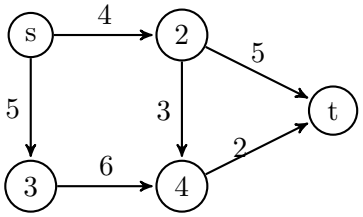
COMS 4995-2: Advanced Algorithms (Spring'20) Feb 4, 2020

Lecture 5: Max-flow Problem, Ford-Fulkerson Algorithm

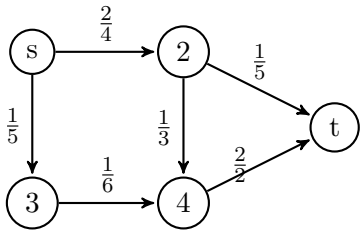
Instructor: *Alex Andoni* Scribes: *Luis Lopez, Jorge Solis*

1 Max-Flow Problem

This lecture we begin our discussion of the Max-flow problem, which is one of an important class of graph problems. We will consider **directed graphs** of the form $G = (V, E, c)$, where each edge $e \in E$ of the graph has a capacity given by the function $c : E \to \mathbb{R}_+$. We will denote the starting vertex by s and the terminal vertex by t, and we will seek to transport some supply, like water or other commodity, from s to t via a network of pipes or other conduits modeled by the edges E of the graph. Furthermore, we seek to maximize the total supply that leaves vertex s and arrives at vertex t, where the capacities represent volume per unit time, etc.



Note that during computation of a solution, we may label edges with ratios, representing the portion of capacity being utilized given by a **flow**, which we will represent by a function $f : E \to \mathbb{R}_+$ that maps an edge to the corresponding volume traveling through it.



This flow is subject to constraints that restrict feasible solutions to the problem:

1. Non-negativity: $\forall\ e \in E$

$$f(e) \geq 0 \tag{1}$$

2. Capacity: $\forall\ e \in E$

$$f(e) \leq c(e) \tag{2}$$

3. Conservation of Flow: $\forall\ v \in V,\ v \neq s, t$

$$\sum_{(v,u)\in E} f(v,u) = \sum_{(u,v)\in E} f(u,v) \tag{3}$$

In the summations above, we identify the function $f(\cdot,\cdot) : V \times V \to \mathbb{R}_+$ defined on ordered pairs of vertices in V with the function $f(\cdot) : E \to \mathbb{R}_+$ defined on the set of edges.

Then, we may define the objective value function or $|f|$:

$$|f| = \sum_{(s,u)\in E} f(s,u) - \sum_{(u,s)\in E} f(u,s) \tag{4}$$

Where, as specified above, the start vertex s (also referred to as the *source*) is not required to obey the conservation of flow constraint, and thus may have positive net flow traveling across the outward facing edges in its neighborhood. This quantity may also be called the **value flow**.

Now, we may formally define the Max-flow problem:

Problem: Find a flow f that maximizes $|f|$.

To recap, a solution to the problem is modeled as a flow vector f that satisfies the constraint of the problem: it satisfies each of the constraints $(1), (2), (3)$ defined above. In other words, each component of the vector $f(e)$, corresponding to an edge in E, satisfies inequalities of the form $0 \leq f(e) \leq c(e)$, and if a vertex is not s nor t, the sum of flows leaving from a vertex must equal the sum of flows entering that vertex. This problem was covered to some extent in COMS 4231: Analysis of Algorithms, so we will discuss solutions encountered during that discussion, but starting Thursday, Feb 6th, we will consider algorithms not seen there.

Our starting point in the development of an algorithm will be to develop the structure of the problem further, beginning with the following result:

Theorem 1. *Decomposition Theorem: For any flow f, we may decompose this flow into a sum of* ***path flows*** *and* ***cycle flows****.*

We define path flows and cycle flows as follows:

Path Flow A path flow is a flow that does not have vertices where flow either joins or bifurcates along the sequence of edges composing the path. Therefore the flow traveling across each edge in the path flow remains constant.

For a path flow, the flow on each edge is equal, so if P is the set of edges composing a path flow f_P, then $\forall\ e \in P,\ f_P(e) = |f_P|$.

Cycle Flow A cycle flow is a flow that sends supply on a path resulting in zero value flow.

A cycle flow is a flow that does not have positive net flow emanating from the source s. One can consider a sequence of edges that begins and ends at the same vertex, resulting in flow that circuits through some convoluted path that forms a cycle or a combination of cycles (essentially wasteful or useless).

What does this theorem mean for us? Essentially, for any flow f, there exist path flows $f_{P_1}, f_{P_2}, \ldots, f_{P_k}$ corresponding to paths $P_1, P_2, \ldots, P_k$ and cycle flows $f_{C_1}, f_{C_2}, \ldots, f_{C_l}$ with accompanying cycles $C_1, C_2, \ldots, C_l$ such that

$$f = \sum_{i=1}^{k} f_{P_i} + \sum_{j=1}^{l} f_{C_j} \tag{5}$$

The point of this discussion is to observe that this is true for all flow vectors, including not just max flows.

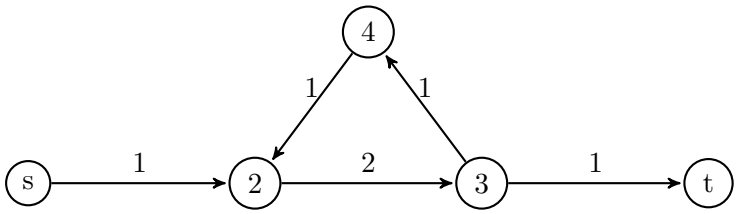
Note: If calling f a vector and then calling f a function is confusing, think about how a linear transformation $T : V_1 \to V_2$ between vector spaces V_1, V_2 is determined by the images $T(e_i) \in V_2$ of the basis elements $e_i \in V_1$ of the vector space (e.g., a 2×2 matrix mapping $\mathbb{R}^2$ to $\mathbb{R}^2$ is determined by its two columns, in fact the images of the unit vectors e_1 and e_2). Thus, one can verify that the set of linear transformations on a vector space forms a vector space: i.e., the set of flows on a graph form a vector space of dimension $|E|$, where the flow $f(e)$ at an edge e is a component of the vector f.

Now, to a proof of the Decomposition Theorem:

Proof:

1. Suppose the value flow $|f| > 0$ for a flow f (so there is an outward facing edge at s with some positive flow).
   This implies that the flow must leave the destination vertex of said edge according to the flow conservation constraints, thus propagating the flow through the graph, until some positive flow arrives at t.
   Let this sequence of edges be denoted $P = \{e_1, e_2, \ldots, e_n\}$ originating at s and terminating at t in the flow f, and define
   $$\delta(P) = \min_{e \in P} f(e)$$
   Next, define
   $$f_P(e) = \delta(P) \cdot \begin{cases} 1, & \text{if } e \in P; \\ 0, & \text{otherwise;} \end{cases}$$
   Then, as long as f_P is a path flow, one can verify that $f' = f - f_P$ by checking non-negativity, capacity, and flow conservation constraints.
   But what if f_P is not a path flow? Then it may be the case after this operation that the new flow

f' does not satisfy flow conservation constraints. Consider the following example:



2. If we have found a cycle $C = \{e_1, e_2, \ldots, e_m\}$ such that $\forall\ e \in C,\ f(e) > 0$, however, we can just carry out the same process:

$$\delta(C) = \min_{e \in C} c(e)$$

$$f_C(e) = \delta(C) \cdot \begin{cases} 1, & \text{if } e \in C; \\ 0, & \text{otherwise;} \end{cases}$$

We then obtain the new flow $f' = f - f_C$ when f_C is a cycle flow.
Thus, if given a flow f such that $|f| = 0$ but $\exists\ e \in E$ such that $f(e) > 0$, we may find the cycle traversed by the flow and decompose it as above.

Now, given some arbitrary flow f, what are the upper bounds on the number of times we may decompose this flow? That is, subtract some path or cycle flow from f until the resulting flow f' is zero everywhere: $\forall\ e \in E,\ f'(e) = 0$.

Observation: Each time we conduct this decomposition process, we set the value of the flow at some edge in the graph to 0. More precisely, the number of edges $e \in E$ with flow $f(e) = 0$ increases by at least one for each iteration. We are considering finite graphs, so after finitely many iterations, the flow at each edge in the graph will be 0, and we will be done. Therefore, we have an upper bound of the number of iterations necessary to decompose a flow: $|E|$.

OK. Now, using this theorem, we may start designing algorithms to solve the problem.

2 Ford-Fulkerson Algorithm

Proto-Alg 1:

1. Find a path from $s \to t$ using edges with $c(e) > 0$.

Claim: By the decomposition theorem, this algorithm will find a path: if $\exists\ f,\ |f| > 0$, then this algorithm will return a path flow f_P with $f_P > 0$

Can we do better? If we modify the edge capacities to reflect the capacity usage of f_P, we can run this algorithm again and possibly find another flow.

Proto-Alg 2:

1. Iteratively find a path P from $s \to t$ using edges with $c_f(e) = c(e) - f(e)$, where f is the current flow (c_f depends on the current flow, and is named the **residual capacity**).

2. Define, as before, $\delta(P) = \min_{e \in P} c_f(e)$, and

$$f_P(e) = \delta(P) \cdot \begin{cases} 1, & \text{if } e \in P; \\ 0, & \text{otherwise;} \end{cases}$$

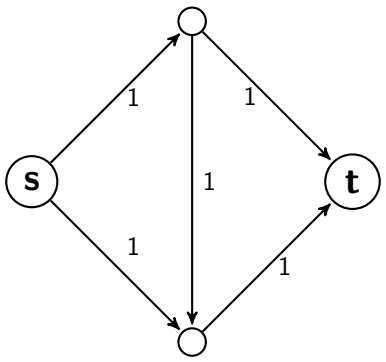
   We then assign new values to the flow at each edge:

$$f_{\text{new}} = f_{\text{old}} + f_P$$

   Both flows satisfy non-negativity, capacity, and flow conservation constraints (verify this). Notice that by the definition of the residual capacities the original capacities are never violated.

3. Terminate when no $s \to t$ paths can be found using edges with $c_f(e) > 0$.

Is it true that this algorithm will always obtain the max flow? Consider the following graph:



Thus, we seek a modification of the algorithm that will reflect an ability to "push back" flow in directions opposite to the current flow, by sending flow across edges with opposite direction, possibly negating the net flow between two vertices.

Residual Graph: G_f is the residual graph for flow f, where:

$$\begin{aligned} \forall\ (u,v) \in E: &\quad c_f(u,v) = c(u,v) - f(u,v) \\ \text{if } f(u,v) > 0: &\quad c_f(v,u) = f(u,v) \end{aligned} \tag{6}$$

After modifying the graph in the above way, a new $s \to t$ path P found that increases flow by "pushing back" flow along an edge such that $\forall\ e \in P,\ c_f(e) > 0$ is is called an *augmenting path.*

We thus arrive at our desired algorithm:

Ford-Fulkerson:

1. Initialize: set $f(e) = 0$ for all $e \in E$, $G_f = G$.

2. While $\exists$ an $s \to t$ path P in G_f such that $\forall\ e \in E,\ c_f(e) > 0$:

   (a) $\delta(P) = \min_{e \in P} c_f(e)$

   (b) $\forall (u, v) \in P$:
   if $f(u, v) > 0$, then $f(v, u) = f(v, u) - \delta(P)$
   else, $f(u, v) = f(u, v) + \delta(P)$

   (c) Recompute G_f.

This algorithm finds positive flows if they exist, like the prototype algorithms, but it is more versatile in that if flow along an edge is increased in one iteration, it may be decreased or negated in a later iteration by an augmenting path, so individual path computed in particular iterations do not irreversibly affect the final result.

Claim 1: The final result is valid flow.

Proof By construction, the flow along augmenting paths computed in each iteration respect our constraints.

Claim 2: FF algorithm terminates with a max flow.

Proof We know that if $|f| = 0$, then the max flow must be zero, otherwise our algorithm would have found a flow.

This does not prove the result, but consider the following to be discussed further on Thursday, Feb 6th:

1. Let the set S be defined as the set of vertices reachable from s:
   $S = \{v \in V : \exists\ s \to v \text{ path } \}$
   If $|f| = 0$, there is no $s \to t$ path: $t \notin S$
   That is, the set of vertices V can be **cut** into two disjoint sets: $S, V - S$.

2. Given a subset of vertices $S \subset V,\ s \in S, t \notin S$, we may define the **capacity** of S:

$$c(S) \triangleq \sum_{u \in S, v \notin S} c(u, v)$$

   In other words, the capacity of S is the sum of capacities of edges leaving S and arriving at $V - S$.

Claim: (Max-flow / Min-cut duality) The value of a max-flow is equal to the minimum capacity of a set $S \subset V,\ s \in S,\ t \notin S$. To be continued.