return** $S_1, \dots, S_k$

d) **Correctness:** Similar as the proof in b), but there are additional flows associated with $v'_1 ... 'v_k$. $f(v'_j, t)$ represents the number of days that are not in A_j but doctor j is scheduled. Since at most c days could be not in A_j, $0 \leq f(v'_j, t) \leq c(v'_j, t) = c$ for $j \in \{1, 2, \dots, k\}$. Similar as $f(u_i, v_j)$, each flow between u_i to v'_j is that $0 \leq f(u_i, v'_j) \leq c(u_i, v'_j) = 1$. There are also two cases such that 1 means the doctor j is scheduled on day i and 0 is not. Similar as b), we check $|f|$ with $\sum_{i=1}^{n} a_i$ since exactly a_i doctors work on day i. For $j \in \{1, 2, \dots, k\}$, S_j will schedule when $f(u_i, v_j) = 1$ but also when $f(u_i, v'_j) = 1$ since at most c days not in A_j can be scheduled. Both conditions are checked in line 11 to 14.

Runtime: The algorithm is modified from a) with difference in line 13 to 14, which takes constant steps to check the condition. Therefore, the total complexity is still $O((n + k + 2)(nk + n + k)^2)$.

Q2:

By Max-Flow-Min-Cut Theorem, if f^* is a maximum flow and (S, T) is the minimum cut, then $|f| = c(S, T)$. Also, we can find that for the minimum cut (S, T), the edge (u, v) where $u \in S$ and $v \in T$ is a critical edge. So we can transfer the problem into finding the edge that crosses the minimum cut (S, T).

Algorithm:

1:procedure FindCriticalEdge(G, s, t**)**
2: (f_m, G_{f_m}):= **Edmonds-Karp**(G, s, t) *# slightly modify to obtain the max-flow and the residual graph from Edmonds-Karp*
3: Find the minimum cut (S, T): $S = \{v \in V$ | there is a path from s to v in $G_{f_m}\}$

$$T = V \backslash S$$

4: **for** each edge $(u, v) \in E$ **do**
5: **if** $u \in S$ and $v \in T$ **then**
6: $e_c := (u, v)$
7: **if** e_c exists **then**
8: **return** e_c
9: **else**
10: **return** "no critical edge exists"

Proof of Correctness:

Since Edmonds-Karp algorithm gives the max-flow, it finds the augmenting paths by a breadth-first search in G, so it chooses a shortest path in G from s o t where each edge has weight 1.

By definition of cut, there exists at least one edge between cut S and T. For every vertice $x \in S$, there is a path from S to T including x. By trace this path, we can find an edge (u, v) such that $u \in S$ and $v \in T$.

Running time:

For line 2, by Edmonds-Karp Algorithm, the complexity is $O(|V||E|^2)$.
For line 3, use breadth first search, so it takes $O(|E|)$.
For line 4 to 6, check every edge if $u \in S$ and $v \in T$, if it satisfies these two requirements, then return the result. It takes $O(|E|)$.
So in total $O(|V||E|^2) + O(|E|) + O(|E|) = O(|V||E|^2)$

Q3:

a)

In directed graph G(V, E), for two different vertices s and t that in G, the pseudo-flow is the flow function such that all vertices u in G that is not s and t,

$\sum_{x \in V} f(x, u) = \sum_{x \in V} f(u, x)$.

By question, if a pseudo-flow f from s to t such that |f| = 1, it means that the inflow and outflow of vertice s is 1,since all edges is positive integer, so that means that only one edge that connects to s with f(s, u) = 1, for all other vertices u' that directly connect to s, f(s, u') = 0. By that logic, we can find list of connected vertices $s, u_1, u_2, u_3 \ldots$, t such that f(s, u_1) = f(u_1, u_2) = …, for all other edges, f is 0.

So every path s, s, u_1, u_2, u_3,...,t is pseudo-flow with |f| = 1 for every edge in the path and |f| = 0 for every edge out of the edge.In such a case, finding the shortest path is equivalent to finding the pseudo-flow.

b)

Let $f(u, v)$be the variables that edge$(u, v) \in E$ in graph G, which represents the pseudo-flow. Then we will solve in LPP as:

$$\text{minimize: } \sum_{(u,v)\in E} f_{(u,v)} l_{(u,v)}$$

$$\text{subject to: } \sum_{(u,v)\in E} f_{(u,v)} = \sum_{(v,u)\in E} f_{(v,u)} \text{ for } v \in V \backslash \{s, t\}$$

$$\sum_{(s,v)\in E} f_{(s,v)} = 1$$

$$f_{(s,v)} \geq 0$$

Justification:

For the three constraints, $\sum_{(u,v)\in E} f_{(u,v)} = \sum_{(v,u)\in E} f_{(v,u)}$ since the pseudo-flow is conserved. $\sum_{(s,v)\in E} f_{(s,v)} = 1$since pseudo-flow with $|f|$ = 1 is the target looking for. Also, $f_{(s,v)} \geq 0$since the pseudo-flow value is non-negative.

By part a), finding paths from s to t in G is equivalent to pseudo-flow from s to t with $|f|$ = 1. The possible solutions are the same as finding the pseudo-flow f from s to t with |f| = 1. In such a case, the solutions is the same as finding paths from s to t such that

$\sum_{(u, v)\in E} f_{(u, v)} l_{(u, v)}$ is equivalent to the total weight of the path. So in order to minimize $\sum_{(u, v)\in E} f_{(u, v)} l_{(u, v)}$. It is the same as minimizing the weight of the path. Therefore, we need to find the shortest path so that it has the minimum weight.

Q4:

a)

Let e_i represents the new variable that l_1 error of data point $((x_i, y_i)$for $i = 1, 2,..., n$. Let a and b also be variables of LPP, then we will solve$e_i$as:

$$\text{minimize: } \sum_{i=1}^{n} e_i$$

$$\text{subject to: } e_i = |y_i - ax_i - b| \text{ for } i = 1, 2,\dots, n$$

By the equivalent relationship, let e_i in terms of inequalities that

$$e_i = |y_i - ax_i - b| \Leftrightarrow e_i \geq (y_i - ax_i - b) \text{ and } e_i \geq - (y_i - ax_i - b)$$

Then $e_i = max\{(y_i - ax_i - b), \ - (y_i - ax_i - b)\}$

$$e_i \geq max\{(y_i - ax_i - b), \ - (y_i - ax_i - b)\}$$

By the definition of optimal solution in LPP, e_i has the minimum objective value that satisfies the optimal solution in minimization LPP.

b)

By the requirements of the separating line maximizes the gap δ, $e_1 \leq |y_i - ax_i - b|$ for $i = 1, 2,\dots, m$ in type 1 points and $e_2 \leq |y_j - ax_j - b|$for $j = 1, 2,\dots, n$ in type 2 points.

Since $y_i < ax_i + b$ for type 1, $ax_i + b - y_i > 0,$ then $|y_i - ax_i - b| = ax_i + b - y_i$. Similarly, $|y_j - ax_j - b| = y_j - ax_j - b$ since $y_i > ax_i + b$ for type 2. Then:

$$e_1 \leq ax_i + b - y_i \text{for } i = 1, 2,\dots, m \text{ and } e_2 \leq y_j - ax_j - b \text{for } j = 1, 2,\dots, n$$

Since the gap $\delta = min\{e_1, e_2\}$, in terms of inequalities:

$\delta \leq ax_i + b - y_i$for $i = 1, 2,\dots, m$ and $\delta \leq y_j - ax_j - b$ for $j = 1, 2,\dots, n.$

Then we will solve the LPP as:

$$\text{maximize: } \delta$$

$$\text{subject to: } \delta \leq ax_i + b - y_i \text{for } i = 1, 2,\dots, m \text{ in type 1 points}$$

$$\delta \leq y_j - ax_j - b \text{ for } j = 1, 2,\dots, n \text{ in type 2 points}$$

By the definition of optimal solution in LPP, δ has the maximum objective value that satisfies the optimal solution in maximization LPP. Note that the optimalδis positive since

$ax_i + b - y_i > 0$ for all type 1 points and $y_j - ax_j - b > 0$ for all type 2 points. There are two cases if the separating line $y = ax + b$ exists:

case 1: $y = ax + b$ is a vertical line

then $a = 0$and $x = b$is the line.b will be in the middle of the barrier of type 1 and type 2 points (i.e. $x_1$is the largest x-coordinate among all type 1 points and $x_2$is the smallest x-coordinate among all type 2 points, b is in between two values). Then the gap δwill be the difference to each x-coordinate.

case 2: $y = ax + b$ is not vertical

subcase 1: the line satisfy that $y_i > ax_j + b$for all type 1 points and $y_i < ax_j + b$ for all type 2 points, which is the goal we want.

subcase 2: the line separates both types but with $y_i < ax_j + b$ for all type 1 points and $y_i > ax_j + b$ for all type 2 points, the optimal δexist. Then, the original type 1 will be type 2 and vice versa.