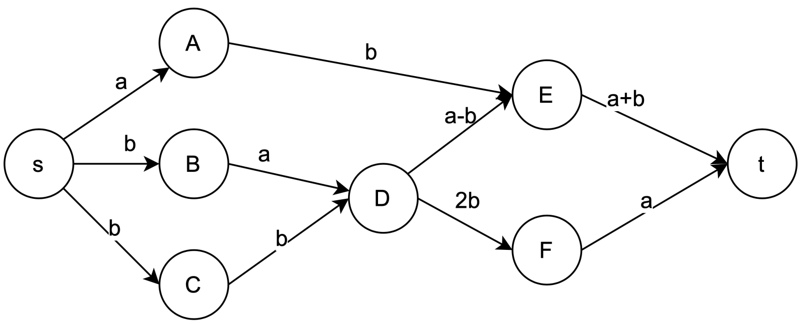
Mid-term 2, Part 1

Started: Nov 11 at 9:40am

Quiz Instructions

Question 1 2 pts

For the network shown below, it's known that $a > 2b$. Which of the following statements is true?



- minimum s-t cut is unique
- value of the maximum s-t flow is $2a + b$
- value of the maximum s-t flow is $a + 2b$
- maximum s-t flow is not unique

Question 2 2 pts

If we can prove NP = co-NP, then this also implies that P = NP.

- True
- False

Question 3 — 2 pts

3SAT problem is PSPACE-complete.

- Above statement has been disproved.
- Above statement has been proved.
- Above statement remains open.

Question 4 — 2 pts

Let (S, T) be one minimum s-t cut with m cut-edges. After increasing the edge capacity by 1 for every edge in the network, the value of the maximum flow will be increased by at least m.

- False
- True

Question 5 — 2 pts

Let $\Phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$.

- $\exists x_1 \forall x_2 \exists x_3 \Phi(x_1, x_2, x_3)$ is false.
- $\exists x_1 \forall x_2 \exists x_3 \Phi(x_1, x_2, x_3)$ is true.

Question 6

2 pts

Let f_1 and f_2 be two distinct maximum s-t flow of a network G with s being the source vertex. Let $S_1 = \{v \in V \mid s\ can\ reach\ v\ in\ the\ residual\ graph\ w.r.t.\ \ f_1\}$, and let $S_2 = \{v \in V \mid s\ can\ reach\ v\ in\ the\ residual\ graph\ w.r.t.\ \ f_2\}$. Which one of the following is always true?

- $S_1 \cap S_2 = \{s\}$
- $S_1 \neq S_2$
- None of the other choices is always true
- $S_1 = S_2$

Question 7

2 pts

If we can solve the 3SAT problem in polynomial-time, then this implies that integer factorization can also be solved in polynomial-time.

- True
- False

Question 8

2 pts

Assume that problem A is NP-complete and problem B is in P, which one of the following problem C must be solvable in polynomial time? ($X \leq_p Y$ means X is polynomial-time reducible to Y.)

- a problem C satisfying $C \leq_p A$
- a problem C satisfying $A \leq_p C$
- a problem C satisfying $C \leq_p B$
- a problem C satisfying $B \leq_p C$

Question 9

2 pts

Considering running the preflow-push algorithm on a network. Let f be the preflow and h be the labeling throughout the running. Assume s and t are the source and sink vertices of the network. Which one of the following is always true?

- $h(s) \geq h(v)$ for every $v \neq s$ at any time of the running.
- At the time that f becomes a flow, f and h will not be compatible.
- After performing a push operation on edge (u, v), the excess of u will become 0.
- In the residual graph w.r.t. f, there is no path from s to t.

Question 10

2 pts

Let G be an undirected graph. Let M be one maximum matching of G. Let V_1 be one minimum vertex cover of G. Then we have $|V_1| = |M|$ if and only if G is a bipartite graph.

- False
- True

Not saved

Submit Quiz

Problem 1 (2 points).

- False. The minimum cut is not unique: $(S = \{s, A\}, T = V \setminus S)$ and $(S = \{s, A, C\}, T = V \setminus S)$ are two different minimum cut.
- False. The value of maximum flow is $3b$.
- False. The value of maximum flow is $3b$.
- True. There are different ways to distribute the $2b$ in-flow that goes to vertex D.

Problem 2 (2 points).

False. Note: the reverse is true: if P = NP then NP = co-NP.

Problem 3 (2 points).

This statement has not been proved or disproved yet. More specifically, suppose that 3SAT is PSPACE-complete. Then by definition, this implies that every problem in PSPACE is polynomial-time reducible to 3SAT. In other words, every problem can be solved by a nondeterministic Turing machine in polynomial-time. This implies that PSAPCE = NP, which hasn't been proved or disproved. On the other hand, suppose that 3SAT is not PSPACE-complete. Then this implies that NP $\neq$ PSPACE (proof: if NP = PSAPCE, then every problem in PSPACE = NP will be polynomial-time reducible to 3SAT, so 3SAT will be PSPACE-complete), but this hasn't been proved or disproved.

Problem 4 (2 points).

False. This is because, there might exist another minimum cut with n cut-edges, $n < m$. For that minimum cut, its total capability will be increased by n. (In other words, after increasing edge capacities, the cut with m edges is not a minimum cut anymore.) So the value of maximum flow will be increased by at most n.

Problem 5 (2 points).

Note: the problem description is a bit confusing. We meant to ask if the given $\Phi(x_1, x_2, x_3)$ is a true-instance or false-instance of the QSAT problem. True-instance means that $\exists x_1 \forall x_2 \exists x_3$ such that $\Phi(x_1, x_2, x_3)$ is true. If it is not a true-instance, then it is a false-instance.

The given $\Phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$ is a true-instance. This is because, we can set x_1 as false, which satisfies the 2nd and the 4th clauses, then if x_2 is true (which satisfies the 1st clause but not the 3rd clause), then we set x_3 as false to satisfy the 3rd clause, and if x_2 is false (which satisfies the 3rd clause but not the 1st clause), then we set x_3 as true to satisfy the 1st one.

Problem 6 (2 points).

According to Problem 1.2 of Assignment 3, we know that $S_1 = \cap_{C=(S,T)\in C} S$ and also $S_2 = \cap_{C=(S,T)\in C} S$, as $\cap_{C=(S,T)\in C} S$ is independent of the any maximum flow. So we must have $S_1 = S_2$.

Problem 7 (2 points).

True. If one can solve the 3SAT problem in polynomial-time, then we have P = NP. We have proved that the integer factorization problem is in NP (and in co-NP), and therefore will be in P.

Problem 8 (2 points).

Given that problem B is in P, if a problem C satisfies that C is polynomial-time reducible to B, then C can also be solved in polynomial-time.

Problem 9 (2 points).

- False. $h(v)$ can go beyond $h(s)$ at a later state of the preflow-push algorithm (so that the excess of v can be pushed back to s).
- False. The preflow f and the labeling h are guaranteed compatible throughout the entire preflow-push algorith, of course including the time that f becomes a flow.
- False. It depends on whether the excess of u is large or equal to the capacity of (u,v) in the residual graph or not.
- True. We proved in class: as long as there exists a labeling h that is compatible with f, then there is no path from s to t in the residual graph w.r.t. f. And when running the preflow-push algorithm, such a compatible labeling is guaranteed throughout the running.

Problem 10 (2 points).

False. If G is bipartite graph, then yes, $|V_1| = |M|$. If $|V_1| = |M|$ then G is not necessarily a bipartite graph. Example: $V = \{a,b,c,d\}$ and $E = \{(a,b),(b,c),(c,d),(d,a),(a,c)\}$. Then $V_1 = \{a,c\}$ and $M = \{(a,b),(c,d)\}$ but there exists odd-length cycles in G.

Problem 1 (15 points).

You are given a directed graph $G = (V, E)$ with $s, t \in V$ being the source and sink vertices. There is also an integral weight $w(v) > 0$ associated with $v \in V \setminus \{s, t\}$. We say $V_1 \subset V \setminus \{s, t\}$ is a path-cover if every path from s to t must cross at least one vertex in V_1. Design a polynomial-time algorithm to find a path-cover V_1 such that $\sum_{v \in V_1} w(v)$ is minimized. *Hint:* reduce this problem into a maximum-flow problem.

Solution: We can transform this problem to a max-flow problem. We make a new directed graph $G' = (V', E')$ by modifying the given directed graph $G = (V, E)$: for each $v \in V$, we split it into two vertices v_1 and v_2 and add a new directed edge (v_1, v_2); the capacity of this new edge will be set as $w(v)$, i.e., the given weight of vertex v in G; any edge $(u, v) \in E$ will be kept in G' and will become pointing from u_2 to v_1, and their capacities will be set as infinity. In G', vertices won't have weights anymore. We then run any max-flow algorithm on G' to find a minimum s_1-t_2 cut. Let C be the set of cut-edges. Note that C won't include any edge with infinity capacity (as it is a minimum cut). In other words, in only includes newly added edges through splitting. We return the set of vertices in G corresponding to those cut-edges as V_1.

Running time: Splitting the vertices and adding new edges can be done in $O(|V|)$, giving capacity to all edges in $O(|E|)$; finding min-cut can be done in polynomial time. Hence, this algorithm takes polynomial-time.

Correctness: According to the construction of G' from G: V_1 is a path-cover of G if and only if $E(V_1) := \{(v_1, v_2) \mid v \in V_1\}$ in G' satisfies that any path from s_1 to t_2 in G' must overlap with at least one edge in $E(V_1)$. By definition, such $E(V_1)$ corresponds to an s_1-t_2 cut of G' as it separates s_1 and t_2. Because of this one-to-one correspondence, a path-cover V_1 in G that minimizes $\sum_{v \in V_1} w(v)$ corresponds to a minimum s_1-t_2 cut in G' following our approach of assigning capacities.

Problem 2 (6 + 9 = 15 points).

Given a directed graph $G = (V, E)$, we say $V_1 \subset V$ is a cycle-cover if every cycle of G must cross at least one vertex in V_1. Given G and an integer k, the cycle-cover problem is to decide if G contains a cycle-cover of size at most k.

- Show that above cycle-cover problem is in NP. *Hint:* considering using the fact that whether a directed graph contains cycle can be determined in polynomial-time; note too that a directed graph may contain exponential number of cycles.

  Solution: As there might be exponentially many cycles in a graph, we cannot directly enumerate all the cycles as part of the certificate. However, because all the cycles cross at least one vertex in V_1, if we remove all the vertices in v_1 from the graph, there would not be any cycles in the graph anymore. It only takes $O(|V|)$ using a DFS to check if the graph is acylic or not. Then we can design a verifier for cycle-cover problem as following:

  function Verifier ($G = (V, E)$, k, V_1)
  If V_1 is not a subset of V: return No
  If $|V_1| > k$: return No
  $G' =$ remove V_1 from G
  If $cycleCheck(G') = true$: return Yes
  return No;
  end function;

- Prove that above cycle-cover problem is NP-complete. *Hint:* pick the vertex-cover problem in the reduction; considering transform one undirected edge into two directed edges in your reduction.

Solution: We already proved cycle-cover problem is in NP, now we want to show there is polynomial time reduction from the vertex-cover problem to the cycle-cover problem. Given an undirected graph G and an integer k for the vertex-cover problem, we create a new graph G' by replace all the edges $e = (u,v)$ in G with two directed edges (u,v) and (v,u). It can be done in $O(|V|+|E|)$ time.

We now prove that, G has a vertex-cover set of size k, if and only if G' has a cycle-cover set in G of size k. If G has a vertex-cover set V_1 of size k, then same V_1 will be a cycle-cover set for G'. Because every edge in G' will cross one vertex in V_1. Then all the cycles in G' will be covered by V_1. Conversely, if G' has a cycle-cover set of size k, the same set must form a set-cover in G. Suppose not, i.e., in G there is an edge $e = (u,v)$ not covered by the set, then in G', the cycle $(u,v)+(v,v)$ would also not be covered by the set. It implies that the set is not a cycle-cover of G' which is contradicted to our assumption. Therefore, the vertex-cover problem is polynomial time reducible to the cycle-problem, and hence the cycle-cover problem is NP-complete.

Bonus Problem (10 points).

You are given an $n \times n$ checkerboard, and there is a positive integer in each square. You want to pick some of the numbers on the checkerboard to maximize their sum. But if one number is picked, then you can not pick any number on its adjacent squares. Design a polynomial-time algorithm for this problem. *Hint:* reduce this problem into a maximum-flow problem.

Solution: We first partition all squares on the board into two sets, white set and black set, using the same way in the chessboard: a square (i, j) will be classified as white/black if and only if $i+j$ is even/odd. We then build a bipartite graph by creating a vertex for each square and connecting the adjacent squares (adjacent squares must be in different colors). We then build a network (similar to what we did in solving the bipartite-matching problem). We add a new source vertex s and add edges from s to each of square u in the white set (left side) with the number on that square u being the capacity of edge (s,u). We also add a sink vertex t and add edges (v,t) from each square v in the black set (right side) to t with its number associated with v as capacity of edge (v,t). At last, we add edges between adjacent squares (must have different colors) with infinite capacity. We then calculate an minimum s-t cut (S,T) of this network. We then pick the white vertices (left side) that are in S (i.e., the S_2 in the figure) and the black vertices (right side) that are in T (i.e., the T_1 in the figure).

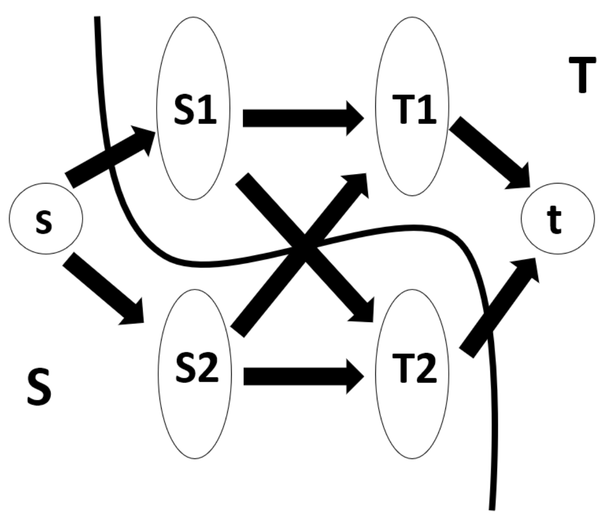


Figure 1: S_2 and T_1 will be picked.

We first show that there is a one-to-one correspondence between a legal pickup (i.e., no adjacent vertices will be picked) of vertices and an *s*-*t* cut with finite capacity. On one hand, if we pick white vertices S_2 and black vertices T_1, there in above network there is no edges from S_2 to T_1, so the *s*-*t* cut $(\{s, S_2, T_2\}, \{S_1, T_1, t\})$ admits a finite capacity. On the other hand, let $(S = \{s, S_2, T_2\}, T = \{S_1, T_1, t\})$ be a cut with finite capacity, then there is no edges from S_2 to T_1, as otherwise the capacity of (S, T) will be infinity, so picking up $S_2 \cup T_1$ is legal.

Now we create their quantitative relation. The capacity of (S, T) will be $c(S, T) = \sum_{u \in S_1} w(u) + \sum_{v \in T_2} w(v)$, which can be further rewritten as $c(S, T) = \sum_{u \in S_1 \cup S_2} w(u) - \sum_{u \in S_2} w(u) + \sum_{v \in T_1 \cup T_2} w(v) - \sum_{v \in T_2} w(v) = C - (\sum_{u \in S_2} w(u) + \sum_{v \in T_1} w(v))$, where C is the sum of the numbers of all vertices, a constant. Therefore an *s*-*t* cut with minimized total capacity gives a legal set of vertices with maximized sum.