Advanced Algorithms **Fall Semester**

Lecture 2: Graph Algorithms (Network Flows)

Faculty: K.R. Chowdhary *: Professor of CS*

2.1 Network Flows

One often uses graphs to model *transportation networks* - networks whose edges carry some sort of traffic and whose nodes act as "switches" passing traffic between different edges. Consider an example of high-way system in which the edges are highways and the nodes are interchanges; or a computer network in which the edges are the links that carry packets and the nodes are switches, or a fluid network in which edges are pipes and that carry liquid, and the nodes are junctures where pipes plugged together. The network models of this category have several ingredients: *capacities* on edges, including how much they can carry; *source* nodes in the graph, which generate traffic; and the traffic itself which is transmitted across the edges.

Networks are important for both electronic communication and for transport of goods. The problem of efficiently moving entities (such as bits, people, or products) from one place to other in an underlying network is modeled by *network flow*. The problem plays central role in the fields of operations research and computer science, and much emphasis has been placed on the design of efficient algorithms for solving it. Many of the basic algorithms plays important role in understanding the network flow algorithms.

The *matching*, we studied earlier, is a special case of *flow problem* - match the flow with capacity of edge so that over all flow is maximum. Assignment is a generalization of the matching problem to the wighted case.

Flow network. A flow network $G = (V, E)$ is a directed graph, with two specially marked nodes, namely a single source node $s \in V$ and single sink node $t \in V$. The s, t are called *internal* nodes. In addition, there is a capacity function $c : V \times V \mapsto \mathbb{R}$ that maps edges to positive real numbers.

Before we proceed further, we shall make some assumptions: 1) no edge enters the source s, and no edge leaves t, and 2) there is at least one edge incident on each node. The figure 2.1 shows an example of such network, with capacity c of each shown along with the edges.

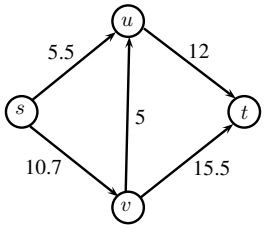

Figure 2.1: A flow network.

Flow. An s-t is a function $f : V \times V \mapsto \mathbb{R}$. The value $f(u, v)$ is amount of flow in edge (u, v).

A flow function is required to satisfy the following constraints:

- *Capacity constraints.* For all $u, v \in V$, $0 \leq f(u,v) \leq c(u,v)$,
- *Skew symmetry constraint.* For an edge $e = (u,v)$, there is $f(u,v) = -f(v,u)$
- *Flow conservation.* For all $u \in V - \{s,t\}$, we require

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v). \tag{2.1}$$

  Whenever, $(u,v) \notin E$, i.e., there can be no flow from u to v, and $f(u,v) = 0$.

The capacity constraint says that the total flow on an edge does not exceed its capacity. The skew symmetry condition says that the flow on an edge is the negative of the flow in the reverse direction. The flow conservation constraint says that the total net flow out of any vertex other than the source and sink is zero. The value of this of net flow flow from s defined as :

$$|f| = \sum_{v \in V} f(s,v) \tag{2.2}$$

In the *maximum-flow problem* the objective is to find a flow function that satisfies the three constraints, and also maximizes the total flow value $|f|$ from source s.

The above formulation of the network flow problem is powerful enough to capture generalization where there are many sources and sinks (single commodity flow), and where both vertices and edges have capacity constraints.

We call the non-negative quantity $f(u,v)$ the flow from vertex u to vertex v, defined for (s,v) as:

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s), \tag{2.3}$$

that is, total flow out of a source minus the flow into the source. Typically, the value $\sum_{v \in V} f(v,s)$ will be zero as there is no flow into the source. In the *maximum-flow problem* we are given the flow network G with source s and sink t, and we wish to find a flow of maximum value.

Example 2.1 *Consider the example in figure 2.2, where figure 2.2(a) shows the capacities $c(u,v)$ of all edges, and figure 2.2(b) shows flows/capacities capacities, i.e., $f(u,v)/c(u,v)$ of all edges.*

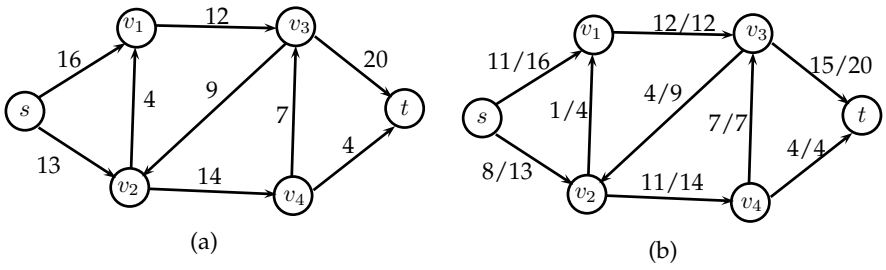

Figure 2.2: (a) A flow network $G = (V,E)$, for transporting trucks, having capacity $c(u,v)$ at each edge, (b) A flow in G with $|f| = 19$. Each edge labeled as $f(u,v)/c(u,v)$.

2.1.1 Max-flow problem

Given a flow network, a natural goal is to arrange the traffic so as to make as efficient use as possible of the available capacity. Thus, the problem is: given a flow network, find a flow of maximum possible value.

As we think about designing an algorithm for this purpose, it is useful to consider how the structure of the flow network places upper bounds on the maximum value of an *s-t* flow. Suppose we divide the nodes of the graph into two sets, S and T, so that $s \in S$ and $t \in T$. Then, any flow that goes from s to t must cross from S into T at some point, and hence will use some of the edge capacity from S to T. This suggests that each such "cut" of the graph puts bound on the maximum possible flow value. The maximum flow algorithm, we are going to discuss, will have maximum flow limited to the minimum capacity of such divisions, called *minimum-cut*. Hence, we would require to find minimum capacity in a flow network for finding the maximum flow.

A flow function $f : (u, v) \mapsto \mathbb{R}$ maps edges to real numbers. For an edge $e = (u, v)$, $f(u, v)$ refers to the flow on edge e, which is also called the net flow from vertex u to vertex v. This notation is extended to sets of vertices as follows: If X and Y are sets of vertices then $f(X, Y)$ can be defined as,

$$\sum_{x \in X} \sum_{y \in Y} f(x, y). \tag{2.4}$$

First, the notation of cuts is defined, and the *max-flow min-cut* theorem is introduced. Then, residual networks, layered networks, and the concept of blocking flows are introduced. Finally, an efficient algorithm for finding a blocking flow is described.

An **s-t cut** of the graph is partitioning of the vertex V into two sets S and $T = V - S$ such that $s \in S$ and $t \in T$. If f is a flow, then the net flow across the cut is defined as $f(S, T)$. The capacity of the cut is similarly defined as,

$$c(S, T) = \sum_{x \in X} \sum_{y \in Y} c(x, y). \tag{2.5}$$

The net flow across a cut may include negative net flow between vertices, but the capacity of the cut includes only negative values, i.e., only the capacity of edges from S to T.

Using the flow conservation principle, it can be shown that the net flow across an *s-t* cut is exactly the flow value $|f|$. By the capacity constraint, the flow across the cut cannot exceed the capacity of the cut. Thus the value of maximum flow is no greater than the capacity of a minimum *s-t* cut. The well-known *max-flow min-cut theorem* proves that the two numbers are actually equal. In other words, if f^* is a maximum flow, then there is some cut $(X, \bar{X})$ such that $|f^*| = c(X, \bar{X})$.

The *residual capacity* of a flow f is defined to be a function on vertex pair (u, v) given by $c'(u, v) = c(u, v) - f(u, v)$. the residual capacity of an edge $(u, v), c'(u, v)$, is the number of additional units of flow that can be pushed from u to v without violating the capacity *constraints*. And edge (u, v) is *saturated* if $c(u, v) = f(u, v)$, that is, if the residual capacity, $c'(u, v)$, is zero.

The *residual graph* $G_R(f)$ for a flow f is the graph with vertex set V, source and sink s and t, respectively, and those edges (u, v) for which $c'(u, v) > 0$.

An augmented path for f is a path $G_R(f)$. The residual capacity of path P, denoted by $c'(P)$, is the minimum value of $c'(u, v)$ over all edges (u, v) in the path P. The flow can be increased by $c'(P)$, by increasing the flow on each edge of P by this amount. Whenever $f(u, v)$ is changed, $f(v, u)$ is also correspondingly changed to maintain the skew symmetry.

Most flow algorithms are based on augmenting the flow in stages. In each stage, a residual graph $G_R(f)$ with respect to the current flow function f is constructed and an augmenting path in $G_R(f)$ is found to increase the value of the flow. Flow is increased along with this path until an edge in this path is *saturated*. The algorithm iteratively keeps increasing the flow until there are augmenting paths in $G_R(f)$, and return the final flow f their output.

Lemma 2.2 *Let f be any flow and f^* a maximum flow in G, and let $G_R(f)$ be the residual flow graph for f. The value of a maximum flow in $G_R(f)$ is $|f^*| - |f|$.*

Proof. Let f' be any flow in $G_R(f)$. Define $f + f'$ to be the flow defined by the flow function $f(u, v) + f'(u, v)$ for each edge (u, v). Observe that $f + f'$ is a feasible flow in G of value $|f| + |f'|$. Since, f^* is the maximum flow possible in G, $|f'| \leq |f^*| - |f|$. Similarly, define $f^* - f$ to be a flow in $G_R(f)$ defined by $f^*(u, v) - f(u, v)$ in each edge (u, v), and this is a feasible flow in $G_R(f)$ of value $|f^*| - |f|$, and it is a maximum flow in $G_R(f)$. □.

Blocking Flow. A flow f is blocking flow if every path in G from s to t contains a saturated edge. Blocking flows are also known as maximal flows. It is important to note that a blocking flow is not necessarily maximum flow. There may be augmented paths that increase the flow on some edge and decrease the flow on other edges (by increasing the flow in the reverse direction).

The maximum flow algorithm proposed by Dinitz starts with zero flow, and iteratively increases the flow by augmenting it with a blocking flow in $G_R(f)$ until t is not reachable from s in $G_R(f)$. At each step the present flow is replaced by the sum of the present flow and blocking flow. The algorithm terminates in $|V| - 1$ iterations since in each iteration the shortest distance from s to t in the residual graph increases. The shortest path from s to t is at most $|V| - 1$, and this gives upper bound of the number of iterations of the algorithm.

2.2 Ford-Fulkerson method

This section section presents a method for solving maximum-flow problem. This method depends on the three important ideas relevant to flow algorithms and problems: residual networks, augmenting paths, and cuts.

The Ford-Fulkerson method iteratively increases the value of the flow. We start with $f(u, v) = 0$ for all $u, v \in V$, giving an initial value of 0. At each iteration, we increase the flow value in G by finding an "augmenting path" in an associated "residual network" G_R. Once we know the edges of an augmenting path in G_R, we can easily identify specific edges in G for which we can change the flow so that we increase the value of the flow. Although each iteration of the F-F method increases the value of the flow, we shall see that the flow on any particular edge of G may increase or decrease; decreasing the flow on some edges may be necessary in order to enable an algorithm to send more flow from the source to sink in other edge. We repeatedly augment the flow until the residual network has no more augmenting path. The max-flow min-cut theoem will show that upon termination, this process yields a maximum flow (see algorithms 1).

We introduce following new concepts, in order to understand the above algorithm.

Residual networks. Given a flow f, the residual network G_f consists of edges with capacities that represent how we can change the flow on edge of G. An edge of the flow network can admit an amount of additional flow equal to the edge's capacity minus the flow on the edge. If that value is positive (> 0), we place that edge into G_R with a "residual capacity" of $c'(u, v) = c(u, v) - f(u, v)$. The only edges of G that are in G_R are those that can admit more flow; those edges (u, v) whose flow equals their capacity have $c'(u, v) = 0$, and they are not in G_R. Note that, in this all paths constructed may not be connected.

Algorithm 1 Ford-Fulkerson algorithm(G, s, t)

```
1: initialize flow, f = 0
2: while there exists an s-t path in the residual network G_R do
3:    Let p be a simple s-t path in G_R
4:    f' = argument(f, p)
5:    update f
6: end while
7: return f
```

The residual network G_R, may also contain edges that are not in G, however. Such edges will be only (v, u) if (u, v) already exists in G. As an algorithm manipulates the flow, with the goal of increasing the total flow, it might need to decrease the flow on a particular edge. That is, you can reduce that flow, to the maximum extent which is equal to the original $f(u, v)$, and not more. In order to represent a possible decrease of a positive flow $f(u, v)$ on an edge in G, we place an edge (v, u) into G_R with residual capacity $c'(v, u) = f(u, v)$, that is, an edge that can admit flow in the opposite direction to (u, v), at most canceling out the flow on (u, v). These reverse edges in the residual network allow an algorithm to send back flow it has already sent along an edge. Sending flow back along an edge is equivalent to decreasing the flow on the edge, which is a necessary operation in many algorithms(see figure 2.3(b). This has been done for all the edges, except those which were running at maximum capacity.

More formally, suppose that we have a flow network $G = (V, E)$ with source s and sink t. Let f be a flow in G, and consider a pair of vertices $u, v \in V$, we define the *residual capacity* $c'(u, v)$ by

$$c'(u,v) = \begin{cases} c(u,v) - f(u,v), & \text{if} (u,v) \in E, \\ f(v,u), & \text{if} (v,u) \in E, \\ 0, & \textit{otherwise.} \end{cases} \tag{2.6}$$

Because our assumption that $(u, v) \in E$ implies $(v, u) \notin E$, exactly one case in equation 2.6 applies to each ordered pair vertices. As an example, if in this equation, $c(u, v) = 16$ and $f(u, v) = 11$, then we can increase $f(u, v)$ by up to $c'(u, v) = 5$ units before we exceed the capacity constraint on edge (u, v). We also wish to allow an algorithm to return up to 11 units of flow from v to u, and hence $c'(v, u) = 11$. This is with $(u, v) = (s, v)$ in figure 2.3(b).

Given the flow network $G = (V, E)$ and a flow f, the *residual network* of G induced by f is $G_R = (V, E_R)$, where,

$$E_R = \{(u, v) \in V \times V : c'(u, v) > 0\}. \tag{2.7}$$

This is as promised above, each edge of the residual network, or residual edge, can admit a flow that is greater than 0.

Figure 2.3(a), (b), (c), respectively show the, flow network G, residual network G_R.

We increase the flow on (u, v) by $f'(u, v)$ but decrease it by $f'(v, u)$ because pushing flow on the reverse edge in the residual network signifies decreasing the flow in the original network as *cancellation*. For example, in a data flow network, if we send 5 packets from u to v, and send send 2 from v to u, we could equivalently just send 3 packets from u to v and none from v to u.

In figure 2.3(b), we can add the flow in forward direction by difference amount, e.g., on edge (s, v_2) we can add 5 units, and at any time we can undo the forward flow by 8 units.

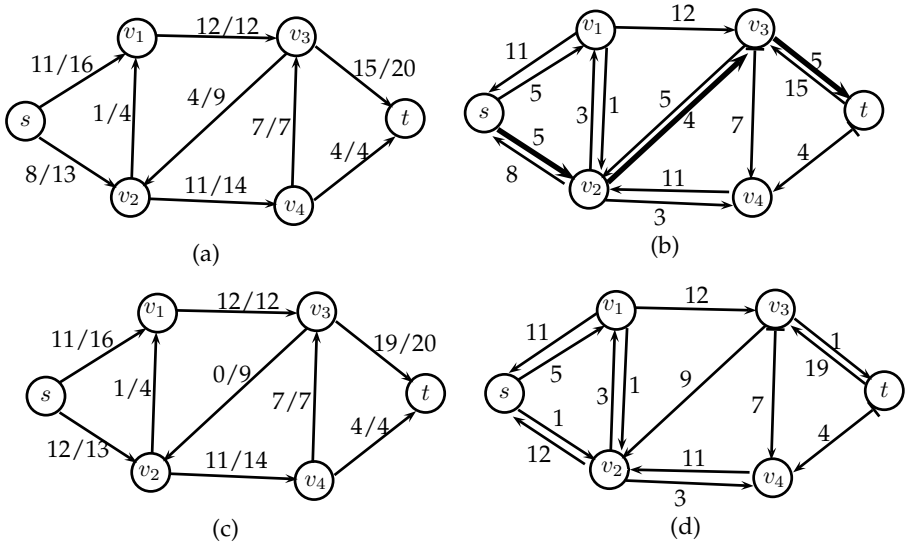

Figure 2.3: (a)Flow network G with flow f, (b) Residual network G_R with augmenting path p (shaded), its residual capacity is $c'(p) = c'(v_2, v_3) = 4$, edges with residual capacity equal to 0, such as (v_1, v_3), are not shown, (c) The flow in G that results from augmenting along path p by its residual capacity 4. Edges carrying no flow, such as (v_3, v_2), are labeled by $c(v_3, v_2)$, (d) The residual network induced by the flow in (c).

After augmenting graph in figure 2.3(a) with the augmenting path $s \to v_2 \to v_3 \to t$, having augmented flow of 4, it cancels existing flow in edge (v_3, v_2), hence flow in edge (v_3, v_2) is zero.

Exercises

1. For the graph shown in figure 2.4 find out the maximum flow. The weights given on edges are of maximum capacities of the edges.

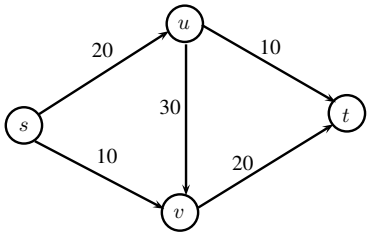

Figure 2.4: Flow graph.

2. List the minimum s-t cuts in the flow network shown in figure 2.5.

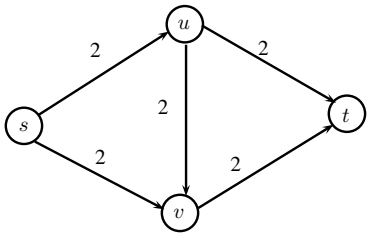

Figure 2.5: Flow graph.

3. What is minimum capacity of a s-t cut in the flow network shown in figure 2.6.

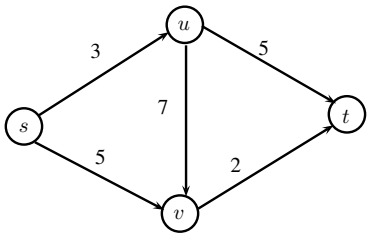

Figure 2.6: Flow graph.

4. For the graph shown in figure 2.7, answer the followings:
   (a) Is this the maximum s-t flow in this graph? Justify.
   (b) Find the minimum s-t cut in this flow network.

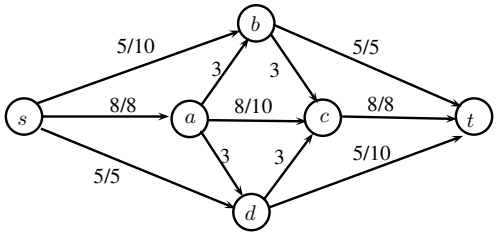

Figure 2.7: Flow graph.

Note: In x/y, x is flow and y is capacity. In case of single value, it is flow.

5. Re-write the Ford-Fulkerson algorithm in your words and explain its working.

References

[1] Thomas H. Corman, et al., "Introduction to Algorithms, 3rd ed." *PHI*, 2009.

[2] Allen B. Tucker, Jr., "The computer Science and Engineering Handbook," *CRC Press*, 1997.