

## Problem 1

1. Given an algorithm taking inputs
  - (a)  $G = (V, E)$  a connected, undirected graph
  - (b)  $w : E \rightarrow \mathbb{Z}^+$  a weight function
  - (c)  $T \subseteq E$ , a MST of  $G$
  - (d)  $e_1 = \{u, v\} \notin E$ , an edge not in  $G$
  - (e)  $w_1 \in \mathbb{Z}^+$ , a weight for  $e_1$

and outputs a MST  $T_1$  for  $G_1 = (V, E \cup \{e_1\})$  with  $w(e_1) = w_1$ . For full marks, your algorithm must be more efficient than computing a MST for  $G_1$  from scratch. Justify that this is the case by analysing your algorithm's worst-case running time. Finally, write a detailed proof that your algorithm is correct.

*Solution.*

□

### Algorithm

- (a) Let  $G_1 = (V, E \cup \{e_1\})$ , let  $e_1 = (s, t)$  for some  $s, t \in V$
- (b) Find the unique simple cycle  $c$  in  $G_1$ . Starting from  $s$  (or  $t$ , the choice is arbitrary), run DFS on  $G_1$  with slight modification. When looking at vertex  $u \in V$ , Check if  $u.color$  is *GRAY*. Break from DFS if true, otherwise continue DFS.
- (c) Find the largest weight edge  $e_2$  in the cycle. Iterate through  $G_1.E$ , find  $e_2 = (u, v) \in G_1.E$  such that  $u.color$  and  $v.color$  are both *GRAY* (in the cycle) and  $w(u, v)$  is maximum of all edges with *GRAY* vertices (max weighted edge).
- (d) Return MST  $T_1 = T \cup \{e_1\} \setminus \{e_2\}$

### Analysis

- (a) Since we are adding a constant time check at every while iteration and DFS itself has a run time of  $O(V + E)$ , the part of algorithm for finding a cycle (modified DFS) has a worst case running time of  $O(V + E)$
- (b) The loop over all vertices to locate the maximum weight edge runs for  $|E|$  iterations, each time doing a constant time operation. hence has a worst case running time of  $O(E)$
- (c) Altogether the algorithm runs for  $O(V + E)$ , which is more efficient by computing MST from ground up which takes  $O(E \lg V)$

```

1 Function DFS-Visit ( $G, u$ )
2    $u.color \leftarrow GRAY$ 
3   for  $v \in G_1.Adj[u]$  do
4     if  $v.color$  is  $GRAY$  then
5       Exit-DFS
6     if  $v.color$  is  $WHITE$  then
7       DFS-Visit( $G, v$ )
8    $u.color \leftarrow BLACK$ 
9 Function Find-MST-1 ( $G, w, T, e_1, w_1$ )
10   $G_1 \leftarrow (V, E = E \cup \{e_1\})$  with  $w : E \rightarrow \mathbb{R}$  and  $w(e_1) = w_1$ 
11   $(s, t) \leftarrow e_1$ 
12  for  $v \in V$  do
13     $v.color \leftarrow WHITE$ 
14  DFS-Visit( $G_1, s$ )
15   $max_w \leftarrow 0$ 
16   $e_2 \leftarrow NIL$ 
17  for  $(u, v) \in G_1.E$  do
18    if  $u.color = GRAY$  and  $v.color = GRAY$  and  $w(u, v) > max_w$  then
19       $max_w \leftarrow w(u, v)$ 
20       $e_2 \leftarrow (u, v)$ 
21   $T_1 \leftarrow T \cup \{e_1\} \setminus \{e_2\}$  return  $T_1$ 

```

**Lemma.** The graph  $G_1 = (V, E = E \cup \{e_1\})$  has a unique simple cycle  $c$ , and  $e_1$  is in  $c$

*Proof.* Let  $T \subseteq E$ , and  $e_1 = (s, t)$ . Since  $T$  is MST,  $s$  is reachable from  $t$ , i.e. exists a path  $p$  such that  $t \xrightarrow{p} s$ . Consider  $E_1 = E \cup \{e_1\}$ , we have  $c = s \rightarrow t \xrightarrow{p} s = \langle v_0 = s, \dots, v_k = s \rangle$ . Therefore  $e_1$  is in  $c$ . Now we prove  $c$  is unique. This is equivalent to proving that  $p$  is unique, since if adding  $e_1$  to  $E$  produces more than 1 cycle, then there must exists  $p'$  such that  $p' \neq p$ . This is not possible since  $s \xrightarrow{p} t \xrightarrow{p'} s$  forms a cycle, which is not possible in MST  $T$ . Also,  $c$  is simple because  $T$  is simple.  $\square$

### Proof of correctness

**Proposition.** The output  $T_1$  is a MST for  $G_1 = (V, E = E \cup \{e_1\})$

*Proof.* By correctness of DFS, when we first explored  $(u, v)$ , if  $v.color$  is  $GRAY$ , then  $(u, v)$  is a back edge, indicating we have found a cycle  $c$  in the graph. By the previous lemma and the fact that we started from  $s$  where  $e_1 = (s, t)$ , we will always find such a cycle (i.e. discover  $t$  such that  $s \in G_1.Adj[t]$  and  $s.color$  is  $GRAY$ ). When EXIT-DFS is called, the vertices  $v \in V$  such that  $v.color$  is  $GRAY$  represents vertices

constituting the cycle  $c$ . The claim holds since by the time EXIT-DFS is called, all ancestors of  $t$  has yet to finish (i.e. setting their color to *BLACK*). The subsequent loop over  $G_1.E$  finds the maximum weighted edge  $e_2$  in the cycle  $c$ . By the cycle property of MST,  $e_2$  cannot be included in any MST of  $G_1$ . Now consider the return value  $T_1 = T \cup \{e_1\} \setminus \{e_2\}$ . Now we prove  $T_1$  is a MST of  $G_1$ . Consider  $A = T \setminus \{e_2\}$ , note  $A$  breaks into 2 connected components as MST  $T$  is connected. Let  $C = (P, Q)$  be the cut where  $s \in P$  and  $t \in Q$ ,  $P \cup Q = A$  and  $P \cap Q = \emptyset$ . The fact that  $e_2 \in T$  implies that  $e_2$  is a light edge cross the cut  $C$ . Now we have  $w(e_1) < w(e_2)$  since  $e_2$  is the maximum weight edge in the cycle  $c$ , therefore  $e_1$  is the light across the cut  $C$ . By corollary in CLRS, since  $P$  respects the cut  $C$  and  $e_2$  is a light edge crossing the cut and  $P \subseteq T$ , therefore  $e_2$  is safe for  $P$ . Therefore  $T_1$  is some MST of  $G_0$   $\square$

2. Give an efficient algorithm that takes the following inputs:

- (a)  $G = (V, E)$  a connected, undirected graph
- (b)  $w : E \rightarrow \mathbb{Z}^+$  a weight function
- (c)  $T \subseteq E$ , a MST of  $G$
- (d)  $e_0 \in E$ , an edge in  $G$

and that outputs a minimum spanning tree  $T_0$  for the graph  $G_0 = (V, E \setminus \{e_0\})$ , if  $G_0$  is still connected your algorithm should output the special value *Nil* if  $G_0$  is disconnected. For full marks, your algorithm must be more efficient than computing a MST for  $G_0$  from scratch. Justify that this is the case by analysing your algorithms worst-case running time. Finally, write a detailed proof that your algorithm is correct. (Note that this argument of correctness will be worth at least as much as the algorithm itself.)

*Solution.*  $\square$

### Algorithm

- (a) Keep track of the cut  $C = (P, Q)$  such that  $P \cup Q = E \setminus \{e_0\}$ 
  - i. Make set on every  $v \in V$
  - ii. Iterate over  $T \subseteq E$ , combine the sets if there is an edge connecting the two connected components
- (b) Find the set of edges  $E' \subseteq E$  such that for all  $e \in E'$ ,  $e$  crosses the cut  $C$ . This can be achieved by checking all  $e = (u, v) \in E$  against cut  $C$  and see if the  $u$  and  $v$  are in different connected components
- (c) Find the light edge  $e_2$  (lowest weight edge crossing the cut  $C$ ) from the  $E'$
- (d) Return MST  $T_0 = T \setminus \{e_0\} \cup \{e_2\}$  if  $e_2$  exists otherwise return *Nil*

```

1 Function Find-MST-2 ( $G, w, T, e_0$ )
2    $G_0 = (V, E \setminus \{e_0\})$ 
3   for  $v \in V$  do
4     Make-Set( $v$ )
5   for  $(u, v) \in T$  do
6     Union-Set( $u, v$ )
7    $E' \leftarrow \emptyset$ 
8   for  $(u, v) \in G_0.E$  do
9     if Find-Set( $u$ )  $\neq$  Find-Set( $v$ ) then
10       $E' \leftarrow E' \cup \{(u, v)\}$ 
11   if  $E' = \emptyset$  then
12     return Nil
13    $min_w \leftarrow \infty$ 
14    $e_2 \leftarrow \text{Nil}$ 
15   for  $e \in E'$  do
16     if  $w(e) \leq min_w$  then
17        $min_w \leftarrow w(e)$ 
18        $e_2 \leftarrow e$ 
19    $T_0 \leftarrow T \setminus \{e_0\} \cup \{e_2\}$ 
20   return  $T_0$ 

```

### Proof of correctness

**Proposition.** *The algorithm returns  $T_0$ , a MST for  $G_0 = (V, E \setminus \{e_0\})$  if exists, NIL otherwise*

*Proof.* Note  $G = (V, E)$  has MST  $T$  implies  $G$  is connected. The removal of an edge  $e_0$  disconnects  $E$  into 2 connected components. Let  $C = (P, Q)$  be the cut such that  $P \cup Q = E \setminus \{e_0\}$ .  $C = (P, Q)$  is kept track of in sets, where all  $e_1 \in P$  belongs to one set and all  $e_2 \in Q$  belongs to another.  $E'$  hence contains edges  $e = (u, v) \in E$  such that  $u \in P \wedge v \in Q$  or  $v \in P \wedge u \in Q$ . If there is no such edges other than  $e_0$  that crosses the cut  $C$ , then  $E'$  is empty and  $G_0$  is disconnected. In this case the algorithm returns NIL, which is correct. Otherwise we find a light edge  $e_2$  and returns  $T_0 = T \setminus \{e_0\} \cup \{e_2\}$ . Here we claim  $T_0$  is some MST of  $G_0$ . By corollary given in CLRS, since  $P$  respects the cut  $C$  and  $e_2$  is a light edge crossing the cut and  $P \subseteq T$ , therefore  $e_2$  is safe for  $P$ . Therefore  $T_0$  is some MST of  $G_0$   $\square$

### Problem 2

Consider the following MST with Fixed Leaves problem:

1. Input: A weighted graph  $G = (V, E)$  with integer costs  $c(e)$  for all edge  $e \in E$ , and a subset of vertices  $L \subseteq V$

2. Output: A spanning tree  $T$  of  $G$  where every node of  $L$  is a leaf in  $T$  and  $T$  has the minimum total cost among all such spanning trees.
1. Does this problem always have a solution? In other words, are there inputs  $G, L$  for which there is no spanning tree  $T$  that satisfies the requirements? Either provide a counter-example (along with an explanation of why it is a counter-example), or give a detailed argument that there is always some solution.

*Solution.*

□

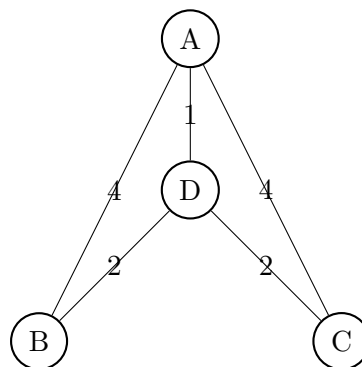
The problem does not always have a solution. Consider  $G = (V, E)$  where  $V = \{1, 2, 3, 4\}$  and  $E = \{(1, 2), (1, 3), (1, 4)\}$  with  $L = \{1\}$  and arbitrary weights. Since there is only 3 edges which happen to connect to all 4 vertices, there is only one MST  $T = E$  for  $G$ . A non-root node in an acyclic graph (tree) implies that there is only one parent. Since vertex 1 connects to 3 other vertices, it cannot be non-root, implying it must be root. Therefore we have a counter example where the only MST possible  $T = E$  do not have  $v \in L$  as leaves.

2. Let  $G, L$  be an input for the MST with Fixed Leaves problem for which there is a solution.
  - (a) Is every MST of  $G$  an optimal solution to the MST with Fixed Leaves problem? Justify.
  - (b) Is every optimal solution to the MST with Fixed Leaves problem necessarily a MST of  $G$  (if we remove the constraint that every node of  $L$  must be a leaf)? Justify.

*Solution.*

□

Both claim are incorrect. Here we provide a counter example. Consider  $G = (V, E)$  where  $V = \{A, B, C, D\}$  and  $E = \{(A, C), (A, B), (D, A), (D, B), (D, C)\}$  and weights labelled in the graph. Let  $L = \{D\}$



There is only one MST  $T = \{(D, A), (D, B), (D, C)\}$  for  $G$  with  $w(T) = 4$ . There is only one MST  $T' = \{(A, C), (A, B), (D, A)\}$  given fixed leaves  $L$  for  $G$  with  $w(T) = 9$ . Note  $T \neq T'$  and  $w(T) < w(T')$ .

- (a) The existence of  $T$ , a MST of  $G$  but not the a solution to MST with fixed leaves problem, disproves this claim
  - (b) The existence of  $T'$ , a MST of  $G$  with fixed leaves problem but not an MST for  $G$ , disproves this claim
3. Write a greedy algorithm to solve the MST with Fixed Leaves problem. Give a detailed pseudo-code implementation of your algorithm, as well as a high-level English description of the main steps in your algorithm. What is the worst-case running time of your algorithm? Justify briefly.

### Algorithm

- (a) Let  $S = V \setminus L$
- (b) Find MST for graph induced by  $S$ , let  $T_S$  be the output MST for  $S$
- (c) For each  $(u, v)$  where  $u \in L$  and  $v \in S$ , adds the lowest weight edge for each  $u$  to  $T_S$ . In other words, find the light edge crossing the cut  $(\{u\}, T_S)$  and adds it to  $T_S$
- (d) Return the augmented  $T$ , consists of  $T_S$  and edges added to it in the previous step

### Analysis

- (a) Assume sets  $S$  and  $T$  membership check can be done in  $O(1)$  time, this is possible if keep an extra bit in adjacency list representation of the graph. Assume insertion to array  $T$  takes  $O(1)$  time.
- (b) MST-KRUSKAL has worst case running time of  $O(E \lg V)$
- (c) The for loop executes  $O(E)$  times altogether, since sum of lengths of all adjacency list is  $2|E|$  for undirected graphs and  $|L| \leq |V|$ . In each iteration, membership check and potential assignment operation takes  $O(1)$ .
- (d) The entire algorithm has a worst time running time of  $O(E \lg V)$

```

1 Function Find-MST-Fixed-Leaves ( $G, w, L$ )
2    $S \leftarrow V \setminus L$ 
3    $G' \leftarrow$  induced by  $S$ 
4    $T \leftarrow \text{MST-Kruskal}(G', w)$ 
5   for  $u \in L$  do
6      $e_{light} \leftarrow \text{Nil}$ 
7      $w_{light} \leftarrow \infty$ 
8     for  $v \in G.\text{Adj}[u]$  do
9       if  $v \in S$  and  $w(u, v) < w_{light}$  then
10          $w_{light} \leftarrow w(u, v)$ 
11          $e_{light} \leftarrow (u, v)$ 
12    $T \leftarrow T \cup \{e_{light}\}$ 
13 return  $T$ 

```

4. Write a detailed proof that your algorithm always produces an optimal solution.

**Lemma.** *Given condition provided by the algorithm, provided that there is a valid solution for fixed leaves problem, then for any  $u \in L$ , there exists some  $v \in S = V \setminus L$  such that  $(u, v) \in E$ .*

*Proof.* Prove by contradiction. Assume there exists  $u \in L$  such that for all  $(u, v) \in E' = E$ , we have  $v \in L$ . Pick any appropriate edge  $(u, v) \in E'$  to be in some valid solution MST  $T$ . Now vertex  $u$  and  $v$  has one edge incident on it. We cannot include any other edge such that they incident on  $u$  or  $v$  since otherwise  $u$  or  $v$  would not be leaves (i.e. have degree of more than 1). Therefore  $\{u, v\}$  is isolated from the rest of the vertices, hence no viable MST exists possibly as a solution. Since we are given there is some solution to the problem, the claim thus holds  $\square$

**Proposition.** *Given  $G = (V, E)$  with cost function  $c : E \rightarrow \mathbb{R}$  and a subset  $L \subseteq V$ , the algorithm returns MST  $T$  where every  $u \in L$  is a leaf and  $T$  has the minimum cost amongst such spanning trees*

*Proof.* By correctness of Kruskal's algorithm, MST-KRUSKAL terminates and returns  $T_S$ , which is a MST for  $S = V \setminus L$ .

- (a) Prove every  $u \in L$  is a leaf in  $T$ . The algorithm finds and adds the least weight edge connecting  $u$  to some  $v \in S$ . Since  $T_S$  spans  $S$  and the fact that the previous lemma holds, such operation is always possible. Since the algorithm only adds one, specifically  $e_{light}$ , to  $T$  for each  $u \in L$ ,  $u$  has degree of one, hence are leaves in MST  $T$
- (b) Prove by contradiction that the algorithm returns a MST amongst such spanning trees. Assume there is some other spanning tree  $T'$  such that  $w(T) > w(T')$ . We

can decompose  $T' = T'_S \cup T'_O$  where  $T'_S$  is a subset of  $T'$  and spans  $S$  and  $T'_O$  be the rest of edges in  $T'$ , specifically  $T'_O$  consists of edges that crosses the cut  $(L, S)$ .

i. By correctness of MST-KRUSKAL we have

$$w(T_S) \leq w(T'_S)$$

ii. After finding  $T_S$ , the algorithm then construct the solution MST  $T$  by adding  $(u, v)$  to  $T_S$  where  $v \in S$  for each  $u \in L$  (Note this is always possible with previous lemma) such that  $w(u, v)$  is minimized. Denote  $E'$  be such set of edges added to  $T_S$  Therefore

$$\sum_{e \in E'} w(e) \leq w(T'_O)$$

Althgether we have

$$w(T) = w(T_S) + \sum_{e \in E'} w(e) \leq w(T'_S) + w(T'_O) = w(T')$$

which contradicts the assumption that  $w(T) > w(T')$ . Therefore the claim holds

This completes the proof □

### Problem 3

An edge in a flow network is called critical if decreasing the capacity of this edge reduces the maximum possible flow in the network. Give an efficient algorithm that finds a critical edge in a network. Give a rigorous argument that your algorithm is correct and analyse its running time.

*Solution.* □

#### Algorithm

1. Find a maximum flow  $f$  for  $G$ , let  $G_f$  be the residual flow network for  $G$  given  $f$
2. Find the transitive closure  $T$  of  $G_f$
3. Iterate over  $E$  find a set of edges  $E' \subseteq E$  such that  $f(u, v) = c(u, v)$  and such that  $u, v$  are not reachable from each other. We can achieve this by evaluating if  $T_{uv}$  is 0
4. Return  $E'$

#### Analysis

1. FORD-FULKERSON has a worst case running time of  $O(VE)$



2. TRANSITIVE-CLOSURE using FLOYD-WARSHALL has a worst case running time of  $O(V^3)$
3. The for loop iterates  $|E|$  times, doing a constant  $O(1)$  operation. Therefore the loop has a worst time running time of  $O(E)$
4. Altogether the algorithm has a worst case running time of  $O(V^3)$

```

1 Function Find-Critical-Edge ( $G, s, t$ )
2    $(f, G_f) \leftarrow \text{Ford-Fulkerson}(G, s, t)$ 
3    $T \leftarrow \text{Transitive-Closure}(G)$ 
4    $E' \leftarrow \emptyset$ 
5   for  $(u, v) \in G.E$  do
6     if  $f(u, v) = c(u, v)$  and  $T_{uv} = 0$  then
7        $E' \leftarrow E' \cup \{(u, v)\}$ 
8   return  $E'$ 

```

**Lemma.** *Given a maximum flow  $f$  for flow network  $G$  and its residual flow network  $G_f$ . If any  $(u, v) \in E$  such that  $f(u, v) = c(u, v)$ , and that  $u$  and  $v$  are not reachable from each other in  $G_f$ , then  $(u, v)$  is a critical edge.*

*Proof.* Note flow  $f(u, v)$  has to be equal to its capacity  $c(u, v)$ , since otherwise a decrease in capacity  $\delta < c(u, v) - f(u, v)$  will not cause any change in flow. Now we prove  $u$  and  $v$  has to be unreachable from each other for  $(u, v)$  to be a critical edge. Prove by contradiction, assume there is a critical edge  $(u, v)$  such that  $f(u, v) = c(u, v)$  and there exists a path  $p$  from  $u$  to  $v$  in  $G_f$ . Consider the arbitrary  $C = (U, V)$  with  $u \in U$  and  $v \in V$ . Since there is a path from  $u$  to  $v$ , at some point along the path, an edge  $(x, y)$  crosses the cut  $C$  where  $x \in U$  and  $y \in V$ . Due to the existence of  $(x, y)$  in  $G_f$ ,  $f(x, y) < c(x, y)$  in  $G$ , in other words, the edge  $(x, y)$  is not saturated to fill its capacity. Therefore  $C$  is not a minimum cut of  $G$ . Since the choice of  $C$  is arbitrary and we will always find an edge  $(x, y)$  that crosses the cut, no cut of  $G$  for which  $(u, v)$  crosses can be the minimum cut. However, by the Max-flow Min-cut theorem,  $|f|$  is bounded by the minimum cut  $C_{min}$ . Therefore, we are able to decrease the capacity of  $(u, v)$  without affecting the max flow  $f$  as long as  $c(C) \geq c(C_{min})$ . Therefore  $(u, v)$  is not a critical edge, which contradicts the assumption. This completes the proof  $\square$

Note the algorithm exactly satisfies the assumption of previous lemma, therefore the returned  $E'$  is a set of critical edges. The algorithm is therefore correct

#### Problem 4

1. Suppose we want to compute the shortest path from node  $s$  to node  $t$  in a directed graph  $G = (V, E)$  with edge lengths  $l_e > 0$  (assume integer valued lengths) for each

$e \in E$ . Show that this is equivalent to finding a pseudo-flow  $f$  from  $s$  to  $t$  in  $G$  such that  $|f| = 1$  and  $\sum_{e \in E} l_e f_e$  is minimized. There are no capacity constraints. Part of this problem requires you to define precisely what we mean by pseudo-flow in a general, directed graph. This is a natural extension of the notion of flow in a network.

*Solution.* □

Define a pseudo-flow as a function  $f : V \times V \rightarrow \mathbb{R}$  such that  $f_{uv}$  represents units of flow from vertex  $u$  to  $v$ .  $f$  still satisfies flow conservation but there is no capacity constraints on the amount of units of flow sent through the network. Given

$$|f| = \sum_{v \in V} f_{sv} = 1$$

we require exactly one unit of flow from the source, and since the graph is connected and by flow conservation, this one unit of flow is transferred conceptually from  $s$  to  $t$  over vertices in some arbitrary path  $p = \langle v_0 = s, v_1, \dots, v_k = t \rangle$  such that  $f_{v_{i-1}v_i} = 1$  for all  $i = 1, \dots, k$ . We claim that

there is not a flow  $f_{uv} > 0$  such that  $(u, v)$  is not in  $p$ .

Prove by contradiction, if there is such a flow  $f_{uv} = 1$  for  $u \rightarrow v$  where  $(u, v)$  is not in  $p$ , then to satisfy flow conservation, there must be some paths  $p_1, p_2$  such that

$$p' = s \xrightarrow{p_1} u \rightarrow v \xrightarrow{p_2} t$$

Since  $(u, v)$  is not in  $p$  by assumption, then  $p' \neq p$ , we then have 2 paths originated from  $s$ , with  $|f| = 2$ , then all  $(u, v)$  with positive flow  $f_{uv} > 0$  is in path  $p$ . Therefore,

$$\sum_{e \in E} l_e f_e = \sum_{e \text{ in } p} l_e \times 1 + \sum_{e \text{ not in } p} l_e \times 0 = \sum_{e \text{ in } p} l_e = w(p)$$

Minimizing  $\sum_{e \in E} l_e f_e$  is equivalent to minimizing  $w(p)$ . Hence, the procedure of finding

a pseudo-flow  $f$  from  $s$  to  $t$  such that  $|f| = 1$  and  $\sum_{e \in E} l_e f_e$  is minimized is equivalent to finding a path  $p$  with minimum weight  $w(p)$ , which is what shortest-path specifies

2. Write the shortest path problem as a linear or integer program where your objective function is minimized, based on your answer to the previous part. Give a detailed justification that your solution is correct.

*Solution.* □

$$\begin{aligned}
\text{Minimize: } & \sum_{e \in E} l_e f_e \\
\text{Subject to: } & \sum_{v \in V} f_{uv} - \sum_{v \in V} f_{vu} = \begin{cases} 1 & \text{if } u = s \\ -1 & \text{if } u = t \\ 0 & \text{otherwise} \end{cases} \\
& f_e \geq 0 \quad \text{for all } e \in E
\end{aligned}$$

The linear program is basically a formal specification of the assumptions made for finding pseudo-flow  $f$  from  $s$  to  $t$  in  $G$ . Since if  $u = s$  we have

$$\sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs} = |f| = 1$$

as well as flow conservation given by

$$\sum_{v \in V} f_{uv} - \sum_{v \in V} f_{vu} = 0 \quad \Longleftrightarrow \quad \sum_{v \in V} f_{uv} = \sum_{v \in V} f_{vu}$$

for all  $u, v \in V \setminus \{s, t\}$ . Note a feasible solution is simply values  $f_e$  for all  $e \in E$ , in other words, a function  $f : V \times V \rightarrow \mathbb{R}$ . Since flow conservation property is satisfied, this is saying that we are trying to find the pseudo-flow  $f$  such that  $|f| = 1$  and  $\sum_{e \in E} l_e f_e$  is minimized. This is equivalent to finding the solution to shortest path problem as justified in the previous question. Now we prove linear program yields correct solution. Let  $f$  be solution to the linear program. Let  $p$  be the path such that for all edges  $e$  in  $p$ , we have  $f_e = 1$ . Such path is always possible by the fact that we constrain source to admit 1 unit of flow and sink to accept 1 unit of flow. Now we prove that the path is the shortest. Prove by contradiction, assume there is some other path  $p'$  with

$$w(p') < w(p) = \sum_{e \text{ in } p} l_e = \sum_{e \text{ in } p} l_e \times 1 + \sum_{e \text{ not in } p} l_e \times 0 = \sum_{e \in E} l_e f_e$$

which contradicts the fact that  $\sum_{e \in E} l_e f_e$  is minimal. Hence the solution to linear programming is correct. Note the transformation in the equation is possible as proved in the previous question

## Problem 5

1. In the Traveling Salesman Problem (TSP), we are given a directed graph  $G = (V, E)$  with an integer weight  $w(e)$  for each edge  $e \in E$ , and we are asked to find a simple

cycle (more strict than MST, since MST cannot have cycle) over all the vertices (a circuit) with minimum total weight. (Note that the weights  $w(e)$  can be positive or negative.) Show how to represent an arbitrary instance of the TSP as an integer program. Justify that your representation is correct, and describe how to obtain a solution to the instance of the TSP from solutions to your integer program.

*Solution.*

□

We formulate similarly to problem 4. Pick arbitrary  $s \in V$ , add another vertex  $t$  and its incident edges to  $G$  such that  $t$  has the exact same incident edges (including their weights) of  $s$ . Let  $G' = (E', V')$  be the resulting graph. Consider pseudo-flow  $f : V \times V \rightarrow \mathbb{R}$  be units of flow from  $s$  to  $t$ . the value of  $f_e$  determines if an edge is used in the circuit.

$$\begin{aligned} \text{Mimimize:} \quad & \sum_{e \in E'} w(e) f_e \\ \text{Subject to:} \quad & \sum_{v \in V'} f_{uv} - \sum_{v \in V'} f_{vu} = \begin{cases} 1 & \text{if } u = s \\ -1 & \text{if } u = t \\ 0 & \text{otherwise} \end{cases} \\ & \sum_{v \in V'} f_{uv} = 2 \quad \text{for all } u \in V' \setminus \{s, t\} \\ & f_e \geq 0 \quad \text{for all } e \in E \end{aligned}$$

The first constraint given states that the model follows flow conservation and that the demand is 1 unit of flow. The second constraint states that any vertex has 2 incident edges for which the flow go through, therefore any edge in  $E' \setminus \{s, t\}$  has degree 2. Altogether the feasible region consists of a flow  $f$  such that for all  $(u, v) \in E'$  for which  $f_{uv} \neq 0$  forms a simple path from  $s$  to  $t$ . The path is simple and spanning precisely because every vertex except from start and end has degree 2. Since we are minimizing  $\sum_{e \in E'} w(e) f_e$ , the path will be least weight. Therefore, if LP has a solution,

the resulting  $f_e$  specifies the least weight flow from  $s$  to  $t$ . Let  $E_p \subseteq E'$  be a set of edges where  $f_e = 1$  for all  $e \in E_p$ . Now to get the simple cycle, we first replace the instance of vertex  $t$  with  $s$  in edges in  $E'$ , then start from  $s$ , and add  $u$  to the sequence of vertices if  $(s, u) \in E_p$ .

2. Your company has recently discovered that several of its problems can be solved using linear programming. Your company doesnt want to write their own solver program, since many efficient programs are available for sale. But your boss has been reading his spam again and went out and purchased the MELPSE system (Most Efficient Linear Program Solver Ever!) in a fit of misdirected leadership. As expected, the

claim is slightly overstated, and the package comes with some serious limitations. From the advertisement:

MELPSE is the fastest and most streamlined LP solver ever! Using the latest technology, it will find the best non-negative values for all your variables, get the biggest value for your objective function, and it will even make sure that  $Ax \leq b$ !

In fact, this is all that MELPSE can do: it only supports non-negative variables (it implicitly enforces a constraint  $x_1 \geq 0$  on all variables  $x_i$ ), only allows maximizations of the linear objective function, and insists that all constraints are in the form  $\sum_{i=1}^n a_{j,i}x_i \leq b_j$  (where  $a_{j,i}$  and  $b_j$  are real numbers, perhaps negative). **Exit-DFS** The linear programs you need to solve are usually minimization problems, where variables occasionally take negative values, and some constraints are expressed with equality or greater-than-or-equal. The boss has already spent your entire budget, so to save your team you will need to figure out how to use the MELPSE system to solve your problems.

- (a) One of your problems fits the restrictions of MELPSE except that you need to minimize your objective function. Describe precisely how to convert your program into one MELPSE can solve; that is, describe how to create an equivalent LP where the objective is being maximized. Briefly explain how to use a solution to your new program to find a solution to your original problem.

*Solution.*

□

Given a minimization linear program  $L$ , we can convert  $L$  to an equivalent maximization linear program  $L'$  by negating the coefficients in the objective function.  $L$  and  $L'$  are equivalent since they share the same feasible solution (since constraints unchanged) and for each feasible solution, the objective value in  $L$  is the negative of the objective value in  $L'$ , therefore by definition of equivalence,  $L$  and  $L'$  are equivalent. The solution to the converted  $L'$  is the solution to  $L$  without any modification

- (b) Another of your problems contains a constraint of the form  $a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b$ . Describe precisely how to convert this to a constraint of the form  $a'_1x_1 + a'_2x_2 + \cdots + a'_nx_n \leq b'$  as required by MELPSE.

*Solution.*

□

Let  $a'_i = -a_i$  for all  $i = 1, \dots, n$  and let  $b' = -b$

- (c) This problem also has a constraint of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

Describe precisely how to create an equivalent LP that can be solved by MELPSE (or in the form of part (b)).

*Solution.*

□

Replace the given equality constraint with

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b \quad \text{and} \quad a_1x_1 + a_2x_2 + \cdots + a_nx_n \geq b$$

The resulting LP are equivalent because the objective function is unchanged and the constraints are equivalent because of the fact that

$$a = b \iff a \geq b \wedge a \leq b$$

- (d) From the previous parts we know how to solve a minimization problem with  $\geq$  or  $=$  constraints. We can assume that all constraints are in the form  $a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$ . But we have a problem where one of the variables,  $x_1$ , could be negative. Describe precisely how to construct an equivalent LP where we replace  $x_1$  with two new variables which can only take non-negative values. Briefly explain how to use a solution to your new program to find a solution to your original problem.

*Solution.*

□

Replace each occurrence of  $x_1$  in original LP  $L$  with  $x'_1 - x''_1$  in both the objective function and the constraints, add the constraints  $x'_1, x''_1 \geq 0$ . Let  $L'$  be the resulting LP. A feasible solution  $\hat{x}$  to  $L'$  corresponds to a feasible solution  $\bar{x}$  to  $L$  where  $\bar{x}_1 = \hat{x}'_1 - \hat{x}''_1$ . Conversely, a feasible solution  $\bar{x}$  to  $L$  corresponds to a feasible solution  $\hat{x}$  given by

- i.  $\hat{x}'_1 = \bar{x}_1$  and  $\hat{x}''_1 = 0$  if  $\bar{x}_1 \geq 0$ .
- ii.  $\hat{x}'_1 = 0$  and  $\hat{x}''_1 = \bar{x}_1$  if  $\bar{x}_1 < 0$ .

Therefore,  $L$  and  $L'$  are equivalent. Note the above 2 points are always the case because  $x'_1$  and  $x''_1$  are always of different signs. And if both are in the objective function, the positively signed variable will be optimized in the objective function, to the point such that the other variable is 0.

- (e) Write an efficient high-level algorithm that will convert a linear program that might be a maximization problem, might contain greater-than-or-equal or equality constraints, and may contain variables lacking a non-negativity constraint, into an equivalent linear program that can be solved by MELPSE. Give a good bound on the size (the size being the number of variables and number of constraints) of your new linear program and briefly justify it.

*Solution.*

□

**Algorithm** Let  $L' = (A', b', c')$  be the result given  $L = (A, b, c)$ . Let there be  $n$  variables in the solution and  $m$  constraints

- i. If the algorithm is as minimization linear program, let  $c' = -c$
- ii. If there exists a constraint  $\sum_{j=1}^n a_{ij}x_j = b_i$  for some  $i = 1, \dots, m$ , replace the constraint with  $\sum_{j=1}^n a_{ij}x_j \geq b_i$  and  $\sum_{j=1}^n a_{ij}x_j \leq b_i$
- iii. If there exists a constraint  $\sum_{j=1}^n a_{ij}x_j \geq b_i$  for some  $i = 1, \dots, m$ , let  $a'_{ij} = -a_{ij}$  for all  $j = 1, \dots, n$ , and revert the sign from  $\geq$  to  $\leq$
- iv. if exists a negative variable  $x_j < 0$  for some  $j = 1, \dots, n$ , then replace  $x_j$  with  $x'_j - x''_j$  in LP and add constraints  $x'_j, x''_j \geq 0$

Now we discuss worst case size of the resulting  $L'$ . The worst case happens when we have an unconstrained  $x_j$  for all  $j = 1, \dots, n$  variables and that every constraint is an equality constraint.

- i. For each  $x_j$  lacking a nonnegative constraint, the conversion doubles the size of variables by removing one variable  $x_j$  and adding 2 variables  $x'_j, x''_j$  into  $L'$ . Hence size of variable at most doubles. The conversion also introduces  $2n$  nonnegative constraints for  $x'_j$  and  $x''_j$
- ii. For each equality constraint, the conversion replaces the constraint with 2 inequality constraints, therefore the size of constraint at most doubles

Altogether, the size of  $L'$  is bounded by  $2n + 2n + 2m = 4n + 2m$