Advanced Network Flow

1 Application of Maximum Flow: Bipartite Matching

1.1 Problem Definition

A **bipartite graph** is a graph $G = (V, E)$ such that it is possible to partition V into two non-empty subset L and R such that every edge has exactly one endpoint in L and one endpoint in R. In other word, no edge connect two vertex from the same set. A bipartite graph is also known as **bicolourable graph** since we can colour the vertices in the graph with black and white such that no edge connect two vertices with the same colour.

A **matching** M of a graph is a subset of the edges E such that no two edges in M shares a common endpoint. The **maximum matching** is the matching with the maximum size (ie $|M|$ is maximized). In general, maximum graph matching in NP-hard. However, a polynomial time algorithm exists in the case of a bipartite graph. In fact, one can use the *Hopcroft-Karp algorithm* to solve the problem in $O(\sqrt{V}E)$ time. Here, we will present a $O(VE)$ solution using maximum flow.

1.2 Algorithm

Suppose we are given a bipartite graph $G = (V, E)$. Since G is bipartite, we will further partition $V = L \cup R$ as defined above. Now let us set up the network graph as follows:

- On top of V, add two more vertex s and t (the source and sink respectively).
- For each edge $(u, v) \in E$, add a pipe with capacity 1 from u to v (assume WLOG that $u \in L$ and $v \in R$).
- For each $u \in L$, add a pipe from s to u with capacity 1.
- For each $u \in R$, add a pipe from u to t with capacity 1.

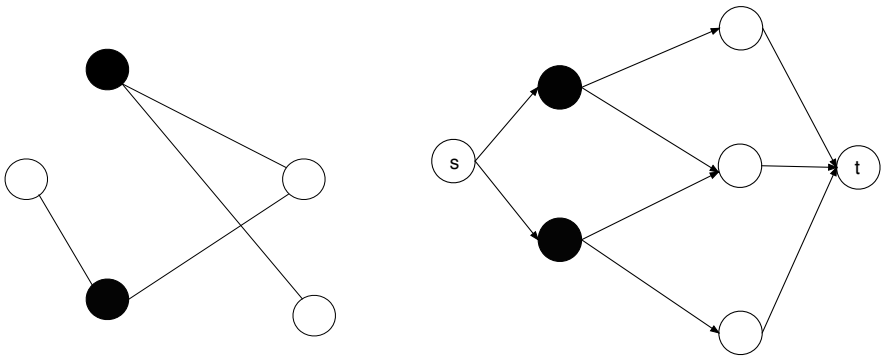

Figure 1: Setting up the network flow graph for a bipartite matching. All pipes have capacity 1.

It is not hard to see why the maximum flow will yield the maximum bipartite matching. To recover the actual matching, we just need to find the pipe that has a flow. Note that in this case, the maximum number of times we will find an augmenting path in the residual graph is V. Thus, this algorithm takes $O(VE)$ time.

1.3 Sample Problem

Here is a typical bipartite matching problem: there are m job applicants and n job openings. Each job opening can accept one applicant and, for obvious reason, each job applicant can only take on one job opening. Furthermore, each job applicants is interested in only certain job openings and each job opening is suitable only for some job applicants. This means that not all job applicant - job opening pair are "compatible". Your job is to allow as many applicants to get a job while ensuring all the pairings are compatible (ie. the job applicant is both suitable and interested in the job opening).

To model this problem as a bipartite matching problem, let $L = \{\text{job applicants}\}$ and $R = \{\text{job opening}\}$. There is an edge between each pair of "compatible" job applicant - job opening pair. Then the answer we want is the maximum bipartite matching.

2 The Max Flow Min Cut Theorem

2.1 Definition

Given a network flow graph $G = (V, E)$, a **cut** is a partition of vertices V into two disjoint subset S and T such that $s \in S$ and $t \in T$ (where s is the source and t is the sink). The **capacity** of a (S, T) cut is defined as

$$c(S,T) \;=\; \sum_{u \in S, v \in T, (u,v) \in E} capacity(u,v)$$

The **minimum cut** (or min cut for short) is the cut with minimal capacity.

2.2 The Main Result

Suppose we have found the maximum flow on the network flow graph $G = (V, E)$ and denote the value by f. Consider an arbitrary (S, T) cut. First, note that since the source is in S and the sink is in T and that flow is conserved in every junction (ie. the amount of flow into a junction is equal to the amount of flow away), so

$$f \;=\; \sum_{u \in S, v \in T, (u,v) \in E} \text{flow along } (u,v) \;-\; \sum_{u \in T, v \in S, (u,v) \in E} \text{flow along } (u,v)$$

It is now a simple exercise to show the following lemma:

Lemma 2.1. *The capacity of any arbitrary (S, T) cut is greater or equal to the maximum flow*

Now, let us define a (S, T) cut ourself by setting $S = \{u \mid \text{a path exists from } s \text{ to } u \text{ in the residual graph}\}$ and $T = V - S$. First, let us verify that this is indeed a cut. It is clear that $s \in S$, so all we need to show is that $t \in T$. However, since we already have the maximum flow, there must not be an augmenting path from s to t (otherwise, we can push flow along the path and further increase the flow - contradicting the maximality of the current flow). Thus $t \notin S$ and $t \in T$. By the definition, there is no edge from a vertex in S to a vertex in T in the residual graph.

Now consider the edges in $(u, v) \in E$ which has one endpoint in S and one in T. The first case is that $u \in S, v \in T$. However, S and T are disconnected in the residual graph, it follows that the flow along (u, v) must be equal to the capacity of (u, v). The second case is that $u \in T, v \in S$. If there

is some positive flow along (u, v) then the edge (v, u) with some positive capacity must exist in the residual graph. This contradicts the fact that S and T are disconnected in the residual graph.

Thus, we have concluded that $\forall(u, v) \in E, u \in S, v \in T$, the flow along (u, v) is equal to $capacity(u, v)$. Similarily, $\forall(u, v) \in E, u \in T, v \in S$ the flow along (u, v) is 0. Substituting these values into the equation for f above, we get:

Lemma 2.2. *The (S, T) cut we constructed has capacity equal to the maximum flow f.*

Combining the two lemmas together, we get the **Max Flow Min Cut Theorem**:

Theorem 2.1. *The minimum cut of a network graph is equal to its maximum flow.*

This theorem tell us that solving for minimum cut is equivalent to solving for maximum flow. Hence, we can use the ford fulkerson method to solve the min cut problem. The sample problem give an example of this.

3 Min Cost Max Flow

3.1 Problem Definition

Let us now add an extra attribute to each pipe - the cost of flowing one unit of water. For a particular flow, the **cost** of the flow is defined as $\sum_{(u,v)\in E} cost(u, v) * \text{flow along}(u, v)$. The problem is to find the **maximum flow of minimum cost**. It is important to note that we maximize flow first before minimizing the cost.

One common application of Min Cost Max Flow is for minimum cost bipartite matching. Let us continue with the job applicant and job opening example above. Suppose you are the hiring manager and that on top of applicant's interest for each opening, they also have a salary preference for each interested opening. Your goal is to fill up as many opening as possible, while paying out as little salary as possible. We can use a similar method to solve this problem. We will set up the network graph as above. In addition, each edge representing a "compatible" pair will have the cost equal to the salary preference. All other edges will have cost 0. We can then find the answer using min cost max flow!

3.2 A Greedy Solution

So how do we solve the Min Cost Max Flow problem? One would hope that there is a simple extension to Ford Fulkerson's method. Fortunately, this is indeed the case. Instead of using BFS or DFS to find an augmenting path, we greedily look for the path from s to t with the minimum cost to push flow - this is essentially a shortest path problem! However, note that we cannot use Dijkstra's algorithm to find the augmenting path. This is because once we push flow along the edge (u, v) with positive cost, then we add the edge (v, u) in the residual graph to allow us to "push flow back". However, when we push some flow back, we must also subtract the cost of those flow. Thus, the cost of (v, u) will be negative! Therefore, the residual graph will contain negative weight edges and we need to use Bellman Ford to find the augmenting path. This can be implemented with relatively little change to the code presented last class - simply change the implementation of `find_augmenting_path()` to use Bellman Ford. Since Bellman-Ford runs in $O(VE)$ time, this algorithm takes $O(VE * f)$ time, where f is the value of the maximum flow (though in practice, it's usually faster). Other faster algorithms exists to solve the problem, but will not be covered.