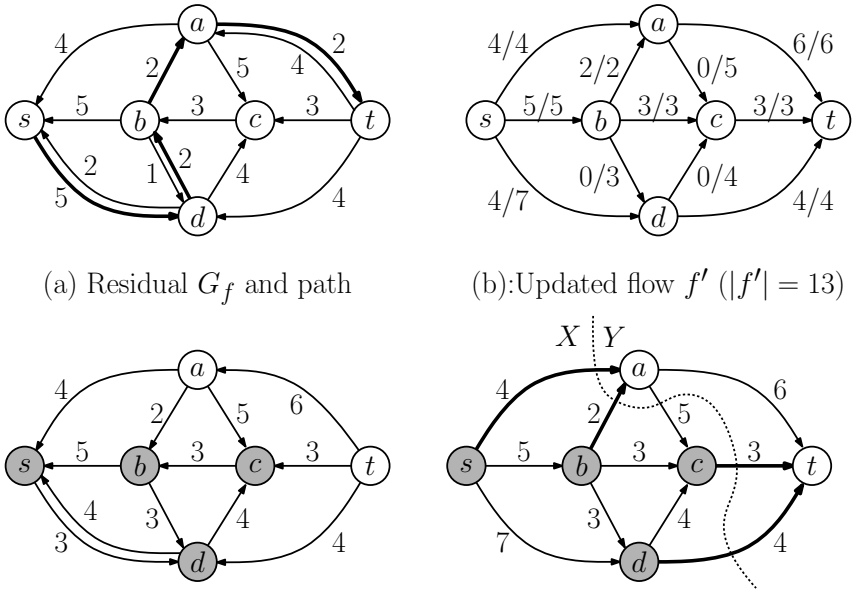
Solutions to Homework 4: Network Flow and Applications

Solution 1:

(a) The residual network G_f is shown in Fig. 1(a).



(a) Residual G_f and path

(b):Updated flow f' ($|f'| = 13$)

(c) Updated residual $G_{f'}$ and reachable nodes from s

(d) Cut of capacity 18

Figure 1: Solution to Problem 1.

(b) The path is highlighted with heavy edges in Fig. 1(a). (Another possible path is $\langle s, d, c, b, a, t\rangle$.) In either case, the maximum flow that can be pushed through this path is 2.

(c) The updated flow is shown in Fig. 1(b) and the updated residual network $G_{f'}$ is shown in Fig. 1(c).

(d) There is no path from s to t in the $G_{f'}$. (The vertices that are reachable from s are shaded.) It follows that this flow is maximum. Its value is the sum of the flow values out of s, which is $4 + 5 + 4 = 13$.

(e) The flow from (c) was already maximum, so the residual graph is the same as for part (c).

(f) As shown in the Min-Cut/Max-Flow Theorem, a minimum cut (X, Y) results by setting X the nodes of the residual reachable from s and Y to the rest. This yields the cut $(\{s, b, c, d\}, \{a, t\})$ shown in Fig. 1(d). The capacity of the cut is equal to the sum of capacities of edges crossing from the X side to the Y side, which is $4 + 2 + 3 + 4 = 13$. This is the minimum cut, since its capacity equals the value of the flow f'. There is a second minimum cut $(\{s, a, b, c, d\}, \{t\})$, which (of course) has the same capacity.

Solution 2: This 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. 2(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. 2(b)):

- For each student s_j, create edge (s^*, s_j) of capacity $[1, 1]$. (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 suitable student-tutor pair, create edge (s_j, t_i) of capacity $[0, 1]$.
- From each tutor t_i, create edge (t_i, t^*) of capacity $[a_j, b_j]$.
- Create edge (s^*, t^*) of capacity $[n, n]$. (Any $[a, b]$ range that contains n works.)

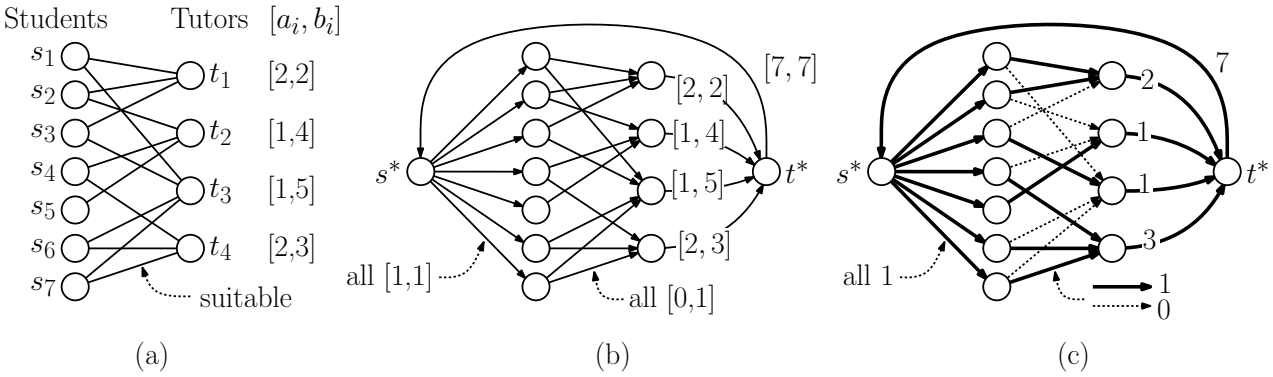


Figure 2: Solution to Problem 2.

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 3: As in class, we make the implicit assumption that there exists at least one path of strictly positive capacity from s to t. (If not, the maximum flow is zero, and no edge is critical.)

We assert that *any edge that crosses a minimum cut is critical.* To see why, suppose that (X, Y) is a minimum cut, and let (x, y) be any edge that crosses this cut for $x \in X$ and $y \in Y$. The capacity of the cut is the sum of capacities of all X-Y crossing edges, and so decreasing the capacity of (x, y) decreases the capacity of the cut. By the Min-Cut/Max-Flow Theorem, decreasing the minimum cut decreases the maximum flow value in the network.

To compute such an edge, we use the procedure suggested in the proof of the Min-Cut/Max-Flow proof. First, compute the maximum flow f in the network (by any max-flow algorithm) and then compute the residual network G_f for this flow. Let X denote the vertices that are reachable from s in G_f, and let $Y = V \setminus X$. (Since f is maximum, there is no s-t path in G_f, implying that Y is nonempty). As shown in the min-cut/max-flow proof, (X, Y) is a cut of capacity $|f|$. Therefore, any edge (x, y) where $x \in X$ and $y \in Y$ suffices.

Solution 4:

(a) We show that M is a maximum matching iff G has no alternating path with respect to M.

($\Rightarrow$) We will show the contrapositive, namely that if G has an alternating path with respect to M, then M is not maximum. Given any alternating path (see Fig. 3(b)), we remove the even numbered edges from the matching and insert the odd numbered edges into the matching. (In Fig. 3(c) we remove the solid edges and add the dashed edges.) Because the path has an odd number of edges, it follows that this increases the number of edges in the matching by exactly one. The result is still a matching, because (except for the first and last vertices, which were unmatched) every vertex along the path loses one edge from the old matching and gains one edge from the new matching. The resulting matching M' has one more edge, implying that M was not the maximum matching.

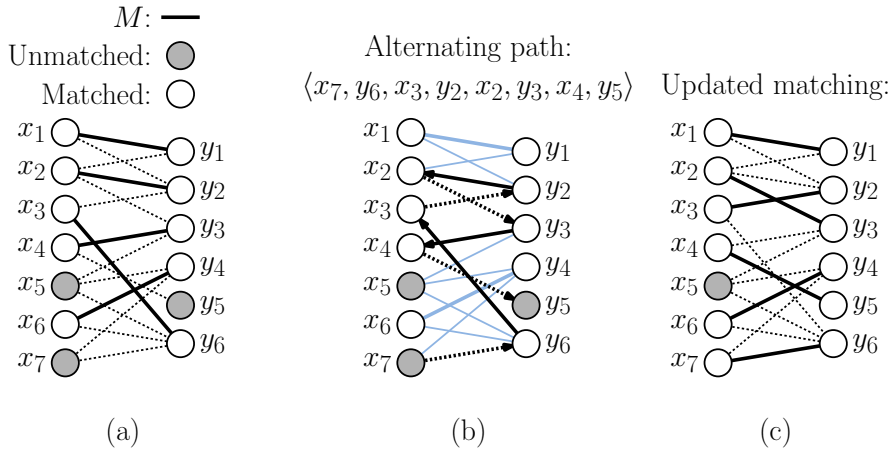


Figure 3: Solution to Problem 4(a) ($\Rightarrow$).

($\Leftarrow$) Again, we will show the contrapositive, namely that if M is not the maximum matching, then G has an alternating path with respect to M. Let M' be any matching, where $|M'| > |M|$ (see Fig. 4(c)). Define the *symmetric difference* of M and M', denoted Δ to be the set consisting of edges from one set or the other, but not both (see Fig. 4(d)). (More formally, $\Delta = (M \cup M') \setminus (M \cap M')$.)

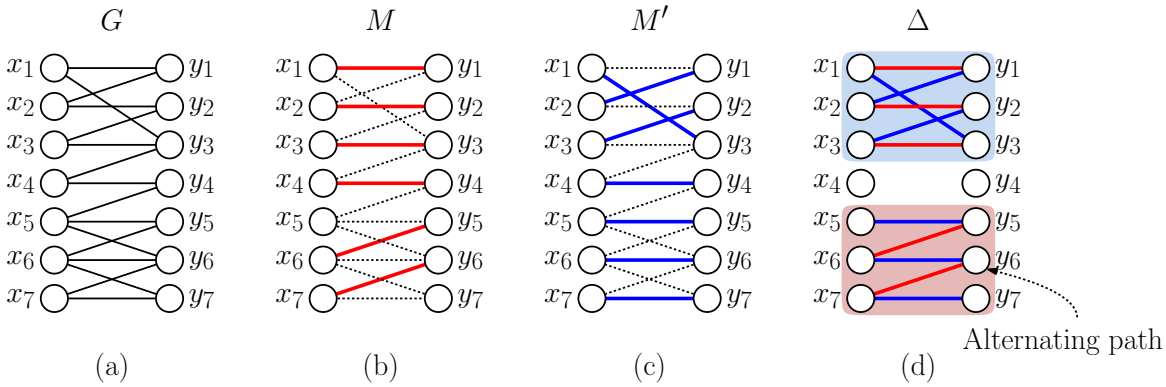


Figure 4: Solution to Problem 4(a) ($\Leftarrow$).

Because every vertex of G is incident to at most one edge of M and at most one from M', each vertex is incident to at most two edges of Δ. Therefore, Δ consists of the disjoint union of paths and cycles from G, where the edges of each path and cycle alternate between M and M'. (For example, in Fig. 4(d) there is one cycle and one path.)

We will show that Δ has an alternating path. Because the graph is bipartite, all its cycles are of even length (since they must alternate between X and Y), and so each cycle of Δ has an equal number of edges from each matching. Because $|M'| > |M|$ it follows that there must be at least one path that starts and ends with an edge of M'. Such a path must start and end at an unmatched vertex of M, and by definition of the symmetric difference, it must alternate between the matched and unmatched edges of M. Therefore, this is an alternating path. (Such a path is shown in Fig. 4(d).)

(b) To determine whether G has an alternating path for a given matching M, we reduce the problem to computing a path in a new network G_M. First, we create a source vertex s that has an outgoing edge to each unmatched vertex of X, and we create a sink vertex t that has an incoming edge from each unmatched vertex of Y (see Fig. 5(b)). For each (undirected) edge (x_i, y_j) in G, we direct the edge from x_i to y_j if the edge is not in M, and we direct it from y_j to x_i if it is in M. Let G_M denote the resulting graph.

We claim that there is an s-t path in G_M if and only if G has an alternating path with respect to M.

- ($\Rightarrow$) If G_M has an s-t path then by construction the edge leaving s enters an unmatched vertex of X. By our choice of edge directions this path then alternates between edges that are not in M with edges that are in M (see Fig. 5(c)). Finally, to complete the path to t the last vertex is an unmatched vertex of Y. Thus, this is an alternating path in G.

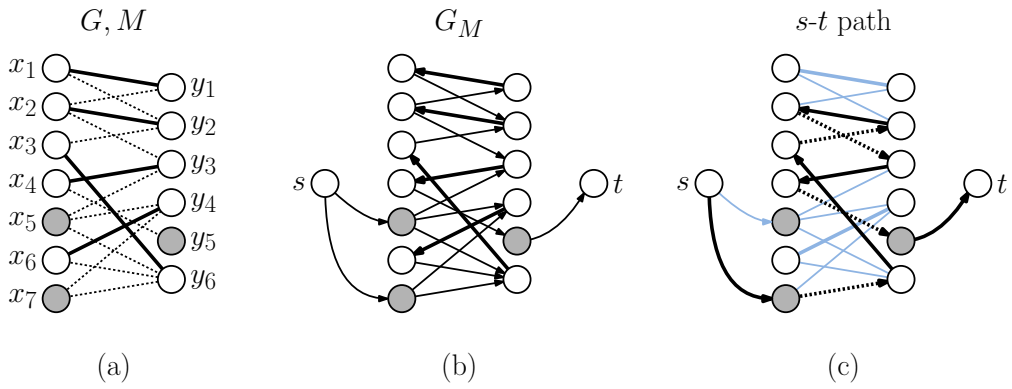


Figure 5: Solution to Problem 4(b).

- ($\Leftarrow$) If G has an alternating path, then this path starts at an unmatched vertex of X, ends at an unmatched vertex of Y, and alternates between non-matching and matching edges. The starting vertex is a neighbor of s, by our edge directions the odd (non-matching) edges of the path proceed from X to Y, and the even (matching) edges of the path proceed from Y to X. Finally, the last vertex of Y is unmatched, implying that it has an edge to t. Therefore, there is an s-t path in G_M.

To compute the alternating path, we invoke DFS (or BFS) to G_M to determine whether there is a path from s to t. If we cannot reach t, there is no alternating path. Otherwise, we can use predecessor links to compute the path.[1] Clearly, if G has n vertices and m edges, the construction of G_M can be performed in $O(n+m)$ time, and DFS/BFS takes the same time.

(c) The algorithm for finding a maximum matching follows directly from (b). Starting with an empty matching ($M = \emptyset$) we repeatedly invoke the procedure from (b) to compute an alternating path in $O(n+m)$ time. We then apply the edge swapping from (a) to increase the size of the matching by one edge. Since a matching can have at most $\lfloor n/2 \rfloor$ edges, the maximum number of iterations is $O(n)$. The total time is $O(n(n+m))$. We may assume that G is connected (for otherwise, just run the algorithm on each component separately), and therefore $n+m$ is $O(m)$, and hence the total time is $O(nm)$.

Solution 5: We model the bridges as a bipartite graph $G = (X, Y, E)$ where the vertices X lie on the upper bank of the river and the vertices Y lower bank of the river. Because the bridges do not cross over each other, we can sort them from left-to-right as $E = \langle e_1, \ldots, e_m \rangle$ (see Fig. 6(a)). For $0 \le i \le m$, let $E_i = \langle e_1, \ldots, e_i \rangle$ denote the i leftmost bridges. For each edge e_i, let $x_i \in X$ denote its endpoint on the north side and let $y_i \in Y$ denote its endpoint on the south side. (Thus, if two consecutive bridges e_{i-1} and e_i share a common upper endpoint, then $x_{i-1} = x_i$.)

We solve the problem of computing the minimum number of guardhouses to a DP problem. A natural way to solve the problem by DP would be to consider computing the minimum number of bridges for E_i, for all values of i. However, there is a dependency between these problems, since

[1] In the DFS/BFS, each time a vertex u discovers a neighbor v, we set pred$[v] = u$. To find the path, we trace the predecessor links back from t to s and then reverse the sequence.

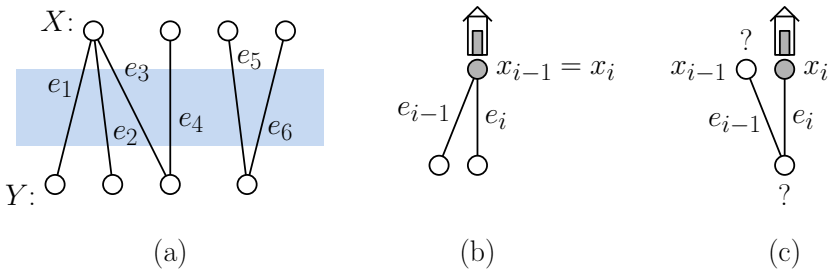


Figure 6: Solution to Problem 5.

different bridges may share a common endpoint. To handle this, we will *condition* the problem based on where the last guardhouse was placed.

For $1 \leq i \leq m$, define $X(i)$ to be the minimum number of new guardhouses needed to guard the bridges of E_i, subject to the assumption that a guardhouse has already been erected at its upper endpoint x_i, and define $Y(i)$ similarly by subject to the assumption that a guardhouse has already been erected at its lower endpoint of y_i.

Let us just consider $X(i)$, since $Y(i)$ is symmetrical. For the basis cases, we have $X(1) = 0$, since this bridge has already been guarded by the guard placed at x_1. In general, for $i \geq 2$, we consider two cases. If the previous edge e_{i-1} shares a common north endpoint with e_i (that is, if $x_{i-1} = x_i$) then the guardhouse at x_i already guards the north end of e_{i-1}, and so we don't need any more guards. We have $X(i) = X(i-1)$ (see Fig. 6(b)). On the other hand, if the previous edge does not share a common north endpoint with e_i ($x_{i-1} \neq x_i$) then we must place a new guardhouse at either x_{i-1} or y_{i-1} to guard edge e_{i-1}. If we place it at x_{i-1}, we pay $+1$ for the new guardhouse and the remaining cost is $X(i-1)$. If we place it at y_{i-1}, we pay $+1$ for the new guardhouse and the remaining cost is $Y(i-1)$. Thus, we have $X(i) = 1 + \min(X(i-1), Y(i-1))$ (see Fig. 6(c)). The rule for $Y(i)$ is symmetrical. In summary, we have

$$X(i) = \begin{cases} 0 & \text{if } i = 1 \\ X(i-1) & \text{if } i \geq 2 \text{ and } x_{i-1} = x_i \\ 1 + \min(X(i-1, Y(i-1)) & \text{if } i \geq 2 \text{ and } x_{i-1} \neq x_i. \end{cases}$$

Reasoning analogously for $Y(i)$, we have

$$Y(i) = \begin{cases} 0 & \text{if } i = 1 \\ Y(i-1) & \text{if } i \geq 2 \text{ and } y_{i-1} = y_i \\ 1 + \min(X(i-1, Y(i-1)) & \text{if } i \geq 2 \text{ and } y_{i-1} \neq y_i. \end{cases}$$

The final guardhouse will be placed either at the north or south end of the final edge, and so the final answer is given by $1 + \min(X(m), Y(m))$. Because each entry $X(i)$ and $Y(i)$ takes $O(1)$ time to compute, this DP formulation will lead to an algorithm with a running time of $O(n)$.

Solution to Challenge Problem 1: While we have described the approach to computing a maximum matching in Problem 4 as a "different approach" it is really equivalent to executing the Ford-Fulkerson algorithm on the reduction of bipartite matching to network flow that was presented in class. In particular, the graph presented in the solution 4(b) is essentially the same as the residual graph that arises in that construction. In particular, each edge (x_i, y_j) that is in

the current matching is carrying a flow of one unit, and (since this edge has a capacity of one) it contributes a backward edge (y_j, x_i) to the residual graph. Each edge that is not in the matching carries a flow of zero, and hence it contributes a forward edge to the residual. The edges of our construction are directed in exactly this same manner. (A similar argument applies to the edges incident to s and t.)

Solution to Challenge Problem 2.: Given a graph $G = (V, E)$, a subset V' of vertices is called a *vertex cover* if for every edge $(u, v) \in E$ either $u \in V'$ or $v \in V'$. The guardhouse problem is equivalent to the problem of computing a minimum-sized vertex cover in a bipartite graph. Even though the two problems may seem unrelated, we will show a rather remarkable fact that the size of a minimum vertex set in a bipartite graph is equal to the size of a maximum matching in the graph.[2] This remarkable relationship is known as *König's Theorem.*

First observe that if any graph has a matching of size k, then any vertex cover must contain at least k vertices (in order to cover just the edges of the matching). Thus, it suffices to compute a vertex cover whose size equals the size of the maximum matching.

We have seen in the solution to Problem 4 how to compute the maximum matching in a bipartite graph through the use of alternating paths. Let us define a *partial alternating path* to be one that starts at an unmatched vertex in X, alternates between unmatched and matched edges, but we don't care where it ends. A partial alternating path that ends at an unmatched vertex of Y is called a *full augmenting path.*

Given a maximum matching M, we can define a vertex cover as follows. Recalling the notation used Problem 4, let $V = X \cup Y$ be the vertices of the bipartite graph. Let Z denote all the vertices that are reachable via partial alternating paths that start from any unmatched vertex in X. Let $C = C_X \cup C_Y$, where $C_X = X \setminus Z$ and $C_Y = Y \cap Z$. We assert:

- C **is a vertex cover:** We will show that every edge (x, y) is covered. It suffices to show either that $x \notin Z$ (and hence is in C) or $y \in Z$ (and hence is in C). First, we may assume that (x, y) is not in any partial alternating path for otherwise we would have $y \in Z$, and we'd be done. If $(x, y) \in M$, then x cannot be in Z since the only way an alternating path can enter x is through a matched edge, contradicting the fact that (x, y) is not in any partial alternating path. Alternatively, if $(x, y) \notin M$, then x cannot be in Z since any alternating path that enters x could be extended by adding (x, y). In either case, $x \notin Z$, as desired.

- **Every vertex of** C **is the endpoint of edge of** M**:** Since Z is a superset of the unmatched vertices of X and since C_X is disjoint from Z, all the elements of C_X are matched. Also, since G has no full alternating paths (by the maximality of M) all the vertices of $Y \cap Z$ are matched, implying that this is true for C_Y as well.

- **At most one vertex of each matched edge is in** C**:** Each edge of M either has both endpoints in Z or both not in Z. By definition of C, both endpoints cannot contribute to C.

Therefore, C is a vertex cover, and (from the second two assertions) $|C| = |M|$, it is the smallest vertex cover possible.

[2]This is quite surprising, given that one problem involves minimizing a number of vertices and the other problem involves maximizing a number of edges. We saw two other examples of this earlier in the semester. The first is where we showed that the earliest-finish time algorithm solved both the (maximum-sized) interval scheduling problem and (minimum-sized) interval stabbing problem. The second is min-cut and max-flow. These are all examples of the fascinating concept of the primal-dual properties of linear optimization problems.