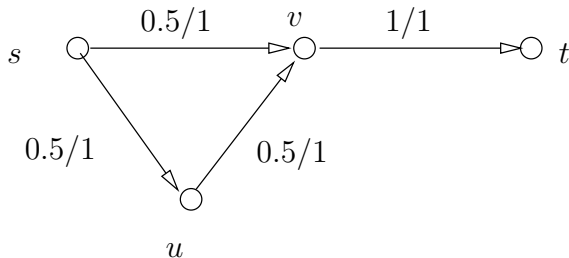


Solutions to Homework Assignment #4

Answer to Question 1.

a.



The label “ f/c ” on each edge indicates that the edge has capacity c , f of which is used by the flow. It is easy to check that the specified flow is proper (i.e., it satisfies the capacity and conservation constraints). It is a maximum flow because it has value 1, and the (s, t) -cut $(\{s, u, v\}, \{t\})$ has capacity 1.

b. For any set of vertices S , let $out(S)$ be the set of edges with tail in S and head not in S . Let (S^*, T^*) be a min-cut of the flow graph G , and f^* be a max-flow in G . By Proposition 7.6 in KT, $V(f^*) = \sum_{e \in out(S^*)} f^*(e) - \sum_{e \in out(T^*)} f^*(e)$. By the max-flow-min-cut theorem, $V(f^*) = \sum_{e \in out(S^*)} c(e)$. Thus, by the capacity constraint, every edge in $out(S^*)$ is saturated in f^* . By assumption, $V(f^*) > 0$, and so there is some $e \in out(S^*)$ such that $f^*(e) > 0$. As we just saw, such an edge is saturated in f^* , i.e. $f^*(e) = c(e)$. Since all capacities are integers by assumption, $f^*(e)$ is a positive integer, as wanted.

c. We will prove, by induction on k , that for each integer k such that $0 \leq k \leq m$, there is an integral flow f such that $V(f) = k$.

BASIS. $k = 0$. Take $f(e) = 0$ for each edge e . This trivially satisfies the capacity and conservation constraints. Thus it is an integral flow whose value is 0, as wanted.

INDUCTION STEP. Step Let k be an arbitrary integer such that $0 < k \leq m$. Assume that there is an integral flow f' whose value is $k - 1$. We will prove that there is an integral flow f whose value is k . Since f' has value $k - 1 < m$, it is not a maximum flow. Therefore, the residual graph $G_{f'}$ has an (s, t) -path, say p . Because f' is an integral flow, the residual capacities of $G_{f'}$ are integers, and so the bottleneck of p has residual capacity at least 1. Let

$$f(e) = \begin{cases} f'(e) + 1, & \text{if } e \text{ is a forward edge on } p \\ f'(e) - 1, & \text{if the reverse of } e \text{ is a backward edge on } p \\ f'(e), & \text{otherwise} \end{cases}$$

The function f is a flow in G : It satisfies the capacity constraint because the residual capacity of every edge on p in $G_{f'}$ is at least 1. It satisfies the conservation constraint because f' does: For each vertex on path p (other than s and t), the incoming traffic changes by v , for some $v \in \{-1, 0, +1\}$, if and only if the outgoing traffic also changes by v ; while for each vertex not on p , neither the incoming nor the outgoing traffic changes. Since f' is an integral flow, so is f . Furthermore, $V(f) = V(f') + 1 = k$.

Answer to Question 2.

a. In Dijkstra's algorithm we keep track of a set S of vertices, and for each vertex v two quantities, $d(v)$ and $p(v)$: $d(v)$ is the minimum length of any path from s to v among all paths whose intermediate vertices (i.e., vertices other than v) are in S , or ∞ if no such path exists; and $p(v)$ is the predecessor of v on such a path. (The length of a path p is the sum of the lengths of its edges: $\sum_{e \text{ on } p} \ell(e)$.)

The set S starts off containing only s (in which case the corresponding d and p values are straightforward to compute). In each iteration we augment S by a vertex u , currently not in S , that has minimum $d(u)$, and we update $d(v)$ and $p(v)$ for each vertex v adjacent to u to reflect the change in S .

In our modification of Dijkstra's algorithm, instead of $d(v)$ we keep track of $b(v)$: the *maximum* bottleneck of any path from s to v among all paths whose intermediate vertices are in S , or 0 if no such path exists. (The bottleneck of a path p is the minimum capacity of its edges: $\min_{e \text{ on } p} c(e)$.) In each iteration we augment S by a vertex u , currently not in S , that has *maximum* $b(u)$. The modified algorithm is shown below:

```

 $S := \{s\}$ 
for each vertex  $v$  do
  if  $v$  is adjacent to  $s$  then  $b(v) := c(s, v); p(v) := s$ 
  else  $b(v) := 0; p(v) := \text{nil}$ 
while  $S \neq V$  do
  let  $u$  be a vertex not in  $S$  that has maximum  $b(u)$ 
   $S := S \cup \{u\}$ 
  for each vertex  $v$  adjacent to  $u$  do
    if  $b(v) < \min(b(u), c(u, v))$  then  $b(v) := \min(b(u), c(u, v)); p(v) := u$ 

```

b. Let S be the set of vertices in G_f that are reachable from s using edges with residual capacity more than δ ; let T be the remaining vertices. Clearly, $s \in S$; by the definition of δ , $t \in T$. Thus, (S, T) is an (s, t) -cut of G .

Let $out(S)$ be the set of edges in G with tail in S and head in T ; $out(T)$ is defined similarly. (Note that in these definitions we are considering edges in the flow graph G , not the residual graph G_f .)

Claim. For every $e \in out(S)$, $f(e) \geq c(e) - \delta$; and for every $e \in out(T)$, $f(e) \leq \delta$.

PROOF OF CLAIM. If $(u, v) \in out(S)$ and $f(u, v) < c(e) - \delta$, then G_f has a forward edge (u, v) with residual capacity more than δ . Since $u \in S$, v is also reachable in G_f from s using edges with residual capacity more than δ , and therefore $v \in S$ — contradicting that $(u, v) \in out(S)$. Similarly, if $(u, v) \in out(T)$ and $f(u, v) > \delta$, then G_f has a backward edge (v, u) with residual capacity more than δ . Since $v \in S$, u is also reachable in G_f from s using edges with residual capacity more than δ , and therefore $u \in S$ — contradicting that $(u, v) \in out(T)$. \square

We now have:

$$\begin{aligned}
 V(f) &= \sum_{e \in out(S)} f(e) - \sum_{e \in out(T)} f(e) && \text{[KT (7.6)]} \\
 &\geq \sum_{e \in out(S)} (c(e) - \delta) - \sum_{e \in out(T)} \delta && \text{[above Claim]} \\
 &= \sum_{e \in out(S)} c(e) - \sum_{e \in out(S) \cup out(T)} \delta \\
 &\geq \sum_{e \in out(S)} c(e) - m \cdot \delta && \text{[} \leq m \text{ edges cross cut } (S, T)\text{]} \\
 &\geq V(f^*) - m \cdot \delta
 \end{aligned}$$

where, in the last inequality, we use the fact that the value of any flow (in particular f^*) is at most the capacity of any cut (in particular (S, T)). Rearranging, we get $V(f^*) - V(f) \leq m \cdot \delta$.

c. Let b be the bottleneck of P , the maximum-bottleneck s -to- t augmenting path in G_f . Therefore, every s -to- t path in G_f has an edge with residual capacity at most b . By part (b),

$$V(f^*) - V(f) \leq m \cdot b \quad (1)$$

Let $f' = \text{AUGMENT}(f, P)$. Thus,

$$V(f') = V(f) + b \quad (2)$$

Therefore,

$$\begin{aligned} \frac{V(f^*) - V(f')}{V(f^*) - V(f)} &= \frac{V(f^*) - (V(f) + b)}{V(f^*) - V(f)} && \text{[by (2)]} \\ &= 1 - \frac{b}{V(f^*) - V(f)} \\ &\leq 1 - \frac{b}{mb} && \text{[by (1)]} \\ &= 1 - \frac{1}{m} \end{aligned}$$

Rearranging, we get $V(f^*) - V(f') \leq (V(f^*) - V(f)) \cdot (1 - \frac{1}{m})$

d. Let f_0 be the trivial flow (that carries no traffic in any edge), and f_i be the flow after i augmentation steps, where in each step we choose the max-bottleneck augmenting path. By part (c) and induction, $V(f^*) - V(f_i) \leq (V(f^*) - V(f_0)) \cdot (1 - 1/m)^i$. Note that $V(f_0) = 0$ and $(1 - 1/m)^i < e^{-i/m}$ (since $1/m \neq 0$, and $1 - x < e^{-x}$ for all $x \neq 0$). Thus, $V(f^*) - V(f_i) < V(f^*) \cdot e^{-i/m}$. Letting $i = m \ln C$, we get $V(f^*) - V(f_i) < V(f^*)/C \leq 1$ (for the last inequality we use the fact that, by definition of C , $V(f^*) \leq C$). Since each f_i is an integral flow and f^* is a maximum flow, it follows that $V(f^*) - V(f_i) = 0$, i.e., $V(f_i) = V(f^*)$. So, after at most $i = m \ln C = \Theta(m \log C)$ augmentations the algorithm computes a maximum flow.

Answer to Question 3. Intuitively, given a graph G we define a new graph as follows: We “split” each vertex v of G into two “halves” v_1 and v_2 ; we add edges connecting the two halves of each vertex, as well as edges connecting the second half of a vertex v to the first half of each vertex u adjacent to v (in G). In this way, paths that shared a vertex v in the original graph will share an edge (the one that connects the two halves of v) in the constructed graph. Consequently, the problem of finding vertex-disjoint paths in the original graph reduces to the problem of finding edge-disjoint paths in the constructed graph. This is a problem we already know how to solve.

More precisely, let $G = (V, E)$ be the given graph. Define

$$\begin{aligned} V' &= V \times \{1, 2\} \\ E' &= \{((v, 1), (v, 2)) : v \in V\} \cup \{((v, 2), (u, 1)) : (v, u) \in E\} \end{aligned}$$

Also define $G' = (V' - \{(s, 1), (t, 2)\}, E' - \{((s, 1), (s, 2)), ((t, 1), (t, 2))\})$. (That is, for technical reasons we throw away the first half of s and the second half of t .)

There is a natural 1-1 correspondence between s -to- t paths in G and $(s, 2)$ -to- $(t, 1)$ paths in G' : The path $p = s, v_1, v_2, \dots, v_k, t$ in G corresponds to the path $p' = (s, 2), (v_1, 1), (v_1, 2), (v_2, 1), (v_2, 2), \dots, (v_k, 1), (v_k, 2), (t, 1)$ in G' . Conversely, the only $(s, 2)$ -to- $(t, 1)$ paths in G' have the form of p' (i.e., alternating edges connecting two halves of a split vertex of G and edges connecting the second half of a vertex in G to the first half of one of its adjacent vertices), which corresponds to p . It is clear that if p_1 and p_2 are vertex-disjoint s -to- t paths in G , then p'_1 and p'_2 are edge-disjoint $(s, 2)$ -to- $(t, 1)$ paths in G' , and vice-versa. Thus, a maximum-size set of vertex-disjoint s -to- t paths in G corresponds to a maximum-size set of edge-disjoint $(s, 2)$ -to- $(t, 1)$ paths in G' .

Our algorithm then is as follows:

construct G'
 run the max edge-disjoint path algorithm on G' with source $(s, 2)$ and sink $(t, 1)$
 let P' be the set of paths it returns
 $P := \{p : p \text{ corresponds to some path } p' \in P'\}$
 return P

Let $n = |V|$ and $m = |E|$. Then \hat{G} has $2n - 2$ vertices and $n + m - 2$ edges. The edge-disjoint path algorithm runs in $O((2n - 2)(n + m - 2)) = O(mn)$ time. Constructing the paths in P from those in P' takes $O(m + n)$ time. Thus, this algorithm for finding a maximum-size set of vertex-disjoint s -to- t paths takes $O(mn)$ time.

Answer to Question 4.

a. We construct a flow network $G = (V, E)$ as follows. The vertex set V consists of

- A source s and a sink t ;
- For each student s_j , three vertices labeled S_j , S_j^1 , and S_j^2 . Intuitively, vertex S_j represents the course load of student s_j ; this course load is split into the student's fall and winter course loads, represented by S_j^1 and S_j^2 respectively.
- For each course c_i , a vertex C_i .

The edge set E consists of

- For each student s_j , an edge (s, S_j) with capacity 5 (the maximum course load allowed per student).
- For each student s_j , edges (S_j, S_j^1) and (S_j, S_j^2) , each with capacity 3 (the maximum course load per student allowed in each term).
- For each student s_j and each course c_i such that $c_i \in C_j$ (i.e., c_i is a course that student s_j is interested in taking) and c_i is offered in the fall (respectively, winter) term, an edge (S_j^1, C_i) (respectively, (S_j^2, C_i)) with capacity 1.
- For each course c_i , an edge (C_i, t) with capacity ℓ_i (the enrolment limit of the course).

Call an assignment of students to courses **feasible** if it satisfies all the constraints: a student is only assigned to courses he/she is interested in taking, a student is assigned to a total of at most five courses, at most three in any term, and no course is assigned more students than its enrolment limit. There is a natural correspondence between feasible assignments and integral flows in the above flow graph: Given a feasible assignment, we define a flow as follows: If s_j is assigned k courses, put a flow of k on the edge (s, S_j) ; if k' of these courses are in the winter term, put a flow of k' on edge (S_j, S_j^1) and a flow of $k - k'$ on edge (S_j, S_j^2) . If s_j is assigned course c_i offered in the fall (respectively, winter) term, put a flow of 1 on edge (S_j^1, C_i) (respectively (S_j^2, C_i)). If a total of ℓ students are assigned to course c_i , put a flow of ℓ on edge (C_i, t) . It is easy to verify that this is indeed a flow (i.e., it satisfies the capacity and conservation constraints). Conversely, from an integral flow, we can obtain a feasible assignment by interpreting each unit of flow along an s -to- t path going through vertices S_j and C_i as meaning that student s_j is assigned to course c_i .

Thus, a maximum flow in this graph corresponds to a feasible assignment of students to courses that maximises the number of courses to which students are assigned, and hence the department's revenue.

b. The tricky part here is expressing the objective function.

We introduce variables x_{ji} , for each student $s_j \in S$ and course $c_i \in C$. The intention is that, in a feasible solution,

$$x_{ji} = \begin{cases} 1, & \text{if student } s_j \text{ is assigned to course } c_i \\ 0, & \text{otherwise} \end{cases}$$

We start with the statement of the constraints that any assignment of students to courses must satisfy, according to the specification of the problem.

- For every $1 \leq i \leq m$ and $1 \leq j \leq n$, if $c_i \notin C_j$, add the constraint $x_{ji} := 0$. (Each student is assigned only to courses in which he/she is interested.)
- For every $1 \leq j \leq n$, add the constraint $\sum_{i=1}^m x_{ji} \leq 5$. (Each student is assigned to at most five courses.)
- Let $F = \{i : c_i \text{ is offered in the fall term}\}$ and $W = \{i : c_i \text{ is offered in the winter term}\}$. For every $1 \leq j \leq n$, add the constraints $\sum_{i \in F} x_{ji} \leq 3$ and $\sum_{i \in W} x_{ji} \leq 3$. (Each student is assigned to at most three courses each term.)
- For every $1 \leq i \leq m$, add the constraint $\sum_{j=1}^n x_{ji} \leq \ell_i$. (The number of students assigned to each course does not exceed that course's enrolment limit.)

To define the objective function, we introduce a variable y_j for every student s_j . The intention is that

$$y_j = \begin{cases} 1, & \text{if student } s_j \text{ has been assigned at least five courses} \\ 0, & \text{otherwise} \end{cases}$$

The objective function then becomes:

- Maximize $\sum_{j=1}^n y_j$.

To ensure that the variables y_j have the intended semantics, we introduce the following constraints:

- For every $1 \leq j \leq n$, add the constraint $y_j \leq \frac{1}{5} \sum_{i=1}^m x_{ji}$.

Notice that, by themselves, these constraints do not ensure the intended semantics for y_j . For example, we could satisfy them by setting all y_j s to 0. However, since the objective function is to *maximize* the sum of y_j s, an optimal solution will assign to each y_j the maximum possible value consistent with the constraint that y_j is at most one-fifth of the number of courses to which s_j is assigned. Since all variables are restricted to take on values 0 or 1, y_j can be set to 1 only if student s_j is assigned to (at least) five courses.

THAT'S IT WITH HOMEWORK SOLUTIONS, FOLKS!