Solutions to Practice Problems 10

Solution 1: We show how to modify Dijkstra's algorithm in order to solve the single-source maximum capacity path problem. We will focus only on the final capacities, and not on how to compute the actual path (but this is an easy extension).

For any two vertices $u, v \in V$, let $\mu(u, v)$ denote the capacity of the maximum capacity path from u to v. (If v is not reachable from u, then $\mu(u, v) = 0$.) To compute the max-capacity path from s to t, we solve the single-source max-capacity problem.

We modify Dijkstra's algorithm as follows. For each $v \in V$, define $m[v]$ to be the current estimate on the max-capacity path from s to v. Initially, $m[s] = +\infty$, and for all other vertices v, $m[v] = 0$. Because we are maximizing, rather than minimizing, the priority queue returns the maximum, rather than the minimum, element.

We also modify the relax operator $\mathsf{relax}(u, v)$ as follows. Given that $m[u]$ is the max-capacity of getting from s to u, we know that there is a path from s to v of capacity $\min(m[u], c(u, v))$. If this is larger than the current estimate, $m[v]$, we should take it. Thus, the relaxation rule for edge (u, v) is changed as follows (see Fig. 1(a)):

$$\text{relax}(u, v) : m[v] \leftarrow \max \begin{pmatrix} m[v], \\ \min(m[u], c(u, v)) \end{pmatrix}$$

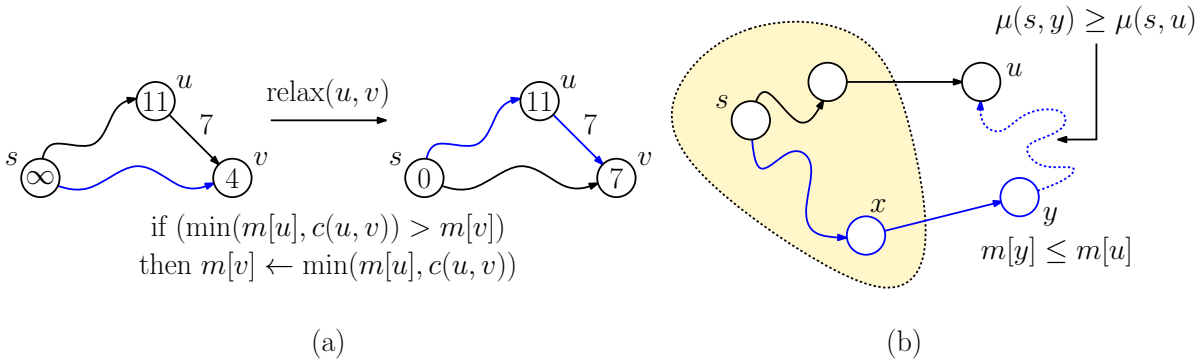

Figure 1: Adapting Dijkstra to compute the max-capacity path.

The final change to Dijkstra's algorithm is that the queue is reverse-ordered, returning the unprocessed vertex with the highest m-value. The running time is clearly the same as Dijkstra's algorithm. To establish the algorithm's correctness, we adapt the correctness proof of Dijkstra's algorithm.

Claim: Whenever the algorithm processes a vertex u, $m[u]$ contains its correct capacity value, that is, $m[u] = \mu(s, u)$.

Proof: Suppose to the contrary that this fails to be true for some vertex, and let u be the first such instance. Since $m[u]$ is based on evidence of an actual path, we have $m[u] < \mu(s, u)$.

Since this is incorrect, consider the true max-capacity path from s to u. This path must first cross an edge (x, y) where x is among the set of processed vertices and y is not (see Fig. 1(b)). (Possibly $x = s$ and/or $y = u$.)

Since x was processed earlier (and u is the first mistake), we know that $m[x] = \mu(s, x)$. Since (x, y) is an edge along the max-capacity path, we have $\mu(s, y) = \min(\mu(s, x), c(x, y))$. Since we performed $\mathsf{relax}(x, y)$, we have correctly propagated this information along the edge. This implies that

$$m[y] \;=\; m[x] + c(x, y) \;=\; \mu(s, x) + c(x, y) \;=\; \mu(s, y).$$

Since u was chosen before y to be processed, we know that $m[u] \geq m[y]$. Furthermore, any edges along the remaining max-capacity path from y to u can only decrease its capacity, implying that $\mu(s, u) \leq \mu(s, y)$. Combining these observations, we have

$$\mu(s, u) \;>\; m[u] \;\geq\; m[y] \;=\; \mu(s, y) \;\geq\; \mu(s, u),$$

which is a clear contradiction.

Solution 2: We solve the drone delivery problem via reduction to circulations with vertex demands and lower and upper flow capacities on each edge. The flow on each edge will reflect the number of drone flights made.

We generate a network $G = (V, E)$ as follows. We create a source vertex s and a sink vertex t. We create two sets of vertices, one for the drone stations, denoted $\{d_1, \ldots, d_m\}$, and one for the customers, denoted $\{c_1, \ldots, c_m\}$. We create the following edges (see Fig. 2(a)):

- For $i \in [1, m]$, and $j \in [1, n]$, edge (d_i, c_j) of capacity range $[0, 2]$ if $\text{dist}(d_i, c_j) \leq 10$. (The capacity constraint enforces the condition that there are at most 2 drone flights from any station to any one customer, and the distance condition enforces the restriction that the drone station must be within 10 miles of the customer.)
- For $i \in [1, m]$, edge (s, d_i) of capacity range $[0, 5]$. (This enforces the FAA requirement that each drone station can launch at total of at most 5 deliveries per day.)
- For $j \in [1, n]$, edge (c_j, t) of capacity range $[\max(1, o_j - 2), o_j]$. (This enforces the condition that at least $\max(1, o_j - 2)$ deliveries succeed and at most the number of requested orders.)
- Edge (t, s) of capacity range $[0, \infty]$. (This is just a technical requirement that the flow be balanced throughout the system. This is because we cannot predict the total number of deliveries in advance, so we do not know exactly how much flow will travel from the source to the sink.)

All the vertex demands are set to 0, meaning that flow is conserved at every vertex. We then invoke the circulation algorithm to generate an integer-valued circulation f. If no circulation exists, we declare that there is no valid delivery schedule. Otherwise, for each edge (d_i, c_j), we ship $f(d_i, c_j)$ deliveries from station d_i to customer c_j. (For example, in Fig. 2(b), $f(1, 2) = 2$, implying that we would ship two deliveries from d_i to c_j.) The following claim establishes the correctness of this reduction.

Claim: There exists a valid delivery schedule if and only if G has a feasible circulation.

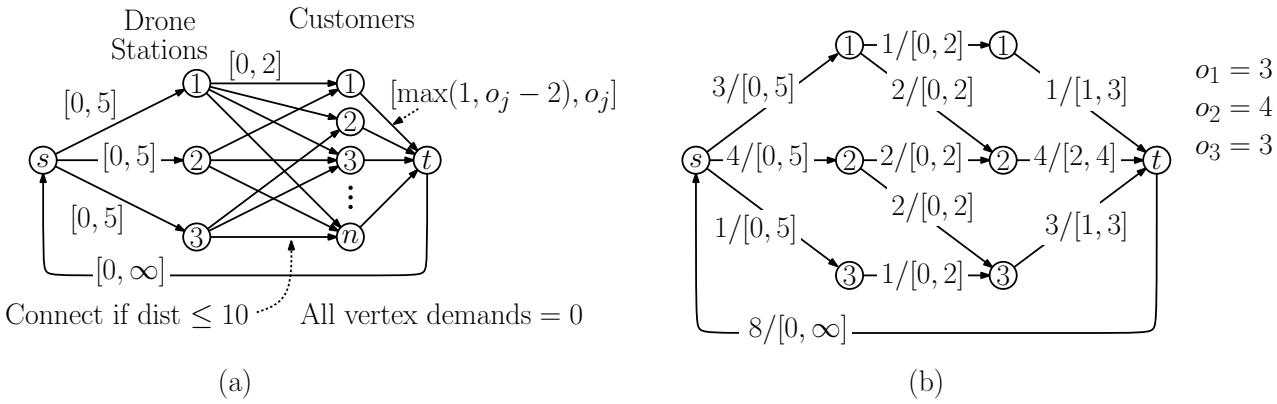

Figure 2: Solving the drone delivery problem by network flow.

Proof: ($\Rightarrow$) Suppose that there is a valid delivery schedule, where station d_i makes $g(d_i, c_j)$ deliveries to customer c_j. We create a circulation f in G as follows. First, for each edge (d_i, c_j), we assign it a flow of $g(d_i, c_j)$. Next, define the flow on edge (s, d_i) to be the total number of deliveries made by station d_i, and define the flow on edge (c_j, t) to be the total number of deliveries made to customer c_j. Finally, define the flow from t to s to be the total number of deliveries to all customers.

To see that this yields a feasible circulation in G, observe that each edge (d_i, c_j) receiving flow must exist, because the fact that the delivery was made implies that $\text{dist}(d_i, c_j) \leq 10$. Next, to see that the capacity constraints are satisfied, observe that the validity of the schedule implies the following:

- No single drone station sends more than 2 deliveries to any customer (satisfying the capacity constraint on (d_i, c_j)).
- No drone station sends more than 5 deliveries (satisfying the capacity constraint on (s, d_i)).
- Each customer receives between $\max(1, o_j - 2)$ and o_j deliveries (satisfying the capacity constraint on (c_j, t)).

It is also easy to see that the node demands (all zero) are satisfied, since the flow on each edge incoming to d_i is balanced by the outgoing deliveries from this station, and the flow on each edge leaving c_j is balanced by the incoming deliveries to this customer. Finally, the flow coming into and out of s and t are easily seen to equal the total number of deliveries in the entire schedule. Therefore, this is a valid circulation in G.

($\Leftarrow$) Suppose that G has a feasible circulation, denoted by f. We may assume that this is an integer flow. We define a delivery schedule as follows. For each edge (d_i, c_j) where $f(d_i, c_j) > 0$, we make this number of deliveries from station d_i to customer c_j. We will show that this is a valid delivery schedule. First, observe that flow is only sent along existing station-customer edges, meaning that they are all within the 10-mile radius. By the capacity constraints, we know that:

- $f(d_i, c_j) \leq 2$, implying that no customer receives more than 2 deliveries from any station.

- By the upper capacity constraint of 5 on edge (s, d_i) and flow conservation, no station sends more than 5 deliveries
- By the capacity constraints of $[\max(1, o_j - 2), o_j]$ on edge (c_j, t) and flow conservation, each customer receives between $\max(1, o_j - 2)$ and o_j deliveries.

Therefore, this satisfies all the requirements of a valid schedule.

Solution 3: The tutor-assignment problem can be reduced to a network circulation problem with lower and upper edge constraints. Given an instance of the student-tutor pairing problem (see Fig. 3(a)), we construct a network G as follows. First, we create student vertices $\{s_1, \ldots, s_n\}$ and tutor vertices $\{t_1, \ldots, t_m\}$, a special source vertex s^*, and a special sink vertex t^*. All the node demands are set to 0. We also create the following edges (see Fig. 3(b)). Intuitively the flow on an edge will be 1 to model the fact that a tutor has been assigned to a student. Recall that the notation $[a, b]$ on an edge means that the flow on the edge must be at least a and at most b.

- For each student s_j, create edge (s^*, s_j) of capacity $[1, 1]$. (This enforces the condition that each student is paired with exactly one tutor.)
- For each suitable student-tutor pair, create edge (s_j, t_i) of capacity $[0, 1]$. (This enforces the condition that students are only paired with suitable tutors.)
- From each tutor t_i, create edge (t_i, t^*) of capacity $[a_i, b_i]$. (This enforces the condition that tutor t_i is paired with at least a_i and at most b_i students. Remember that lower and upper bounds can only be placed on edges. They do not apply to vertex demands.)
- Create edge (s^*, t^*) of capacity $[n, n]$. Equivalently, we could have defined s^* to have a demand of $-n$ and t^* to have a demand of n. (Since each of n students is paired with exactly one tutor, this enforces the condition that there will be a total of n assignments in any valid solution.)

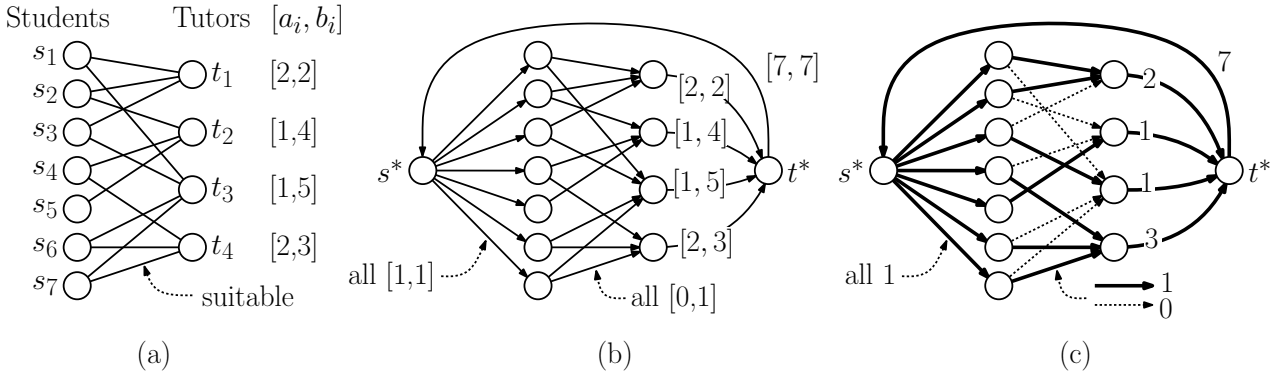

Figure 3: Solution to the tutor-assignment problem.

Correctness follows from the next lemma.

Lemma: There is a feasible solution to the student-tutor pairing instance if and only if G has a feasible circulation.

Proof: ($\Leftarrow$) Let f be a feasible circulation for G. Because the edge capacities are integers, we may assume that this is an integer flow. We define a pairing as follows. For each student-tutor edge (s_j, t_i) that carries a nonzero flow, we pair student s_j to tutor t_i. We assert that this is a valid student-tutor pairing. Because we only create edges between suitable student-tutor pairs, the pairings selected are all suitable. Because the demands are all zero, and each student has an incoming capacity of $[1, 1]$ it follows that each student is paired with exactly one tutor. Analogously, because each tutor has an outgoing capacity of $[a_i, b_i]$, it follows that each tutor is paired with the appropriate number of students.

($\Rightarrow$) Suppose there is a valid student-tutor pairing. We create a circulation for G as follows. First, we generate one unit of flow along each edge (s^*, s_j). For each student-tutor pairing (s_j, t_i), we create a singly unit of flow on the corresponding edge, and otherwise we set the flow to 0. For each tutor t_i, let c_j denote the number of students paired to this tutor. We set the flow along edge (t_i, t^*) to c_j. Finally, we set the flow along edge (t^*, s^*) to n. We assert that this flow function defines a valid circulation for G. Because the pairing is valid, each student is paired with exactly one tutor, and thus its demand of zero is satisfied. Also, each tutor is paired with between a_i to b_i students, and so its flow demand is also satisfied.

Solution 4:

(a) We reduce the roadblock problem to that of computing the maximum flow in a network with vertex capacities. (A reduction from this problem to standard network flows was given in Practice Problems 9.) In particular, construct a network where the network G' is equal to the given graph G, where each undirected edge $\{u, v\}$ is replaced by a pair of directed edges (u, v) and (v, u). Let $s \leftarrow p$ denote the source node and $t \leftarrow q$ denote the sink node for the network. Assign s and t capacities of ∞ and all the other vertices are assigned a capacity of 1. Then compute the maximum flow in the resulting vertex-capacitated network using the reduction mentioned above. Correctness is established in the following lemma.

Lemma: G' has an s-t cut of capacity k if and only if k roadblocks suffice to separate p from q in G.

Proof: ($\Rightarrow$) Given an s-t cut in G' of capacity k, we construct k roadblocks as follows. The reduction generates two types of edges, those of capacity 1 and those of capacity ∞. The unit capacity edges each correspond to a vertex in G. Since the cut has capacity k, none of the infinite capacity edges cross the cut, only the edges of unit capacity. There are clearly k such edges. Define the roadblocks to be the corresponding set of k vertices. Because this is a cut, there cannot be a path from p to q that avoids these vertices, therefore this is a valid roadblock set, as desired.

($\Leftarrow$) Given a set of k roadblock vertices in G, consider the corresponding k edges in G'. Since these vertices all have unit capacity, these edges all have unit capacity. Because there is no path from p to q in G that avoids these vertices, the removal of these k edges disconnects s from t, implying that they define a cut in G' of capacity k, as desired.

The total running time is $O(n+m)$ to construct the networks G and G', $T(n, m)$ to solve the max-flow problem, and $O(n+m)$ to compute the reachable vertices X by an graph traversal algorithm (e.g., BFS or DFS). Thus, the total time is $O(n + m + T(n, m)) = O(T(n, m))$.

(b) This variant corresponds to a multi-source, multi-sink variant of the max-flow problem. We create a super source node s and a super sink node t. Both of these have capacity ∞. We then add edges from s to each of the vertices of P and add edges from each node of Q to t. We then apply part (a) of this problem to the resulting network. (Observe that neither s nor t can be be in the minimum cut since both have infinite capacity.) Since we have added only two vertices to the network, its size is essentially the same as before, so the overall asymptotic running time is unchanged.

Solution 5: Suppose that $G[f]$ has a cycle, say $\langle u_0, u_1, \ldots, u_k \rangle$, where $u_k = u_0$. Let $h = \min_{1 \le i \le k} f(u_{i-1}, u_i)$ be the minimum flow value on all of these edges. Construct a new flow f', which is equal to f on all the edges of G except those of the cycle, where we define

$$f'(u_{i-1}, u_i) \leftarrow f(u_{i-1,u_i}) - h.$$

We assert that f' is a valid flow and $|f'| = |f|$. First, because we have subtracted the minimum of the flows on the cycle, no flow value has become negative. Second, observe that for every vertex of G, the number of edges of the cycle entering the vertex equals the number of edges of the cycle exiting the vertex, and thus, the decrease in the incoming flow equals the decrease in the outgoing flow. Therefore, f' satisfies flow conservation. Finally, the value of the flow has not changed, because by our definition of an s-t network, s has no incoming edges, and therefore the value of the flow (sum of flows out of s) has not changed.

In going from $G[f]$ to $G[f']$, the flow values on edges has only decreased, and at least one edge has been removed (the one whose flow defined h). Such a process must eventually terminate. Therefore, after a finite number of applications of this process, we will arrive at a valid flow f'' such that $|f''| = |f|$, and f'' has no cycles. For example, in Fig. 4(a), we show a flow of value 3 with a cycle. We reduce the flow on the cycle by 3 units. There is still another cycle, and reduce its flow also by 3 units. The final network has no flow cycles and the same flow value.

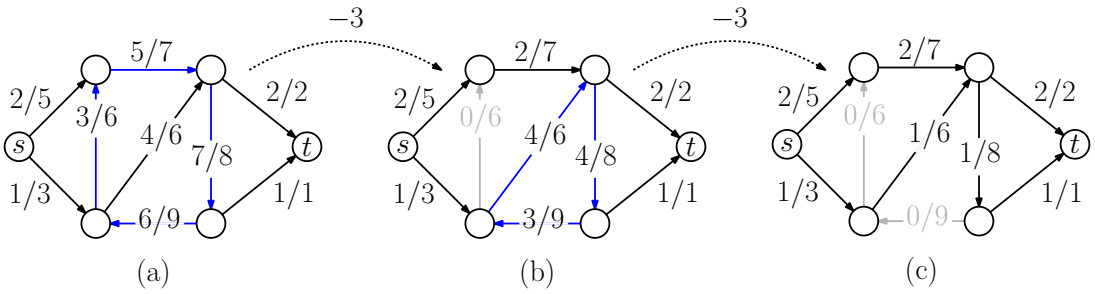

Figure 4: Removing cycles from a flow.

Solution to the For-Fun problem: In order to satisfy the conditions, you must be holding either a 4 or 5 and your roommate holds the next higher number. If you see a 4, you announce that your roommate holds a 5 (and your roommate announces that you hold the 4), and if you see a 5, you announce that your roommate holds a 6 (and your roommate announces that you hold the 5). We'll show that if your numbers had been smaller, then answer would have been discovered earlier in the process. If your numbers had been larger, then you both would have wound up losing.

Assuming you and your roommate are both pretty smart, you each reason as follows from the starting of the process:

- If you saw the number 1 on your card, you would know that your roommate must hold the number 2, since there is no number 0 and they differ by 1. When you say "no," your roommate can infer that you do not have a 1.
- Once you say "no", if your roommate had either a 1 or 2, they would be able to infer your number. If they had a 1, then they know you have the 2. If they had a 2, they already know that you cannot have a 1, so you must have a 3. When your roommate says "no," you know that your roommate has neither a 1 nor a 2.
- Once your roommate says "no", you know that your roommate has neither 1 nor 2. If you had either 2 or 3, you would now know your roommate's number. (If you had 2 your roommate must have 3, and if you had 3, your roommate must have 4.) When you say "no," your roommate knows that do not have any of the numbers 1–3.
- After your second "no", if your roommate had either a 3 or 4, they would be able to infer your number. If they had a 3, then they know you have the 4. If they had a 4, they already know that you cannot have a 3, so you must have a 5. The second time your roommate says "no," you know that your roommate has none of the numbers 1–4.
- Generalizing this inductively, with each pair of exchanges 2 more numbers are eliminated from the pool of possibilities. In particular, after the kth round of no's, you have been able to infer that your roommate has none of the numbers 1 through $2k$, and so if you are holding either of the numbers $2k$ or $2k + 1$, you can infer that your roommate must be holding the next larger number.

Just as the devil has given up, you get lucky. The number on your card happens to be either 4 or 5. Since your roommate cannot hold 1–4, if you see a 4, you know that your roommate must hold a 5. If you see a 5, you know that your roommate must hold a 6. Your roommate reasons similarly that you hold the other number.