Algorithmics II (H)

Tutorial Exercises on Graph and Matching Algorithms

Outline Solutions

1. Form a graph G' by setting $G' = G$ initially and by deleting every vertex in $U \backslash R$ together with all edges incident to such vertices. Let M' be a maximum matching in G', which may be formed in $O(r(n+m))$ time using the augmenting path algorithm, where $r = |R|$. Then every edge in M' is incident to a red vertex. Now use M' as the starting point for the augmenting path algorithm in G. Once the algorithm terminates, M is a maximum matching in G. Furthermore, this step takes $O(r'(n+m))$ time, where $r' = |U| - |R|$. Finally, M matches at least as many red vertices as M' (this is because every matched vertex in M' remains matched in M). Hence M is a maximum red-matching in G, and M may be computed in $O(r(n+m) + r'(n+m)) = O(n(n+m))$ time.

2. Form a graph G by adding a vertex for each school-leaver in S. Add a vertex $v_{j,k}$ for each university place ($1 \le j \le q$, $1 \le k \le c_j$). For each school-leaver $s_i \in S$ and for each university $u_j \in A_i$, add an edge from s_i to $v_{j,k}$ ($1 \le k \le c_j$). A maximum matching in G is then a maximum feasible matching.

   The number of vertices on the school-leavers' side of G is p. For each university $u_i \in U$, let b_i denote the number of school-leavers who find u_i acceptable. Let m denote the number of edges in G. Then

$$\begin{aligned} m &= b_1c_1 + b_2c_2 + \ldots + b_qc_q \\ &\le C(b_1 + b_2 + \ldots + b_q) \\ &= CL. \end{aligned}$$

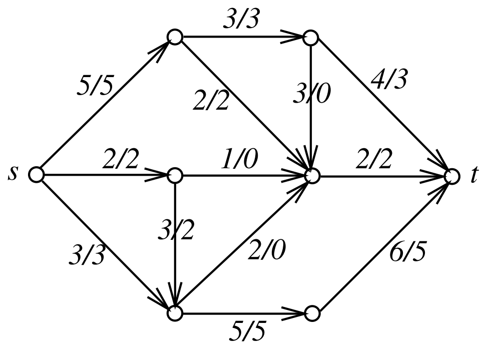
   The augmenting path algorithm has complexity $O(p(p+m)) = O(p(p+CL))$ as required.

3. Create a graph $G = (V, E)$ by adding a vertex for each student in S and for each project in P. For each student $s_i \in S$ and for each project $p_j \in A_i$, add an edge in G from s_i to p_j. Now let $X = \{x_1, \ldots, x_{t-s}\}$ and add the $t - s$ new vertices in X to the students' side of G. For each i ($1 \le i \le t - s$) and j ($1 \le j \le t$), add an edge in G from x_i to p_j.

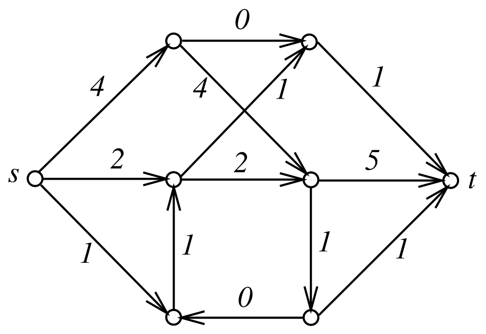
   Let $M' = \{\{x_i, p_i\} : 1 \le i \le t - s\}$. Then M' is a matching in G of size $t - s$. We may use M' as a starting point for the augmenting path algorithm. Let M be the matching returned upon termination. Then M is a maximum feasible matching. For, it is clear that Conditions (i), (ii) and (iii) are satisfied. We now consider Condition (iv). Every vertex $x_i \in X$ is matched in M', and by the properties of the augmenting path algorithm, x_i remains matched in M. The vertices in X are adjacent only to vertices in P'. Hence $t - s$ vertices in P' are matched to vertices in X, so at most $t - (t - s) = s$ vertices in P' are matched in M to student vertices, as required.

Finally we note that G has $p+(t-s)+q = O(n)$ vertices and $O(m+t(t-s))$ edges. All the vertices in X are matched in M', so that the outermost loop of the augmenting path algorithm has at most p iterations. Hence the whole process takes $O(p(n+m+t(t-s)))$ time.

4. The flow is not maximum; another flow with a larger value is shown as follows.



5. A maximum flow is shown as follows. (The integer adjacent to each edge denotes the flow along that edge.)



One minimum cut can be found by identifying the vertices that could send more flow to the sink if they had any to send. So here, one subset of the cut consists of the sink together with the vertices immediately to its west and south-west.

6. Let f be a maximum flow in G'. Define a set of edges M as follows:

$$M = \{\{u,w\} : (u,w) \in E' \wedge f(u,w) = 1\}.$$

Firstly, we claim that M is a matching in G. For, let $u \in U$. The only edge of G' directed towards u is (s,u), which has capacity 1. Hence the flow into u is at most 1, so by conservation of flow, the flow out of u is at most 1. Thus u is adjacent to at most one edge of M in G. By a similar argument, each $w \in W$ is adjacent to at most one edge of M in G.

Secondly, we claim that M is a maximum matching in G. For, suppose not. Then M admits an augmenting path $P = \langle u_1, w_1, u_2, w_2, \ldots, u_k, w_k \rangle$ for some $k \geq 1$, where $\{u_i, w_i\} \notin M$ $(1 \leq i \leq k)$ and $\{u_{i+1}, w_i\} \in M$ $(1 \leq i \leq k-1)$. Hence f admits the augmenting path $P' = \langle s, u_1, w_1, u_2, w_2, \ldots, u_k, w_k, t \rangle$ in G', a contradiction.

7. By definition, an augmenting path is a simple path from s to t in the residual graph G'_R of G' at any iteration of the Ford-Fulkerson algorithm. Since G has no edges between two vertices in U and no edges between two vertices in W, neither does the corresponding network G', and hence neither does G'_R. Also, the only edges involving s or t connect s to vertices in U and connect vertices in W to t. Note that in G', edges between vertices in U and W are only directed from U to W. However in G'_R, such edges may also be directed from W to U. Hence any augmenting path in G' has the form

$$s \to u_{i_1} \to w_{j_1} \to u_{i_2} \to w_{j_2} \to \ldots \to u_{i_k} \to w_{j_k} \to t,$$

for some $k \geq 1$, where $u_{i_r} \in U$ and $w_{j_r} \in W$ $(1 \leq r \leq k)$. No vertex may be used twice, and hence the path contains s, t and equal numbers of distinct vertices from U and W, i.e. at most $2 + 2\min(|U|, |W|)$ vertices in all. The length of an augmenting path is thus bounded above by $2\min(|U|, |W|) + 1$.

8. Let v be a vertex that is not on a path from the source s to the sink t. Then either (i) there is no path from s to v, or (ii) there is no path from v to t, or both (i) and (ii) are true. Consider any flow f in the network.

   Case (i): there can be no flow into vertex v, for any flow must come along one or more paths from s. So the flow along all edges entering or leaving v must be zero.

   Case (ii): there can be no flow out of vertex v, for any flow must go along one or more paths to t. So again the flow along all edges entering or leaving v must be zero.

   Hence deleting v and all incident edges does not alter the problem of finding the maximum flow in the network.

9. The student-optimal stable matching is

$$M_0 = \{(s_1, l_2), (s_2, l_6), (s_3, l_1), (s_4, l_3), (s_5, l_5), (s_6, l_4)\}$$

   and the lecturer-optimal stable matching is

$$M_z = \{(s_1, l_6), (s_2, l_3), (s_3, l_4), (s_4, l_5), (s_5, l_1), (s_6, l_2)\}.$$

10. (a) If s applies to l, then in the stable matching M output by the algorithm, either $(s, l) \in M$ or s has a worse partner than l in M. But M is the student-optimal stable matching, so there is no stable matching in which s has a partner better than $M(s)$.

    (b) If l receives an application from s, then in the stable matching M output by the algorithm, either $(s, l) \in M$ or l has a better partner than s in M. But M is the student-optimal stable matching, so there is no stable matching in which l has a partner worse than $M(l)$.

    (c) During the final iteration of the while loop, some lecturer l receives an application from a student s. If l was already assigned to some student s' at that point, then l rejects either s or s'. Since that student is then free again, the while loop cannot terminate at this iteration, a contradiction. Hence l does not reject any student, and the only way this can happen is that l was free when s applied. Hence l accepts this application, the algorithm terminates, and s was the first student to apply to l.

(d) By part (c), the algorithm terminates when the final lecturer receives his/her first application. Since the other $n-1$ lecturers can each receive at most n applications, the total number of applications is at most $n(n-1)+1 = n^2-n+1$.

11. Let I be an instance of SMT and let I' be any instance of the Stable Marriage problem obtained by breaking the ties in I arbitrarily (so that each person has a strictly-ordered preference list). Using the Gale/Shapley algorithm, we may find a stable matching M in I' in $O(n^2)$ time. It is then straightforward to verify that M is a stable matching in I. (For, in I', if some student s and lecturer l strictly prefer each other to their partners in M, then (s,l) would block M in I, contradicting the stability in M in I.)

12. Consider the following instance I_1 of SMT (where brackets indicate tied entries in a given preference list):

$$\begin{array}{lll} s_1: & l_1 & l_2 \\ s_2: & l_1 & l_2 \end{array} \qquad\qquad \begin{array}{lll} l_1: & (s_1 & s_2) \\ l_2: & s_1 & s_2 \end{array}$$

Students' preferences   Lecturers' preferences

Then I_1 has two stable matchings, namely $M_1 = \{(s_1,l_1),(s_2,l_2)\}$ and $M_2 = \{(s_1,l_2),(s_2,l_1)\}$. In M_1, student s_1 has his/her first-choice partner and student s_2 has his/her second-choice partner, whilst in M_2, student s_1 has his/her second-choice partner and student s_2 has his/her first-choice partner. Hence I_1 has no student-optimal stable matching.

13. We prove the result by induction on k.

For the base case, $k=0$. Consider any i and j ($1 \le i \le n$ and $1 \le j \le n$). If $i=j$ then the shortest path distance from v_i to v_j with no intermediate vertices is 0. Hence suppose that $i \ne j$. If $(v_i,v_j) \in E$ then a shortest path from v_i to v_j with no intermediate vertices is exactly the edge (v_i,v_j) and has weight $wt(v_i,v_j)$. Otherwise $(v_i,v_j) \notin E$, in which case there is no path from v_i to v_j with no intermediate vertices, and hence the shortest path distance is ∞ in this case. It can be verified that in all cases, these values correspond to the values in $D_0(i,j)$.

Now suppose that $0 < k \le n$ and the result is true for $k-1$. Consider any i and j ($1 \le i \le n$ and $1 \le j \le n$). Either (a) a shortest path from v_i to v_j whose intermediate vertices belong to $v_1,\dots,v_k$ involves v_k, or (b) it does not. In case (b), the intermediate vertices of such a path belong to the set $v_1,\dots,v_{k-1}$ – by induction hypothesis the length of such a path in contained in $D_{k-1}(i,j)$. In case (a) a shortest path must be obtained by following (i) a shortest path from v_i to v_k, whose intermediate vertices belong to the set $v_1,\dots,v_{k-1}$, and (ii) a shortest path from v_k to v_j, whose intermediate vertices again belong to the set $v_1,\dots,v_{k-1}$. By induction hypothesis the length of path (i) is $D_{k-1}(i,k)$ and the length of path (ii) is $D_{k-1}(k,j)$. Hence in case (a), the required shortest path length is $D_{k-1}(i,k)+D_{k-1}(k,j)$. It follows that the length of a shortest path from v_i to v_j whose intermediate vertices belong to $\{v_1,\dots,v_k\}$ is the minimum of the quantities computed for cases (a) and (b), and this is exactly the value in $D_k(i,j)$.

14.

$$D_0 = \begin{pmatrix} 0 & 6 & 2 & 7 & \infty \\ \infty & 0 & \infty & -1 & \infty \\ \infty & 2 & 0 & \infty & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} = D_1. \quad D_2 = \begin{pmatrix} 0 & 6 & 2 & 5 & \infty \\ \infty & 0 & \infty & -1 & \infty \\ \infty & 2 & 0 & 1 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 4 & 2 & 3 & 8 \\ \infty & 0 & \infty & -1 & \infty \\ \infty & 2 & 0 & 1 & 6 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} \quad D_4 = \begin{pmatrix} 0 & 4 & 2 & 3 & 4 \\ \infty & 0 & \infty & -1 & 0 \\ \infty & 2 & 0 & 1 & 2 \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix} = D_5.$$

$$\Pi_0 = \begin{pmatrix} - & 1 & 1 & 1 & - \\ - & - & - & 2 & - \\ - & 3 & - & - & 3 \\ - & - & - & - & 4 \\ - & - & - & - & - \end{pmatrix} = \Pi_1. \quad \Pi_2 = \begin{pmatrix} - & 1 & 1 & 2 & - \\ - & - & - & 2 & - \\ - & 3 & - & 2 & 3 \\ - & - & - & - & 4 \\ - & - & - & - & - \end{pmatrix}$$

$$\Pi_3 = \begin{pmatrix} - & 3 & 1 & 2 & 3 \\ - & - & - & 2 & - \\ - & 3 & - & 2 & 3 \\ - & - & - & - & 4 \\ - & - & - & - & - \end{pmatrix} \quad \Pi_4 = \begin{pmatrix} - & 3 & 1 & 2 & 4 \\ - & - & - & 2 & 4 \\ - & 3 & - & 2 & 4 \\ - & - & - & - & 4 \\ - & - & - & - & - \end{pmatrix} = \Pi_5.$$

Given any $v_i, v_j \in V$, the length of a shortest path from v_i to v_j is stored in $D_5(i,j)$, and the index k such that v_k is a predecessor of v_j on a shortest path from v_i to v_j is stored in $\Pi_5(i,j)$.