CSCI 570 Homework 5

Due Date: Nov. 19, 2023 at 11:59 P.M.

1. In the network below (See Fig. 1), the demand values are shown on vertices (supply values are negative). Lower bounds on flow and edge capacities are shown as (lower bound, capacity) for each edge. Determine if there is a feasible circulation in this graph. Please complete the following steps.

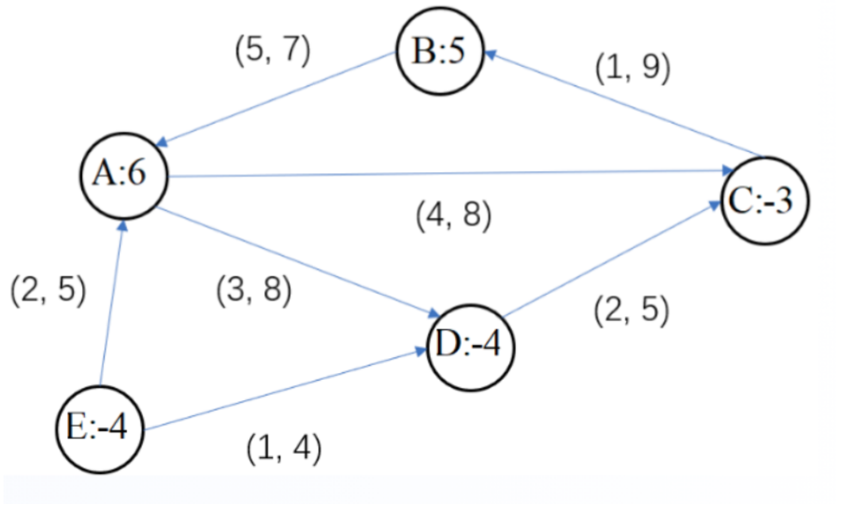


Figure 1

   (a) Remove the lower bounds on each edge. Write down the new demands on each vertex A, B, C, D, E, in this order.
   $A : 6, B : 9, C : -8, D : -6, E : -1.$

   (b) Solve the circulation problem without lower bounds. Write down the max-flow value.
   The max-flow value is: 9 (See Fig. 2).

   (c) Is there is a feasible circulation in the original graph? Explain your answer.

   Check if there is a s–t flow $|f| = 15$. The max-flow of this flow network is 9, so there is no feasible circulation (See Fig. 3).

2. The organizers of the 2022 US Open Tennis Championships are working on making the draws for the first round and scheduling the matches.

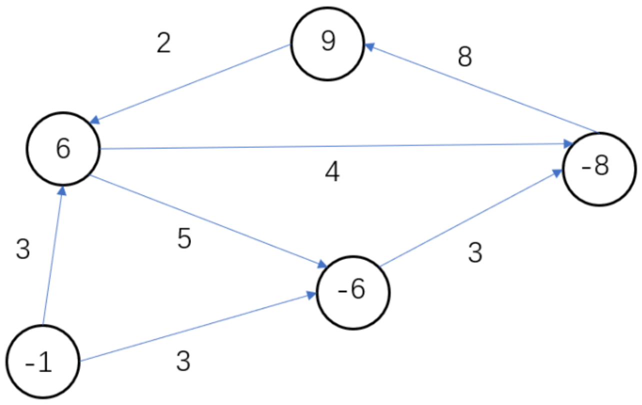


Figure 2

They are faced with several constraints in doing so.
There are $2N$ players, seeded (ranked) from 1 to $2N$. The players are divided into two halves - the top half containing players seeded 1 to N, and the bottom half containing seeds $N + 1$ to $2N$. The first round consists of N matches among the $2N$ players, each being one where a top-half player plays a bottom-half player.
Additionally, to keep them a bit more competitive, each match must be between players whose seeds differ by at most N' (a given constant $> N$).
The sports facility has courts $C_1, ..., C_m$. The matches will be scheduled in timeslots $T_1, ..., T_k$. Matches can run in parallel on different courts in a given timeslot. Assume all courts are available in all timeslots, however a single court can have at most p matches to preserve surface quality. For broadcasting reasons, in any given timeslot, there should be at least one match being played and at most r of them.
The bottom-half players were given the option to make requests if they preferred to play in certain timeslots, and each of them has specified k' of the k timeslots that they are okay to play in. Seeded players on the other hand were given the option to make requests to avoid playing on certain courts, and each of them has specified m' of the m courts that they do not want to play on.
Describe a Network Flow/Circulation based algorithm to determine if it is possible to come up with a feasible schedule of matches based on the

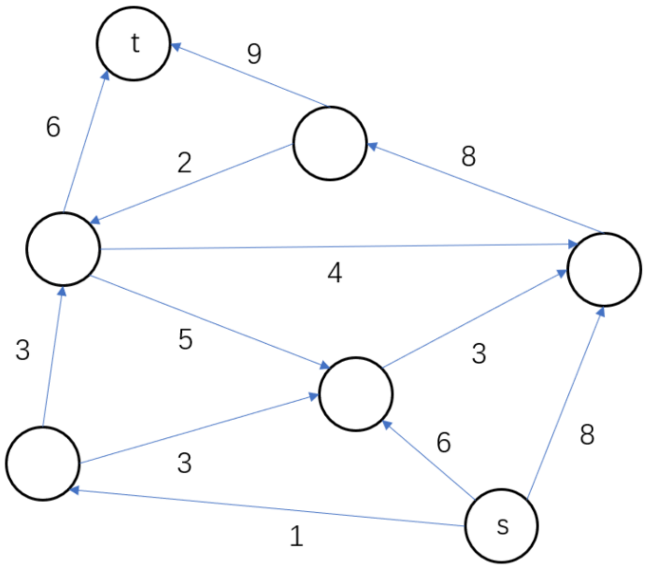


Figure 3

above constraints.

Solutions & Rubrics (See Fig. 4):

Nodes (2 pts):
S [0.5 pt], T [0.5 pt], and Partitions [1 pt] for:

- $C_1, ..., C_m$ courts
- $1, ..., N$ seeded (top-half) players
- $N+1, ..., 2N$ unseeded (bottom-half) players
- $T_1, ..., T_k$ timeslots

Edges (4 pts):

- Edges from S to all courts [0.25 pt]
- Edges from all slots to T [0.25 pt]
- Edge from every court to every seeded player except if that court is to be avoided for that player [1 pt]
- Edge from every seeded to every unseeded where seed diff is at most N' [1 pt]

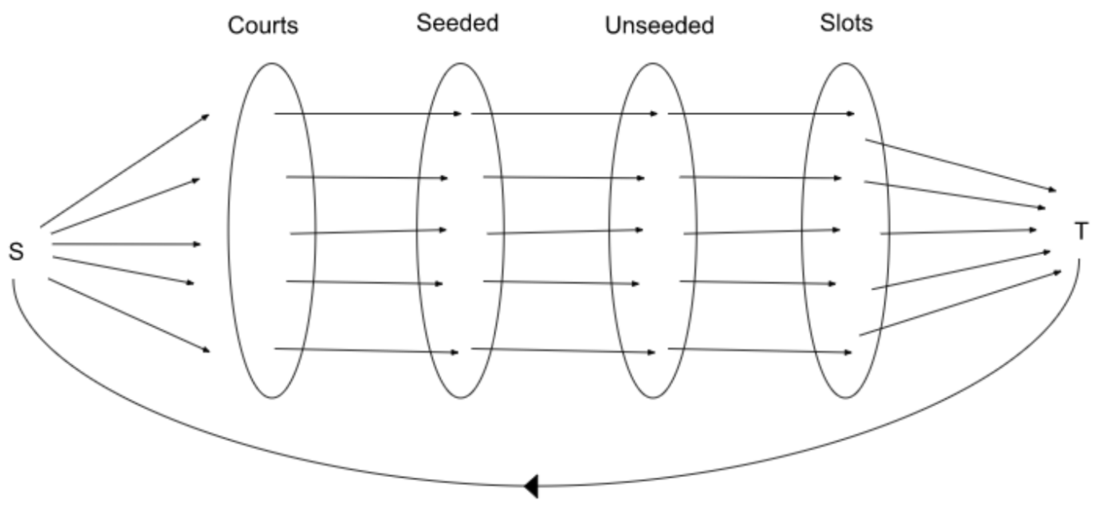


Figure 4

- Edge from every unseeded player to every slot which the player is okay with [1 pt]
- Back-edge from T to S [0.5 pt]

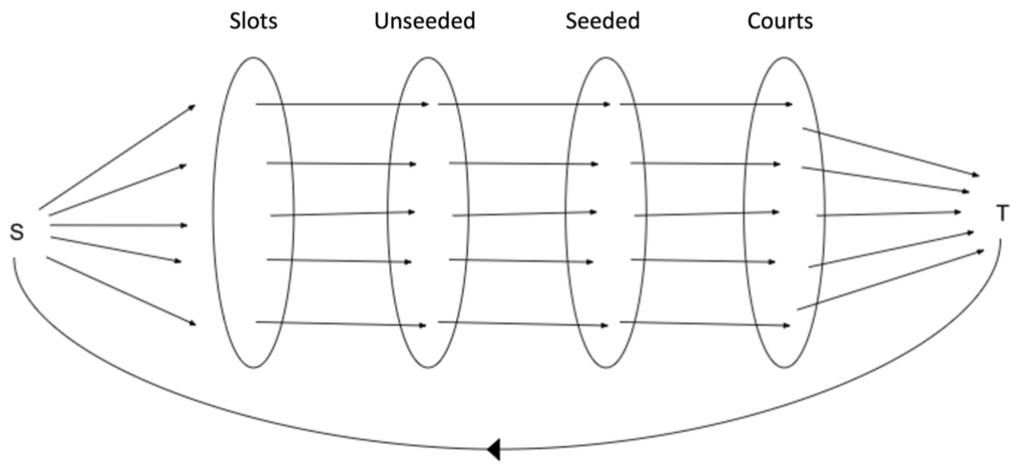


Figure 5

Capacities (2 pts):
1 for all except [0.5 pt]

- r for Slots-T edges [0.5 pt]
- p for S-Courts edges, [0.5 pt]
- infinity or N for the back-edge. [0.5 pt]

Lower bounds (1 pt):

- 1 for Slots-T edges [0.5 pt]
- N for the back-edge [0.5 pt]

Claim (1 pt):
First round pairings with feasible schedule exist if feasible circulation exists. [1 pt]

- If you design the network with N demands for S, T specified, you can also claim that the feasible schedule exist if the maximum flow in G' has the value of N.
- Otherwise, all the other claims saying that maximum flow value should be N in $S - T$ network without specifying the fixed demand as N is wrong.

Note:

(a) The back-edge from T to S with $[N, \text{inf})$ capacity [2 pts in total] can be equal to the design of giving demand $-N$ to S [1 pt] and N to T [1 pt]

(b) *To avoid the case where the players can play multiple games simultaneously, You could add an auxiliary node (mappings for each player) between courts and seeded and another auxiliary node (for each player) between unseeded and slots, to ensure that a player can play only one match at a given time.*

(c) An alternative solution shown with the corresponding arrangement is also correct (See Fig. 5).

ALTERNATE SOLUTION FOR THE MODIFIED (EASIER) VERSION
For the modified version after removing the N' constraint (i.e. All players can play with all other players), we just don't have the backwards edge from T to S. For this modification, the below alternative solution is also correct.

- Node Representation and Network Initialization
  (a) Create a source node S and a sink node T.

(b) Create a set P for players, a set C for courts, and a set T for timeslots.

(c) Create directed edges from S to each player $p_i \in P$ with capacity 1 (representing one match each player will play).

- Surface quality constraint

  (a) Introduce auxiliary nodes between each court c_j and sink T.

  (b) Add an edge from each court auxiliary node to T with capacity p, and another edge to each court-timeslot pair node with lower bound 0 and upper bound 1. This ensures that atmost p matches are played on a single court.

- Player to Court-Timeslot Pairing

  (a) For each player p_i in the top half, link to every court-timeslot pair (c_j, t_k) that is not in their exclusion list $m^{'}$ with capacity 1.

  (b) For each player p_i in the bottom half, link to every court-timeslot pair (c_j, t_k) that is in their preference list $k^{'}$ with capacity 1.

- Competitiveness Constraint

  (a) When connecting players to court-timeslot pairs, ensure that the seed difference between paired top and bottom players is at most N.

- Broadcasting Constraint

  (a) Introduce auxiliary nodes between each timeslot t_k and sink T.

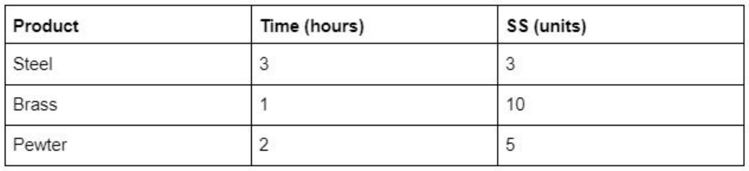
  (b) Add an edge from each timeslot node to T with capacity $r1$, and another edge to each court-timeslot pair node with capacity 1. This ensures at least one match and at most r matches in each timeslot.

- Run Max-Flow Algorithm

  (a) Run the Max-Flow algorithm on this network. If the maximum flow is N, then it is possible to arrange N matches as needed, making the scheduling feasible.

- If the maximum flow equals N, retrieve the flow paths to reconstruct the schedule. Otherwise, output "Scheduling Not Feasible".

3. A company is currently trying to fill a large order of steel, brass, and pewter, measured in tons. Manufacturing each ton of material requires a certain amount of time, and a certain amount of special substance (SS), both of which are limited and are given in below table. Note that it is acceptable to manufacture a fraction of a ton (e.g. 0.5t) of material. Specifically, the company currently has 8 hours of time, and 20 units of special substance (SS). Manufacturing each ton of the three products requires:



Product	Time (hours)	SS (units)
Steel	3	3
Brass	1	10
Pewter	2	5

Figure 6: Table describing the amount of time taken and special substance (SS) required for each product.

(a) Write down a linear program that determines the maximum amount of products (in tons) that the company can make.

Maximize:

$$s + b + p \tag{1}$$

Subject to:

$$3s + b + 2p \leq 8 \tag{2}$$

$$3s + 10b + 5p \leq 20 \tag{3}$$

$$s, b, p \geq 0 \tag{4}$$

(b) Due to the potential danger posed by the special substance (SS), the company would like to use up as much of its supply of special substance (SS) as possible, while:

i. spending at most 8 hours

ii. manufacturing a total of at least 2 tons of steel plus pewter.

Maximize:

$$3s + 10b + 5p \tag{5}$$

Subject to:

$$3s + b + 2p \leq 8 \tag{6}$$

$$3s + 10b + 5p \leq 20 \tag{7}$$

$$s + p \geq 2 \tag{8}$$

4. Suppose that a cement supplier has two warehouses, one located in city A and another in city B. The supplier receives orders from two customers, one in city C and another in city D. The customer in city C needs at least 50 tons of cement, and the customer in city D needs at least 60 tons of cement. The amount of cement at the warehouse in city A is 70 tons, and the number of units at the warehouse in city B is 80 tons. The cost of shipping each ton of cement from A to C is 1, from A to D is 2, from B to C is 3, and from B to D is 4.

   Formulate the problem of deciding how many tons of cement from each warehouse should be shipped to each customer to minimize the total shipping cost as a linear programming. You can assume that the values of units to be shipped are real numbers.

   Let x_1 and x_2 represent the number of units to be shipped from A to C and to D, respectively, and x_3 and x_4 represent the number of units to be shipped from B to C and to D, respectively. Then, the given problem can be formulated as the following linear program:

   minimize:

$$x_1 + 2x_2 + 3x_3 + 4x_4 \tag{9}$$

   subject to:

$$x_1 + x_3 \geq 50 \tag{10}$$

$$x_2 + x_4 \geq 60 \tag{11}$$

$$x_1 + x_2 \leq 70 \tag{12}$$

$$x_3 + x_4 \leq 80 \tag{13}$$

$$x_1, x_2, x_3, x_4 \geq 0 \tag{14}$$

5. Write down the dual program of the following linear program. There is no need to provide intermediate steps.

$$\max(x_1 - 3x_2 + 4x_3 - x_4)$$

subject to

$$\begin{aligned} x_1 - x_2 - 3x_3 &\leq -1 \\ x_2 + 3x_3 &\leq 5 \\ x_3 &\leq 1 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

solution:

$$\min(-y_1 + 5y_2 + y_3)$$

subject to

$$\begin{aligned} y_1 &\geq 1 \\ -y_1 + y_2 &\geq -3 \\ -3y_1 + 3y_2 + y_3 &\geq 4 \\ y_1, y_2, y_3 &\geq 0 \end{aligned}$$

We also have a trivial constraint $0 \geq -1$.

6. Given an undirected graph $G = (V, E)$, a vertex cover is a subset of V so that every edge in E has at least one endpoint in the vertex cover. The problem of finding a minimum vertex cover is to find a vertex cover of the smallest possible size. Formulate this problem as an integer linear programming problem.

   solution:

   For every vertex $v \in V$, introduce a variable x_v and consider the following integer linear program:

$$\min \sum_{v \in V} x_v$$

   subject to

$$\begin{aligned} x_u + x_v \geq 1, &\quad \forall (u, v) \in E \\ x_v \in \{0, 1\}, &\quad \forall v \in V \end{aligned}$$

   In the following, we prove that G has a vertex cover of size k if only if the above linear program with the objective being k. Suppose the G has a vertex cover A of size k. Set $x_v = 1$ if $v \in A$ and $x_v = 0$ if $v \notin A$. Since A is a vertex cover, for every edge $(u, v) \in E$, either $u \in A$ or $v \in A$, which ensures that $x_u + x_v \geq 1$. Hence all the constraints in the integer linear program are satisfied, which shows that x obtained from A is a feasible solution. Moreover, the objective is $\sum_{v \in V} x_v = |A| = k$.

   Conversely, assume that there exists a solution x' (that is, each x'_v is assigned a 0/1 integer such that all the constraints are satisfied) to the integer linear program such that $\sum_{v \in V} x'_v = k$. If $x'_v = 1$, we add v to set B. For every edge $(u, v) \in E$, since $x'_u + x'_v \geq 1$, either $x'_u \geq 1$ or $x'_v \geq 1$. Thus for every edge $(u, v) \in E$, either $u \in B$ or $v \in B$, which implies that B is a vertex cover of G. Moreover, the size of B is $\sum_{v \in V} x'_v = k$

   Thus G has a vertex cover of size k if and only if the corresponding integer linear program has a solution where the objective is k.

7. Assume that you are given a polynomial time algorithm that given a 3-SAT instance decides in polynomial time if it has a satisfying assignment. Describe a polynomial time algorithm that finds a satisfying assignment (if it exists) to a given 3-SAT instance.

To find a satisfying assignment (if it exists) for a given 3-SAT instance, we can use the following polynomial time algorithm:

(a) Start with an arbitrary assignment of truth values to the Boolean variables in the formula.

(b) For each clause in the formula that is not yet satisfied, choose an unsatisfied literal in the clause and set its value to true. If the literal is negative, set its corresponding variable to false. If the literal is positive, set its corresponding variable to true.

(c) Repeat step 2 until all clauses are satisfied or it is not possible to satisfy any more clauses.

(d) If all clauses are satisfied, then the current assignment is a satisfying assignment for the 3-SAT instance. Otherwise, the 3-SAT instance does not have a satisfying assignment.

The time complexity of this algorithm is $O(nm)$, where n is the number of variables in the 3-SAT instance and m is the number of clauses. Since n and m are both polynomial in the size of the input, this algorithm runs in polynomial time.

8. The graph five-coloring problem is stated as follows: Determine if the vertices of G can be colored using 5 colors such that no two adjacent vertices share the same color. Prove that the five-coloring problem is NP-complete.

   Hint: You can assume that graph 3-coloring is NP-complete

   Solution:

   Show that 5-coloring is in NP:

   Certificate: a color solution for the network, i.e., each node has a color.

   Certifier:

   (a) Check for each edge (u,v), the color of node u is different from the color of node v.
   (b) Check at most 5 colors are used.

   Show that 5-coloring is NP-hard we need to prove that 3-coloring $\leq_P$ 5-coloring

   Graph construction:

   (a) Given an arbitrary graph G. Construct G' by adding 2 new nodes u and v to G.
   (b) Connect u and v to all nodes that existed in G, and to each other.

   G' can be colored with 5 colors iff G can be colored with 3 colors.

   (a) If there is valid 3-color solution for G, say using colors 1,2,3, we want to show there is a valid 5-coloring solution to G'. We can color G' using five colors by assigning colors to G according to the 3-color solution, and then color node u and v by additional two different colors. In this case, node u and v have different colors from all the other nodes in G', and together with the 3- coloring solution in G, we use at most 5 colors to color G'.

(b) If there is a valid 5-coloring solution for G', we want to show there is a valid 3- coloring solution in G. In G', since node u and v connect to all the other nodes in G and to each other, the 5-coloring solution must assign two different colors to node u and v, say colors 4 and 5. Then the remaining three colors 1,2,3 are used to color the remaining graph G and form a valid 3-color solution.

Solving 5-coloring to G' is as hard as solving 3-color to G, then 5-coloring problem is at least as hard as 3-coloring, i.e., 3-coloring $\leq_P$ 5-coloring.

9. Longest Path is the problem of deciding whether a graph $G = (V, E)$ has a simple path of length greater or equal to a given number k. Prove that the Longest path Problem is NP-complete by reduction from the Hamiltonian Path problem.

   solution: The Longest Path Problem is in NP. Given a solution path P, we just check that P consists of at least k edges, and that these edges form a path (where no vertex is used more than once). This verification can be done in polynomial time. The reduction follows directly from Hamiltonian Path. Given an instance of Hamiltonian Path on a graph G = (V, E). We create an instance of the longest path problem G' as follows. We use exactly the same graph, i.e G' = G and we set k = V-1. Then there exists a simple path of length k in G' iff G' contains a Hamiltonian path. The proof is obvious.

10. There are a set of courses in USC, each of them requiring a set of disjoint time intervals. For example, a course could require the time from 9am to 11am and 2pm to 3pm and 4pm to 5pm. You want to know, given a number K, if it's possible to take at least K courses. Since you want to study hard and take courses carefully, you can only take one course at any single point in time (i.e. any two courses you choose can't overlap). Show that the problem is NP-complete, which means that choosing courses is indeed a difficult thing in our life. Use a reduction from the Independent set problem.

    Solution:

    (a) The solution of the problem can be verified in polynomial time (just check the number of the courses in the solution is larger or equal to K, and they don't have time overlap), thus it is in NP.

    (b) Given an independent set problem, suppose the graph has n nodes and asks if it has an independent set of size at least M. Establish an injection $f : V \times V \to \mathbb{N}$

    s.t.

    $$f(v_i, v_j) = \begin{cases} i * n + j, & \text{if } i \leq j, \\ f(v_j, v_i), & \text{otherwise.} \end{cases}$$

    Now we construct an instance of the course choosing problem, each course corresponds to a vertex of the graph, and if there exists an edge (v_i, v_j) in the original graph, we let the i-th courses require the $f(v_i, v_j)$-th hour. The problem is to determine whether we can choose M courses.

    Notice that, if there exists an edge (v_i, v_j) in the original graph, then the i-th course and the j-th course will jointly require the $f(v_i, v_j)$-th hour, which means that we can't choose these two courses at the same time.

    If the Independent problem is a "yes" instance (has an independent set of size at least M), then we can choose the corresponding courses, and they don't overlap. For the other direction, if we can choose the corresponding courses, then it follows that the independent set problem is a “yes” instance. Thus we can reduce the independent set problem to the course choosing problem in polynomial time. Since independent set problem is NP-Complete, so the course choosing problem is in NP-Hard.

Thus the course choosing problem is NP-Complete.