MFMC and it's variation

Simo Ryu
cloneofsimo@gmail.com

LP@SSHS — September 9, 2019

1 What is it? Why should you care?

It all begins with a simple question. Suppose that you've succesfully learned MaxFlowMinCut theory. Why should you care? Why would it even matter to you? Under assumption that you are a programmer, or computer scientist, this question is almost meaninglessly self evident, as MaxFlowMinCut theory is not only a simple theory, it's also a framework of idea itself. Think about it as learning function. It itself has so little meaning, untill it becomes a form of real life problem. This is called "reduction" of problems, where one problem can be thought of another and vice versa.

Here are some examples from Princeton. [Pr]

- Network connectivity
- Bipartite matching
- Data mining
- Open-pit mining
- Airline scheduling
- Image processing
- etc etc...

Examples speak louder than words.

2 Grasping the Idea of Max Flow

Let's start with the following problem.

Project selection problem

Input: Set P of projects, Project i has revenue of p_i
-Some projects generate money, others require money.
-Some projects have "dependencies", i.e. require other projects beforehand to get them done.

Question: Ok. given all the data of projects, how do would we choose these projects, to maximize revenue?

You begin to realize that this almost seemingly has nothing to do with flows in classic physical way. But we will formalize later on, and they will make sense. First off, what do we mean by "Flow"? In general,

- Static beings, You, people, projects, etc..(Nodes)
- Fluid beings, Data, objects, etc.. (Flows)

Take this idea further, and let's define vertices and directed edges in classic graph theoric sense.

We define set of vertices $V = \{s, t, u_1, u_2, ...\}$ and set of directed edges $E = \{e_1 = (u_1, u_2), e_2 = (u_1, u_3), ...\}$ Then, we will define **Capacity function** and **Flow function** each $c : E \to \mathbb{R}$ $f : E \to \mathbb{R}$ where basically we are **assigning values to these edges** that implies **how much this can hold, and how much flow it has**.

Notice that in set V, we have element s and t, they each represent **Source of the flow** and **Sink of the flow**. See where we are headed?

Following directed graph has 5 vertices, and 8 directed edges.

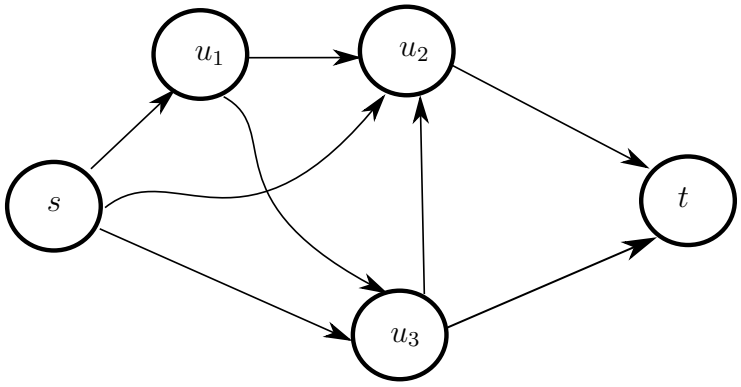

Figure 1: Directed Graph with Source(s) and Sink(t)

Moreover, let's assign capacity of the edges. Ususally capacities are "given", in the sense that one only has to concern about flow of the network.

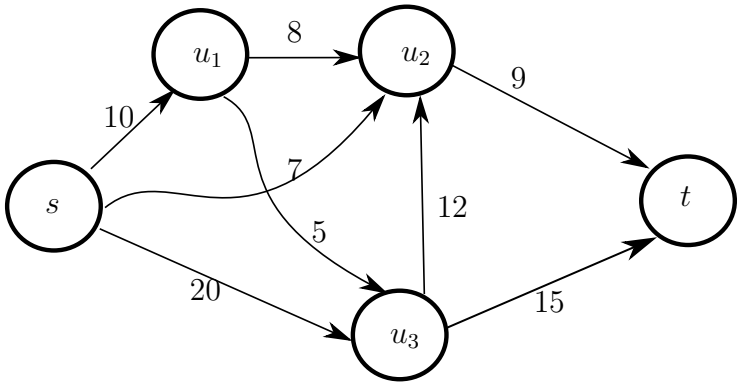

Figure 2: Graph with given capacity values

We would expect two things from flow function.

- Conservation of flow (that is, sum of all flow into a vertex should be equal to sum of all flow out of the vertex.)
- Bounded by capacity ($c(e) \leq f(e)$ for all $e \in E$ (except, obviously, for s and t.))
- Skew symmetry ($f(x, y) = -f(y, x)$ for vertices x, y)

Idealy we want to generalize notion of $f(e), c(e)$ into much general form, so we define the following.

$$f(X, Y) = \sum_{v \in X, u \in Y} f(x, y), X, Y \subset V$$

vice versa for capacity. In such way, we can define the flow of the network as $|f| = f(s, V)$. From condition of conservation of flows, we expect $f(s, V) = f(V, t)$ and indeed, this is true. (proof as exersise) Alright, now the main problem is this.

Max Flow Problem
Input: Directed Graph G, with capacity function c.
Question: How would we set the function f, so that the flow of the graph $|f|$ is `maximized?`

This is the killer problem. Before we cover how we will make the algorithmic choices to maximize the flow, we have to trigger another problem.

3 Minimum Cut

3.1 ST cut

What is a "ST cut"? Well, ST cut is simply partition of set V into two subset S, T, that $s \in S$ and $t \in T$ is satisfied. In other words, you are making "source" side of a network, and "sink" side of a network.

Now, given network G and ST cut S, T, **capacity of a ST cut** is defined as follows.

$$c(S,T) = \text{sum of all capacities of edges leaving S to T}$$

In the following example, we have $S = \{s, u_1, u_2\}$ and $T = \{t, u_3\}$

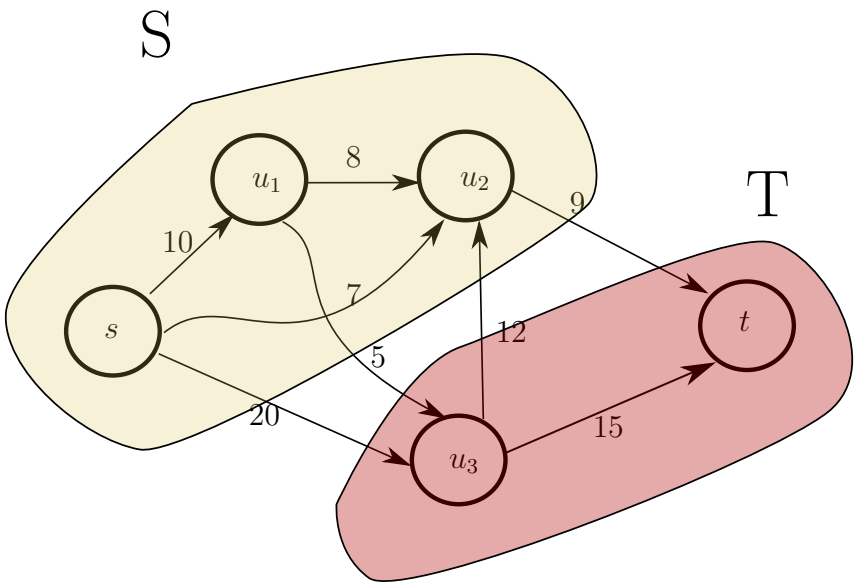

Figure 3: Network with ST cut

Using the above definition, we can simply calculate $c(S,T)$ by adding all the capacities from S to T.

$$c(S,T) = c(s,u_3) + c(u_1,u_3) + c(u_2,t) = 34$$

Obviouly we see that x element of $c(x,y)$ is always from source S, and y element is always from T. Here we see that flow is not mentioned. Now we are ready for the next big problem.

Min Cut Problem

Input: Directed Graph G, with capacity function c.

Question: How would we make a ST cut of G, that capcity of the ST cut is `minimized?`

3.2 Observations

First thing first, we realize that, once we have a cut, the capacity of the cut is self evidently greater or equal to the flow of the network, because... well, it's not so self evident, so we'll prove by next two step.

$$|f| = f(S, T)$$

$$f(S, T) \leq c(S, T)$$

and obviously, as exercises to the readers.

Now as we see, hence all ST cut's capacity is greater or equal to all possible flows, minimum ST cut is also greater or equal to maximum flow, thus we have the following inequality.

$$\max |f| \leq \min c(S, T)$$

Surprisingly, above equality holds for some f and some ST cut, regardless of the given network G. This is called **Max Flow Min Cut Theorem**.

4 Residual Network and Ford-Fulkerson algorithm

To prove the Max Flow Min Cut Theorem, we need to know bout **Residual Network**.

4.1 Residual Network

Let f be a flow on G. The residual network G_f is a network with same vertices, but different edges: each defined to be

$$c_f(u, v) = c(u, v) - f(u, v)$$

As always, if the capacity is not positive, we do not express it in the graph. Instead, by above definition, both $c_f(u, v), c_f(v, u)$ have a possibility to become positive. In that case, we mark them both in the Network. Therefore in general, residual network do not meet the skew symmetry condition.

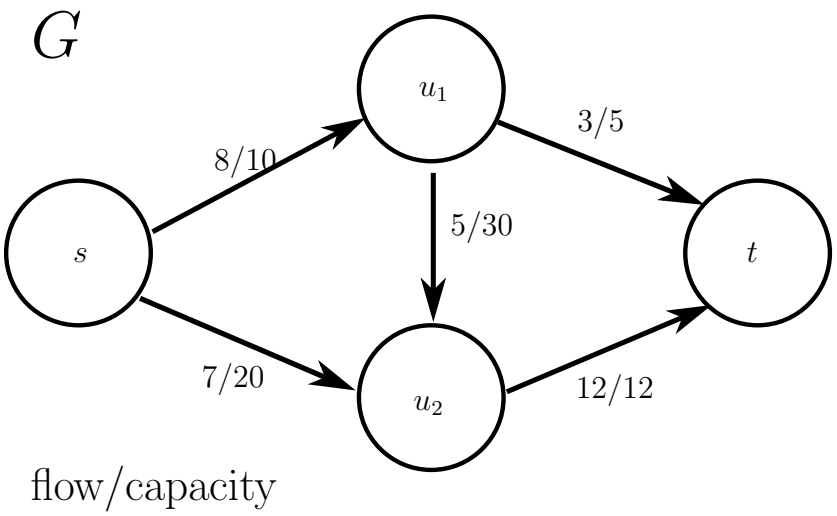

Figure 4: Network of $G = (\{s, t, u_1, u_2\}, E)$ and f

As an example, in the above case, we have Graph G and flow f ready and setup. Each edge has x/y where x is current flow, and y is capacity of the edge.

Now using the above network, we can define the Residual network of G and f.

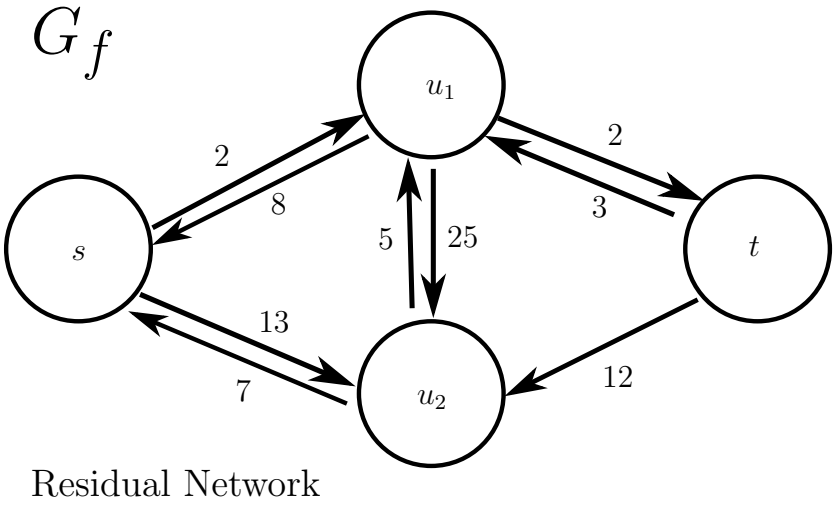

Figure 5: Residual Network of G and f

You can see that here, we only have capacities in a sense that, we never talk about any kindof flows within the Residual network `Yet`. Also you can check that the name "residual" is almost self evident. If there are values on edges within this residual graph, the origianl graph is capable of "accepting" more flows within the flow f. Taking this argument further, we can deduce that any kindof path connecting s to t within the residual network will imply that that path can be smoothly added to the original graph with current flow. Thus, taking the flow closer to MaxFlow state. One can then vaguely suspect that once no more paths can be added, the flow is indeed Maxflow. Indeed, this is the case.

4.2 Ford-Fulkerson, and proof of MaxFlowMinCut

Taking above argument to more formal form, we define "augmenting path of G and f" as "path that connects s to t within G_f". Then it is left to prove the following 3 statements as equivalences.

1. $|f| = c(S,T)$ for some ST cut

2. f is maxflow.

3. f admits no augmenting path.

We've already kinda proved 1 to 2 and 2 to 3, so we are left with 3 to 1. The proof goes something like this.

1. First think about all the residual network of f, and since they have no path from s to t, from s, where would following any path lead?

2. Now that you've thought about connected components of G_f, notice that part that has s can be viewed as S, therefore making ST Cut.

3. There is no path from S to T. That means capacity is all full of flows. Thus $|f| = c(S, T)$!

Now, finding augmenting path in Bruth force manner obviously takes infinite time. finding path in BFS? sure. O(E) time. But now just think about the following "famous" graph.

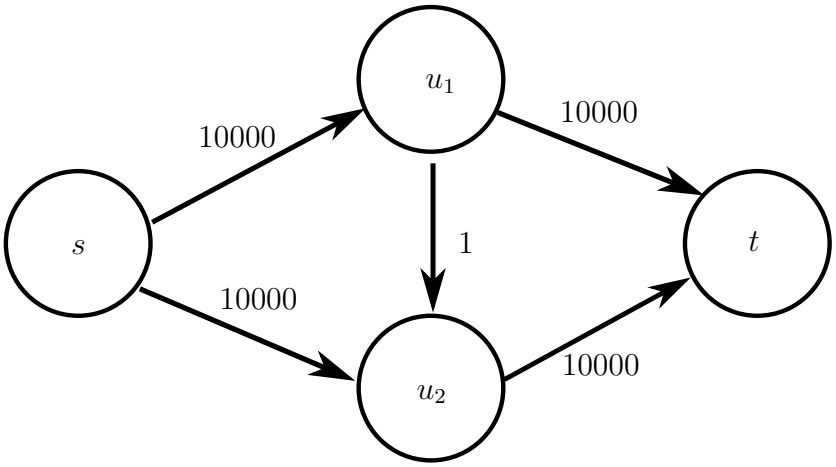

Figure 6: Not-so-good graph for just BFS Ford Fulkerson

Just how many iterations will there be? Well, if we just asuume the worst case senario, we are talking about 40000 iterations, and we don't want that. In this case, every path finding step was exteremly simple, but we are having ridiculously many iterations due to the wild "gap" between minimum capacity and maximum capacity of edges. In this case the algorithm could possiblily take $O(Ef)$ time. We really don't want that. Next time we'll talk about Edmond-Karp and finally, the problem of Project Selection.