CSE 6140 / CX 4140

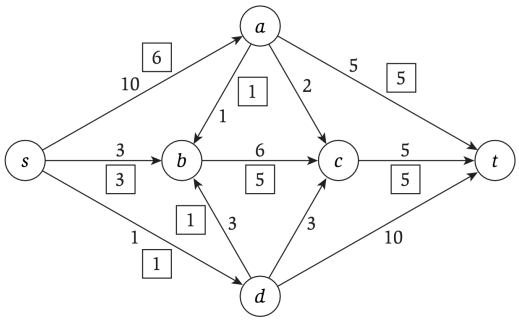
Computational Science & Engineering Algorithms

Homework 5

Please type all answers.

1. (10 points)

The figure below shows a flow network on which an s-t flow has been computed. The capacity of each edge appears as a label next to the edge, and the numbers in boxes give the amount of flow sent on each edge. (Edges without boxed numbers have no flow being sent on them.)



(a) What is the value of this flow? Is this a maximum (s,t) flow in this graph?

Solution: Amount of flow in the network = Amount of flow out of source = 6 + 3 + 1 = 10. From the residual graph G_f (refer to Figure 1), we see that there is still an $s - t$ path and, hence, space to augment flow. Hence, 10 is not the maximum flow in the given network.

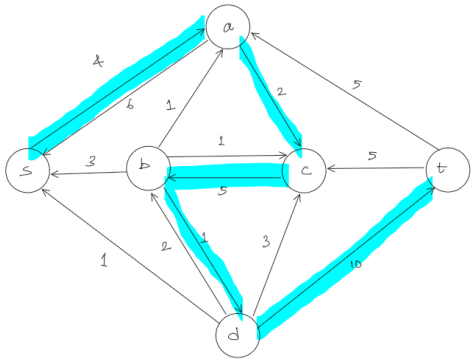
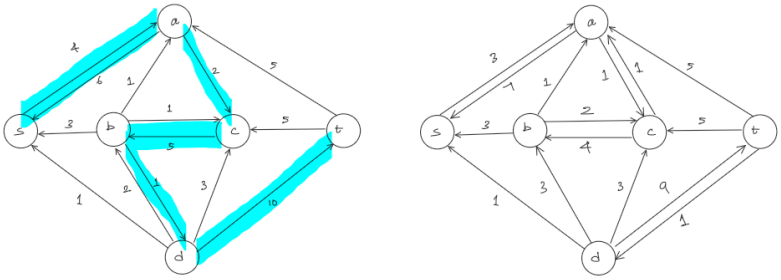


Figure 1: Residual graph G_f for the given flow.

(b) Find a minimum s-t cut in the flow network, and also say what its capacity is.

Solution: First, we find max flow in the network and then apply the Max-flow Min-Cut Theorem. So, we augment the flow along path $P : s - a - c - b - d - t$. The minimum capacity along this path is 1 (edge $b - d$). The edges $s - a$, $a - c$ and $d - t$ are forward edges and thus we increase these edges in G by 1. The edges $c - b$ and $b - d$ are backward edges and thus we reduce the flow on edges $b - c$ and $d - b$ in G by 1. Doing so results in the residual graph on the right in Figure 2. We see that there are no more $s - t$ paths in the residual graph and thus the algorithm stops and thus the maximum flow is 11. By the Max flow Min Cut theorem, **the capacity of the min $s - t$ cut is 11. The min $s - t$ cut is formed by (A, B) where $A = \{s, a, c, b\}$ and $B = V(G) - A$.**



P : s – a – c – b – d – t

min capacity in P = 1

s – a : forward ⟹ add 1 to flow

a – c : forward ⟹ add 1 to flow

c – b : backward ⟹ decrease 1 to flow

b – d : backward ⟹ decrease 1 to flow

d – t : forward ⟹ add 1 to flow

Figure 2: Augmenting the flow and finding max flow

---

2. (10 points) **(a)** Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e. If f is a maximum $s - t$ flow in G, then f saturates every edge out of s with flow (i.e., for all edges e out of s, we have $f(e) = c_e$).

Solution: False. For example, the network in Figure 3 has reached max flow of $f = 8$ but f does not saturate the edges out of s.

(b) Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Let G be an arbitrary flow network, with a source s, a sink t, and a positive integer capacity c_e on every edge e; and let (A, B) be a mimimum $s - t$ cut with respect to these capacities $\{c_e : e \in E\}$. Now suppose we add 1 to every capacity; then (A, B) is still a minimum $s - t$ cut with respect to these new capacities $\{1 + c_e : e \in E\}$.

Solution: False. Refer to Figures 4 and 5, where increasing the capacity of each

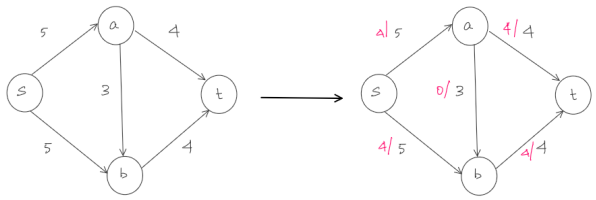


Figure 3: Counterexample with the network with the max flow 8 and the flows out of s is not saturated.

edge by 1 changes the min-cut. The intuition behind this is as follows: Increasing edge capacity by 1 and calculating min $s-t$ cut is equivalent to finding the partition (A, B) of $V(G)$ that minimizes $h(A) = \sum_{e \text{ out of A}} c(e) + \texttt{\#\_edges\_out\_of\_A}$. So, the partition (A, B) that minimizes $g(A) = \sum_{e \text{ out of A}} c(e)$ might not minimize $h(A)$.

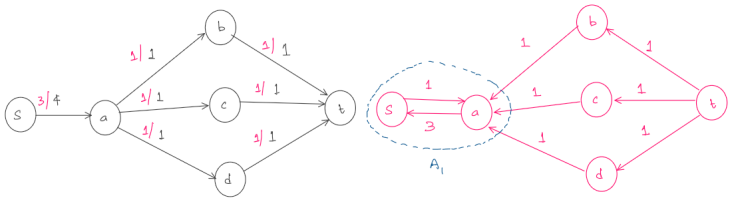


Figure 4: Min $s-t$ cut in G with edge capacities $\{c_e : e \in E(G)\}$ is formed by $(A, V(G) - A)$ where $A = \{s, a\}$

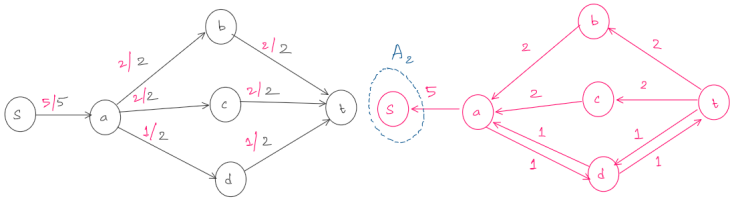


Figure 5: Min $s-t$ cut in G with edge capacities $\{c_e + 1 : e \in E(G)\}$ is formed by $(A, V(G) - A)$ where $A = \{s\}$

---

3. (20 points) Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible base stations. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a range parameter r – a client can only be connected to a base station that is within distance r. There is also a load parameter L – no more than L clients can be connected to any single base station.

Your goal is to design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a

base station, subject to the range and load conditions in the previous paragraph.

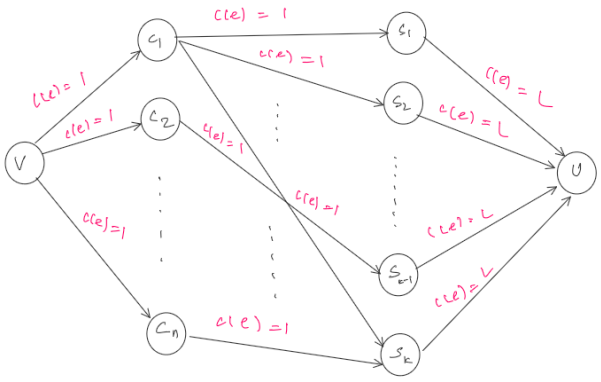


Figure 6: Constructed flow network for the mobile computing problem (edge weights are the capacities of flow)

Solution: STEP 1: Convert this into a maximum flow problem. For the same, we construct the graph as follows:

- For each client i create a node c_i and for every base station j create a node s_j - **done in linear time (linear in number of clients + base stations)**.
- Now, add edge (c_i, s_j) to the graph if and only if the distance between c_i and s_j is at most r - **done in $O(nk)$ time**.
- Next, add node U and add edges (s_j, U) for all j. Next, add node V and add edges (V, c_i) for all i - **done in linear time (linear in number of clients + base stations)**.
- Now, $G := (V, E)$ where $V = \{c_i | i = 1, 2, \ldots, n\} \bigsqcup \{s_j | j = 1, 2, \ldots, k\} \bigsqcup \{U, V\}$ forms the required flow graph.

STEP 2: Assign the edge capacities to satisfy the connection requirements:

- Capacity of each (c_i, s_j) edge is 1 indicating at most one connection can be made by a client with a base station - **done in linear time in number of edges of type** (c_i, s_j).
- Capacity of each (s_j, U) edge is L since each base station can provide at most L connections - **done in linear time (linear in number of base stations)**.
- Capacity of each (V, c_i) edge is 1 - **done in linear time (linear in number of clients)**.

STEP 3: Run the Ford-Fulkerson algorithm on the above-constructed network (Refer to Figure 6). There is a feasible way to connect all the n clients to base stations if and only if the maximum flow value is n:

- If the max-flow value is $n \Rightarrow$ all the (V, c_i) edges are saturated (since n edges each with capacity 1) $\Rightarrow$ flow out of each c_i is 1 (no demands so flow into c_i = flow out of c_i) $\Rightarrow$ each client gets connected to a base station without overloading any stations (total flow out of the base stations is n and without having the flow out of each station $< L$, we wouldn't have a valid flow).
- If there is a feasible way to connect the n clients to base stations, we add a flow of 1 on each edge (c_i, s_j) if the client c_i connects to station s_j in the feasible assignment. Next, assign a flow of 1 on each (V, c_i) edge to satisfy flow conditions in the network. Also, add flows to the (s_j, U) edges equal to the number of clients c_i the station s_j has a connection - this ensures that flow into s_j = flow out of s_j. Since the assignment is such that no station is overloaded, the flow on each (s_j, U) edge is at most L. So, we have a valid flow and the flow value is n since all the edges from V to c_i are saturated at flow 1. Note that this will be the maximum value since all the edges from the source V has saturated. Thus, maximum flow value is n.

Thus, we have established the equivalence. The total time for solving the problem is the sum of the time to convert the original problem to a flow network ($O(nk)$, which is polynomial time, and the time to run Ford-Fulkerson algorithm on the flow network with $O(n + k)$) nodes and $O(nk + n + k)$ edges. Thus, the provided algorithm is a polynomial time algorithm.

---

4. (20 points)

Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time.

Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now).

At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most $\lceil n/k \rceil$ people.

Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

Solution: STEP 1: Convert this into a maximum flow problem. For the same, we construct the graph as follows:

- For each injured person i create a node p_i and for every hospital j create a node

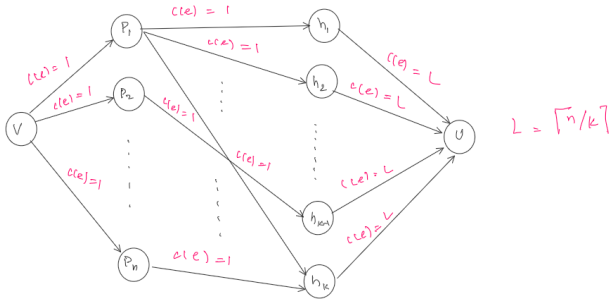


Figure 7: Constructed flow network for the injured persons problem (edge weights are the capacities of flow)

h_j - **done in linear time (linear in number of patients + hospitals)**.

- Now, add edge (p_i, h_j) to the graph if and only if the hospital h_j can be reached by p_i within half-hour - **done in $O(nk)$ time**.
- Next, add node U and add edges (h_j, U) for all j. Next, add node V and add edges (V, p_i) for all i - **done in linear time (linear in number of patients + hospitals)**.
- Now, $G := (V, E)$ where $V = \{p_i | i = 1, 2, \ldots, n\} \bigsqcup \{h_j | j = 1, 2, \ldots, k\} \bigsqcup \{U, V\}$ forms the required flow graph.

STEP 2: Assign the edge capacities to satisfy the connection requirements:

- Capacity of each (p_i, h_j) edge is 1 indicating a patient can go to at most one hospital - **done in linear time in number of edges of type** (p_i, h_j).
- Capacity of each (h_j, U) edge is $\lceil n/k \rceil$ since each hospital can treat with at most $\lceil n/k \rceil$ patients - **done in linear time (linear in number of hospitals)**.
- Capacity of each (V, p_i) edge is 1 - **done in linear time (linear in number of patients)**.

STEP 3: Run the Ford-Fulkerson algorithm on the above-constructed network (Refer to Figure 7). There is a feasible way to take all the n patients to one of the k hospitals if and only if the maximum flow value is n:

- If the max-flow value is $n \Rightarrow$ all the (V, p_i) edges are saturated (since n edges each with capacity 1) $\Rightarrow$ flow out of each p_i is 1 (no demands so flow into p_i = flow out of p_i) $\Rightarrow$ each patient reaches a hospital (proximity taken care of by edge existence) without overloading any hospital (total flow out of the hospitals is n and without having the flow out of each station $< \lceil n/k \rceil$, we wouldn't have a valid flow).

- If there is a feasible way to take all the n patients to a hospital within 30 minutes, we add a flow of 1 on each edge (p_i, h_j) if the patient p_i is taken to hospital h_j in the feasible assignment. Next, assign a flow of 1 on each (V, p_i) edge to satisfy flow conditions in the network. Also, add flows to the (h_j, U) edges equal to the number of patients p_i the hospital h_j is treating - this ensures that flow into h_j = flow out of h_j. Since the assignment is such that no hospital is overloaded, the flow on each (h_j, U) edge is at most $\lceil n/k \rceil$. So, we have a valid flow and the flow value is n since all the edges from V to p_i are saturated at flow 1. Note that this will be the maximum value since all the edges from the source V have saturated. Thus, the maximum flow value is n.

Thus, we have established the equivalence. The total time for solving the problem is the sum of the time to convert the original problem to a flow network ($O(nk)$), which is polynomial time, and the time to run the Ford-Fulkerson algorithm on the flow network with $O(n + k)$ nodes and $O(nk + n + k)$ edges. Thus, the provided algorithm is a polynomial time algorithm.

---

5. (20 points)

Consider a large-scale atmospheric science experiment. You need to get good measurements on a set S of n different conditions in the atmosphere (such as the ozone level at various places), and you have a set of m balloons that you plan to send up to make these measurements. Each balloon can make at most two measurements. Unfortunately, not all balloons are capable of measuring all conditions, so for each balloon $i = 1, ..., m$, they have a set S_i of conditions that balloon i can measure. Finally, to make the results more reliable, you plan to take each measurement from at least k different balloons. (Note that a single balloon should not measure the same condition twice.)

Example. Suppose that $k = 2$, there are $n = 4$ conditions labeled $c1, c2, c3, c4$, and there are m =4 balloons that can measure conditions, subject to the limitation that $S1 = S2 = \{c1, c2, c3\}$, and $S3 = S4 = \{c1, c3, c4\}$. Then one possible way to make sure that each condition is measured at least $k = 2$ times is to have

- balloon 1 measure conditions $c1, c2$,
- balloon 2 measure conditions $c2, c3$,
- balloon 3 measure conditions $c3, c4$, and
- balloon 4 measure conditions $c1, c4$.

(a) Give a polynomial-time algorithm that takes the input to an instance of this problem (the n conditions, the sets S_i for each of the m balloons, and the parameter k) and decides whether there is a way to measure each condition by k different balloons, while each balloon only measures at most two conditions.

(b) Suppose each of the balloons is produced by one of three different sub-contractors involved in the experiment. A requirement of the experiment is that there be no condition for which all k measurements come from balloons produced by a single subcontractor.

For example, suppose balloon 1 comes from the first subcontractor, balloons 2 and 3 come from the second subcontractor, and balloon 4 comes from the third subcontractor. Then our previous solution no longer works, as both of the measurements for condition c_3 were done by balloons from the second subcontractor. However, we could use balloons 1 and 2 to each measure conditions c_1, c_2, and use balloons 3 and 4 to each measure conditions c_3, c_4.

Explain how to modify your polynomial-time algorithm for part (a) into a new algorithm that decides whether there exists a solution satisfying all the conditions from (a), plus the new requirement about subcontractors.

Solution: (a)

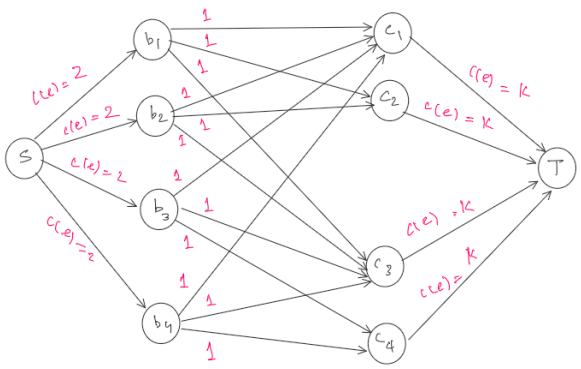


Figure 8: Constructed flow network for the Balloon experiment problem without contractors constraint (edge weights are flow capacities)

STEP 1: Construct a flow network equivalent to the given problem. First, construct the graph as follows:

- For each balloon i we a create a node b_i and for each condition j to be measured, we create a node c_j - **done in** $O(n+m)$ **time**.
- Add source node S and a target node T - **done in constant time**.
- Add edge (b_i, c_j) if and only if $c_j \in S_i$ (i.e., condition j can be measured by balloon i) - **done in** $O(nm)$ **time**.
- Add edges (S, b_i) for all i and edges (c_j, T) for all j - **done in** $O(n+m)$ **time**.

STEP 2: Assign the edge capacities:

- Each edge (S, b_i) has capacity of 2, since each balloon i can measure at most two conditions - **done in** $O(m)$ **time**.

- Each edge (c_j, T) has capacity of k, since we wish to take k different measurements of the condition c_j - **done in $O(n)$ time**.
- Each edge (b_i, c_j) has capacity of 1 since a balloon b_i can take at most one measurement of the condition c_j - **done in $O(nm)$ time**.

STEP 3: Run the Ford-Fulkerson algorithm on the above-constructed network (Refer to Figure 8). There is a feasible way to measure all the n conditions by k different balloons with each balloon measuring at most 2 conditions if and only if the maximum flow value is nk:

- If max flow is nk, then all the edges to T are saturated with flow on each edge equal to $k \Rightarrow$ Each condition j has been measured by k different balloons without crossing the limit of each balloon measuring 2 conditions (else we would not have a valid flow) $\Rightarrow$ there is a way to get k different measurements of each condition with each balloon measuring 2 conditions.
- If there is a way to get k different measurements of each condition with each balloon measuring 2 conditions, then assign flow 1 to each edge (b_i, c_j) if balloon i is measuring condition j. Next, assign flow of k to each edge (c_j, T) since each condition has k different measurements. Note that this also satisfies the network constraint that flow into c_j = flow out of c_j. Next, assign a flow of 2 to each edge (S, b_i) since each balloon measures 2 conditions, and also satisfying that network flow constraint. With the flows assigned, we observe that the flow value in the network is equal to the flow into node T, which is nk. This value is also the maximum flow value possible since the edges to node T have saturated and thus no more $S-T$ augmenting paths available in the residual graph.

Thus, we have established the equivalence. The total time for solving the problem is the sum of the time to convert the original problem to a flow network ($O(nm)$), which is polynomial time, and the time to run the Ford-Fulkerson algorithm on the flow network with $O(n+m)$ nodes and $O(nm+n+m)$ edges. Thus, the provided algorithm is a polynomial time algorithm.

(b)

In this case, we modify the network as follows:

- Delete all the edges of type (b_i, c_k) - **done in $O(nm)$ time**.
- For each subcontractor j, add nodes $p_{j,k}$ ($j = 1, 2, 3$ and $k = 1, 2, \ldots, n$)- **done in linear time ($O(n)$)**.
- Add edge $(p_{j,k}, c_k)$ for all conditions k, with edge capacity $(k-1)$ (to ensure that not all k measurements of a condition k come from balloons of same subcontractor) - **done in $O(n)$ time**.

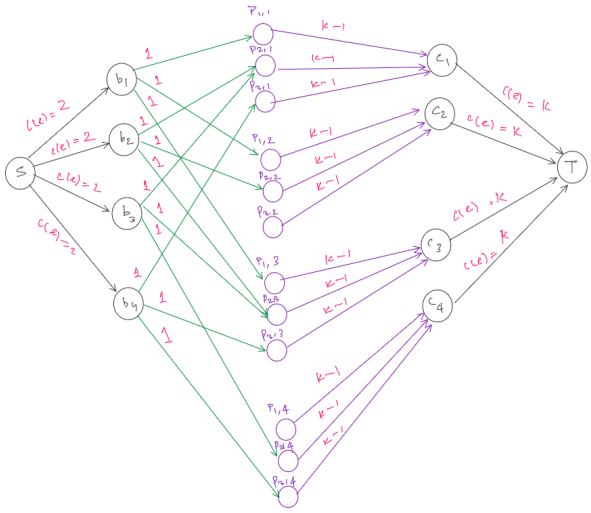


Figure 9: Constructed flow network for the Balloon experiment problem with contractors constraint (edge weights are flow capacities)

- Add edge $(b_i, p_{j,k})$ if and only if:
  - Balloon i is produced by subcontractor j and
  - Condition k can be measured by balloon b (i.e., $c_k \in S_i$)

  For each of these edges, the capacity is 1 since the balloon either measures the condition k or not - **done in** $O(nm)$ **time**.

Now, run the Ford-Fulkerson algorithm on this modified network (Refer to Figure 9). There exists a feasible way to measure all the n conditions measured by k different balloons (produced by different subcontractors) with each balloon measuring 2 conditions if and only if the maximum flow found is equal to nk:

- If max flow is nk, then all the edges to T are saturated with flow on each edge equal to $k \Rightarrow$ Each condition j has been measured by k different balloons without crossing the limit of each balloon measuring 2 conditions (else we would not have a valid flow) as well as all k are not by same contractor since the edge capacity is $(k-1)$ as per construction $\Rightarrow$ there is a way to get k different measurements (by balloons of different contractors) of each condition with each balloon measuring 2 conditions.
- If there is a way to get k different measurements (by balloons of different contractors) of each condition with each balloon measuring 2 conditions, then assign flow 1 to each edge $(b_i, p_{j,k})$ if balloon i is measuring condition k. Next, assign a flow for edges $(p_{j,k}, c_k)$ such that the flow into $p_{j,k}$ = flow out of $p_{j,k}$. Note that this flow value will be the number of balloons by subcontractor j measuring condition k. Due to feasibility, the flow value is at $< k$.

  Next, assign a flow of k to each edge (c_j, T) since each condition has k different measurements. Note that this also satisfies the network constraint that flow

into c_j = flow out of c_j. Finally, assign a flow of 2 to each edge (S, b_i) since each balloon measures 2 conditions, and also satisfying that network flow constraint. With the flows assigned, we observe that the flow value in the network is equal to the flow into node T, which is nk. This value is also the maximum flow value possible since the edges to node T have saturated and thus no more $S - T$ augmenting paths available in the residual graph.

Thus, we have established the equivalence. The total time for solving the problem is the sum of the time to convert the original problem to a flow network ($O(nm)$), which is polynomial time, and the time to run the Ford-Fulkerson algorithm on the flow network with $O(n + m)$ nodes and $O(nm + n + m)$ edges. Thus, the provided algorithm is a polynomial time algorithm.
