Solutions to Practice Problems 9

Solution 1:

(a) The network G' is the same as G, but with the following modifications. For each vertex v with a nonzero leakage capacity $h(v)$, we create an edge (v, t) and assign it capacity $h(v)$ (see the dashed edges in Fig. 1(b)). (This is a multi-digraph, since we may generate duplicate edges, but we can always merge edges and combine their capacities.)

To see that this is correct, suppose that there is a leaky flow f in G. For each vertex v, let $g(v) = f^{\text{in}}(v) - f^{\text{out}}(v)$ denote the amount of leakage. By definition of leaky flows, we know that $0 \leq g(v) \leq h(v)$. We assign flow of $g(v)$ to the edge (v, t) (see Fig. 1(d)). Call the resulting flow f' in G'. This additional flow compensates for the leaked flow at u, implying that f' satisfies flow conservation. Since the leakage cannot exceed $h(v)$, this flow satisfies the capacity constraint. The converse is similar.

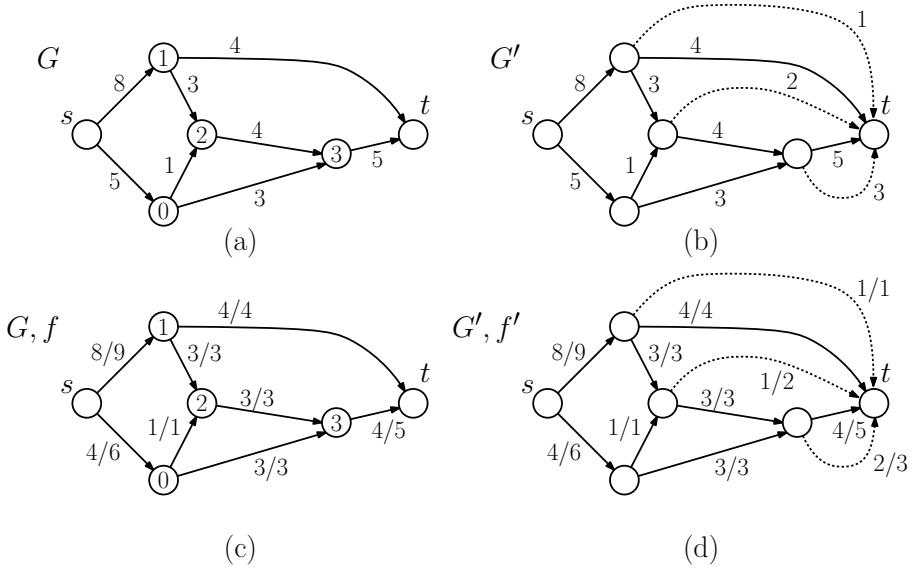

Figure 1: (a) A leaky network G, (b) the transformed (non-leaky) network G', (c) a leaky flow in G, and (d) the equivalent non-leaky flow in G'.

(b) See Fig. 1(b).

Solution 2: Create an s-t network by making the root node r the source and creating a super-sink node t, which will have edges coming from all the terminal nodes $t_i \in T$. Observe that if $C = (X, Y)$ is any cut of the resulting s-t network, then removing the edges that cross the cut from X to Y separates r from all the terminals. Therefore, this problem is equivalent to computing a minimum weight cut in this network.

We can reduce this to a network flow problem. First, set all the capacities of all the nodes of the network to 1 except the edges that enter t to capacity ∞, or more practically, any numeric value that is larger than the maximum possible flow (see Fig. 2). Now, run any network flow algorithm on the resulting network. Let f denote the resulting flow.

We compute the minimum cut as follows. First, construct the residual graph G_f, and determine the subset of nodes X that are reachable from r, and let $Y = V \setminus X$ be the remaining nodes. Note that none of the nodes of T can be reachable from r in G_f, for otherwise we could push more flow through this terminal into t. Therefore, $T \subseteq Y$. By the max-flow/min-cut theorem, the edges (x, y) of G that cross the cut (that is, where $x \in X$ and $y \in Y$) define the minimum cut in G. Since these edges are of capacity 1, the capacity of this cut is equal to the number of edges in the cut. Therefore, this is the minimum number of edges needed to separate r from all the vertices of T.

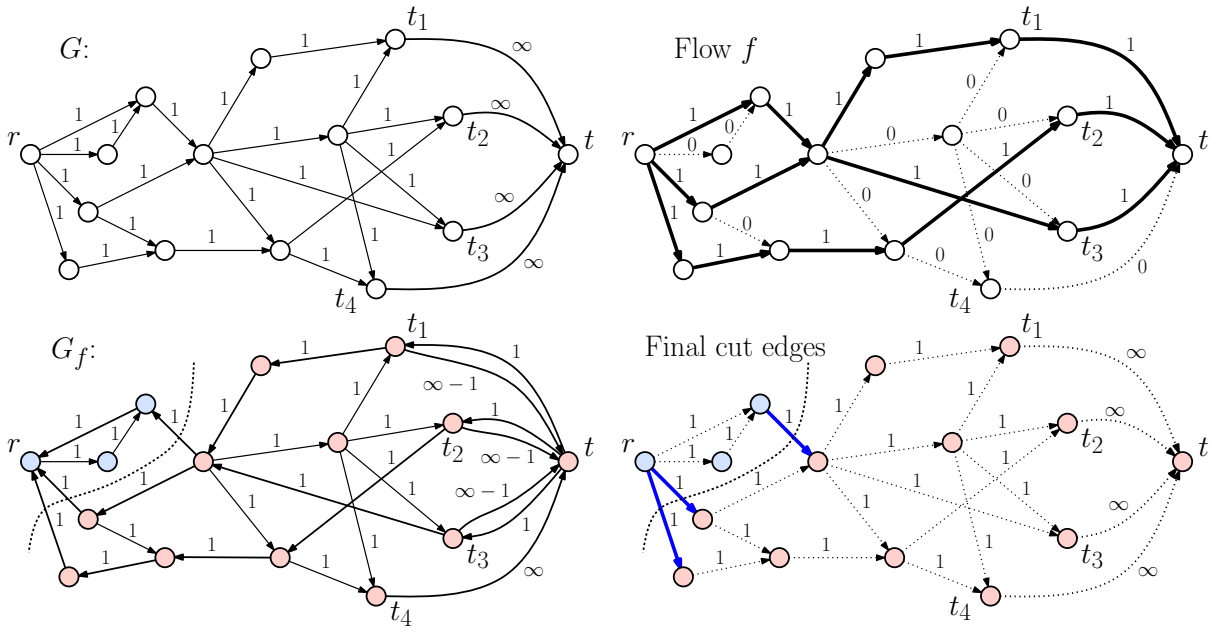

Figure 2: Eliminating edges to separated r from terminals.

Solution 3: Our method for doing this is to generate a network which has many disjoint paths each of which can carry flow, but this network has a single s-t path that passes through all these paths. By choosing this path first, it is impossible to route more flow through the network.

The network is shown in Fig. 3 (for the case $b = 4$). In addition to s and t, the network consists of b pairs of vertices arrayed in a zig-zag pattern. All edges have a capacity of 1. If by bad luck, the algorithm augments one unit of flow along the zig-zag path shown in part (b), then it is impossible to route any more flow in this network without decreasing the flow on some edges. This yields a flow of value 1. On the other hand, if flow is routed through each of the b vertical edges, we achieve a flow of value b. Thus the ratio between the output of the nondecreasing-flow algorithm and the optimal flow is b.

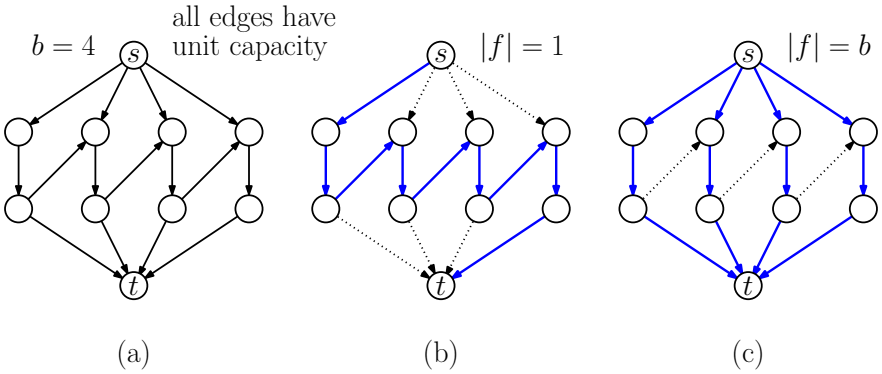

Figure 3: Counterexample for nondecreasing flows.