5 Maximum Flows

In this chapter, we study network flows and cuts. A *flow network* is a directed graph $G = (V, E)$ (on n vertices and m edges, as usual), where each edge e has *capacity* $c_e \geq 1$, along with a designated *source* $s \in V$ and *sink* $t \in V$. For simplicity, we assume that every capacity is an integer, no edges enter s, and no edges leave t.

Any subset of vertices S is known as a *cut*, and the set of edges leaving (or *crossing*) the cut is denoted by $\delta^{\mathsf{out}}(S) = \{(u, v) \in E : u \in S, v \notin S\}$. Similarly, let $\delta^{\mathsf{in}}(S)$ denote the set of edges entering S. In this section, we will focus on a subset of cuts known as *s-t cuts*; these are cuts S that contain the source vertex s.

A *flow* is a function $f : E \to \mathbb{R}^+$; let $f^{\mathsf{out}}(S)$ denote the sum of flow on edges in $\delta^{\mathsf{out}}(S)$, and define $f^{\mathsf{in}}(S)$ analogously. When working with a flow, we often implicitly assume that it is *feasible*, that is, it satisfies the following conditions:

1. Capacity constraints: For each edge $e \in E$, we have $0 \leq f(e) \leq c_e$.
2. Conservation: For each vertex $u \in V \setminus \{s, t\}$, we have $f^{\mathsf{in}}(u) = f^{\mathsf{out}}(u)$.

The *value* of a flow f is defined as $|f| = f^{\mathsf{out}}(s)$, and the maximum flow problem is to find a feasible flow with maximum value. The minimum s-t cut problem is to find an s-t cut $S \subset V$ such that the total capacity $c(S)$ of edges in $\delta^{\mathsf{out}}(S)$ is minimized. In Sec. 5.1, we show how the Ford-Fulkerson algorithm solves both problems, and in the subsequent sections, we illustrate applications of these problems.

But before we begin, we state an important connection between flows and cuts. We leave the proof of the following lemma as exercises.

Lemma 5.1. *For any flow f and s-t cut S, we have $|f| = f^{\mathsf{out}}(S) - f^{\mathsf{in}}(S) \leq c(S)$.*

The lemma formalizes two intuitive ideas about flows and s-t cuts: flow value is conserved across the cut, and this value cannot exceed the capacity of the cut.

5.1 The Ford-Fulkerson algorithm

Problem statement. Let G be a flow network with source s and sink t; our goal is to find a maximum flow from s to t. As we will see, we can also consider the problem of finding an s-t cut of minimum (outgoing) capacity.

Remark. A natural approach is to use BFS (see Sec. 1.1) to find a path from s to t, increase $f(e)$ for every edge e on this path, remove edges that have zero remaining capacity, and repeat until the graph contains no s-t path. This algorithm has the right idea but won't always work; we leave this an exercise.

Algorithm. Start with $f(e) = 0$ for every edge e. For any flow f, we define the *residual network* G^f as follows: G^f and G have the same set of vertices. For each edge $e = (u, v)$ in G, if $f(e) < c_e$, we add e to G^f and set its residual capacity to $c^f_e = c_e - f(e)$; these are *forward* edges. Furthermore, if $f(e) > 0$, we add another edge (v, u) to G^f with capacity $c^f_e = f(e)$; these are *backward* edges. (So G^f has at most twice as many edges as G.)

The algorithm repeatedly finds an s-t path P in G^f (using, e.g., BFS), and uses P to increase the current flow. Such a path is known as an *augmenting path*. The amount by which we modify the current flow value is the minimum residual capacity in P, which is always positive.

Algorithm 12 Ford-Fulkerson

```
1: Initialize f(e) = 0 for all e ∈ E.
2: while there exists an augmenting path P in G^f do
3:     Let Δ_P = min_{e∈P} c^f_e.                ▷ Δ is the "bottleneck" capacity in P
4:     for each edge e ∈ P do
5:         if e is a forward edge then
6:             Increase f(e) in G by Δ_P.
7:         else
8:             Decrease f(e') in G by Δ_P, where e' = (v, u).
9:     Update the residual graph G^f.
10: return f
```

Theorem 5.2. *Algorithm 12 computes a maximum flow in $O(mv)$ time, where m is the number of edges in G and v is the value of the maximum flow.*

Proof. We start by proving that the algorithm terminates in $O(mv)$ time. We do this by proving that $|f|$ increases by $\Delta_P \geq 1$ in every iteration; this suffices because $|f|$ must always be at most v, and each iteration takes $O(m)$ time.[1] Observe that for any augmenting path P, the first edge e of P must leave s, and no edges of G enter s, so e must be a forward edge. This means $f(e)$ increases by Δ_P, so $|f|$ increases by $\Delta_P \geq 1$.

Now we prove that the final flow f is a maximum flow.[2] Let S^* denote the set of vertices reachable from s at the end of the Ford-Fulkerson algorithm; note that $t \notin S^*$ because G^f has no augmenting path. Also, for every edge $e = (u, v) \in \delta^{\mathsf{out}}(S^*)$, we have $f(e) = c_e$. For if not, then e would be a forward edge in G^f, so v would be reachable from s, contradicting the fact that $v \notin S^*$. We can similarly show that for every edge $e' \in \delta^{\mathsf{in}}(S^*)$, we have $f(e') = 0$. Combining this with Lemma 5.1, we get

$$|f| = f^{\mathsf{out}}(S^*) - f^{\mathsf{in}}(S^*) = \sum_{e \in \delta^{\mathsf{out}}(S^*)} c_e - 0 = c(S^*).$$

Recall that Lemma 5.1 also states that the value of any flow is at most the capacity of any cut. Thus, f must have maximum value, because any larger flow f' would have $|f'| > |f| = c(S^*)$, contradicting the lemma. □

[1] From the definition of G^f, we must have $\Delta_P > 0$. Also, since all residual capacities are initially the original (integer) capacities, the first Δ_P is an integer. So after the first iteration, all residual capacities are still integers. Inductively, this implies Δ_P is always an integer; hence $\Delta_P \geq 1$.

[2] Technically, we should first show that f is a feasible flow throughout the algorithm. The proof is fairly straightforward but somewhat tedious, so we leave it as an exercise.

As a bonus, we can also solve the minimum s-t cut problem in $O(mv)$ time. Note that this doesn't necessarily return the minimum cut of the graph, because that cut might not include s. We leave this problem as an exercise.

Theorem 5.3. *We can compute the minimum cut of a flow network G with maximum flow value v in $O(mv)$ time.*

Proof. We first compute a maximum flow f in $O(mv)$ time using the Ford-Fulkerson algorithm. After that, we are left with the residual network G^f that contains no s-t paths. In $O(m)$ time, we find the cut S^* containing vertices reachable from s in G^f. This cut must be a minimum cut, because if there were a cut S' with smaller capacity, then we'd have $c(S') < c(S^*) = |f|$, which violates Lemma 5.1. □

Finally, we can use the proof of Theorem 5.2 to prove the *Max-Flow Min-Cut Theorem* stated below; we leave the details as an exercise.

Theorem 5.4. *In any flow network, the value of a maximum s-t flow is equal to the capacity of a minimum s-t cut.*

5.2 Bipartite Matching

Problem statement. Let $G = (V, E)$ be a bipartite graph where $V = L \cup R$. Recall that a *matching* is a subset of edges such that no two edges share an endpoint. Our goal is to find a matching in G containing the largest number of edges.

Algorithm. Construct a flow network $G' = (V', E')$ as follows: add vertices s, t to V to form V'.[3] The edge set E' contains every edge in G directed from L to R, as well as (s, u) for every $u \in L$ and (v, t) for every $v \in R$. All edges in G' have capacity 1. Use the Ford-Fulkerson algorithm to find a maximum s-t flow f in G', and returns the edges (u, v) (where $u, v \in V$) that satisfy $f(u, v) = 1$.

Theorem 5.5. *The algorithm above returns a maximum matching in G in $O(n^2)$ time.*

Proof. Let M denote the edges returned by the algorithm; we first show that M is a matching. Consider a vertex $u \in L$. There exists at most one $v \in R$ such that $f(u, v) = 1$ because the amount of flow entering u is at most 1, so the amount of flow leaving u is also at most 1. We can similarly show that every vertex in R is incident to at most one edge in M.

Next, we show $|f| = |M|$ by applying Lemma 5.1 to the cut $\{s\} \cup L$. Each edge of M contributes exactly 1 unit of flow leaving this cut, and no edges enter this cut. Thus, $|f|$ is just the amount of flow leaving the cut, which is equal to $|M|$.

Finally, we show that M is a maximum matching. For any matching M', we can construct a flow f' as follows: for every $(u, v) \in M'$, we set $f'(s, u) = f'(u, v) = f'(v, t) = 1$ (and 0 everywhere else). Since f' consists of $|M'|$ edge-disjoint paths, it has value $|M'|$. This value is at most $|f| = |M|$ because f is a maximum flow, so $|M'| \leq |M|$.

[3] We can visualize G' as follows: L and R are two columns of vertices, where L is left of R. Vertex s is left of L, and t is right of R. All edges go from left to right, and no edge "skips" over any column.

The running time of the algorithm is bounded by the running time of computing a maximum flow. If $|L| = n/2$, then the maximum flow value is at most $n/2$, so running the Ford-Fulkerson algorithm would take $O(n^2)$ time. □

5.3 Vertex Cover in Bipartite Graphs

Problem statement. Let $G = (V, E)$ be a bipartite graph where $V = L \cup R$. Recall that $S \subseteq V$ is a *vertex cover* if every edge of G is incident to at least one vertex in S. Our goal is to find a vertex cover of G containing the fewest number of vertices.

Algorithm. We construct a flow network $G' = (V', E')$ very similar to the one from Sec. 5.2. The vertex set V' contains V and two additional vertices s, t. The edge set E' contains every edge in G directed from L to R, (s, u) for every $u \in L$, and (v, t) for every $v \in R$. The original edges have capacity ∞ and the new edges have capacity 1. Use the Ford-Fulkerson algorithm to find a minimum s-t cut S in G', and return $(L \setminus S) \cup (R \cap S)$.

Theorem 5.6. *The algorithm above returns a minimum vertex cover of G in $O(n^2)$ time.*

Proof. Let C denote the output of the algorithm; we first show that C is a vertex cover. Any edge $(u, v) \in E$ not covered by C must satisfy $u \in L \cap S$ and $v \in R \setminus S$. Thus, (u, v) crosses the cut, so $c(S) = \infty$, which contradicts the fact that S is a minimum s-t cut.

Now we show $|C| = c(S)$. There are two types of edges crossing S: those of the form (s, u) where $u \in L \setminus S$, and those of the form (v, t) where $v \in R \cap S$. Each of these edges contributes capacity 1 to $c(S)$, and the number of these edges is $|L \setminus S| + |R \cap S| = |C|$.

Finally, we show that C is a minimum vertex cover. For any vertex cover C', consider the s-t cut $S' = \{s\} \cup (L \setminus C') \cup (R \cap C')$. By the same reasoning as above, we have $c(S') = |C'|$. Since S is a minimum s-t cut, we have $|C| = c(S) \leq c(S') = |C'|$, as desired.

The running time is again (see Theorem 5.5) bounded by the running time of computing a maximum flow. Since the maximum flow value in G' is at most $|L|$, the running time of Ford-Fulkerson is $O(n^2)$. □

5.4 Exercises

1. Let f be a feasible flow from s to t. Prove that $|f| = f^{\mathsf{in}}(t)$.

2. Prove that f remains a feasible flow during any run of the Ford-Fulkerson algorithm.

3. Let S be a cut in a flow network G, and let f be an s-t flow in G. Prove that the value of f is equal to $f^{\mathsf{out}}(S) - f^{\mathsf{in}}(S)$.[4]

4. Use the previous exercise to prove $|f| \leq c(S)$ for every flow f and cut S.

5. Show why the "natural approach" described at the beginning of Sec. 5.1 doesn't always return a maximum flow.

[4]This statement is intuitively true, but you should still write a proof. In particular, at least one $\sum_{u \in S}$ should probably appear somewhere in the proof.

6. Give an $O(mnv)$-time algorithm that finds the *global* minimum cut (i.e. not necessarily an s-t cut) of a flow network.

7. Prove the Max-Flow Min-Cut Theorem (Theorem 5.4).

8. Consider the maximum flow problem, but suppose there are $k \geq 1$ sources $\{s_1, \ldots, s_k\}$ and $\ell \geq 1$ sinks $\{t_1, \ldots, t_\ell\}$. No edges enter any source or leave any sink. A feasible flow is one that satisfies capacity constraints, as well as conservation at all vertices other than sources and sinks. The value of a flow is now defined as the total amount of flow leaving the sources, and we want to find a flow of maximum value. Show how this problem can be solved using the Ford-Fulkerson algorithm.

9. Consider the minimum s-t cut problem, but now each *vertex* (instead of edge) has a capacity; our goal is to remove a minimum weight subset of vertices such that in the resulting graph, there is no path from s to t. Show how this problem can be reduced to the standard minimum s-t cut problem.

10. Consider the same problem as Sec. 5.3, but now each vertex u has weight $w(u) \geq 0$, and we want to find the minimum weight vertex cover. (Sec. 5.3 was the special case where all weights are 1.) Show how to modify the algorithm in that section to solve this problem.

11. Prove that in any bipartite graph, the size of the maximum matching is equal to the size of the minimum vertex cover. (This is known as the König-Egeváry Theorem.)

12. Let $G = (V, E)$ be a bipartite graph where $V = L \cup R$ and $|L| \leq |R|$. For any subset $S \subseteq V$, let $N(S)$ denote the set of neighbors of S. Prove the following: G has a matching of size $|L|$ if and only if $|S| \leq |N(S)|$ for every $S \subseteq L$. (This is known as Hall's Theorem.)

13. Let G be a flow network in which all edges have capacity 1. Prove that there are k edge-disjoint s-t paths if and only if the maximum s-t flow has value at least k.

14. Consider the same setting as the previous exercise. Prove that the maximum number of edge-disjoint s-t paths is equal to the minimum number of edges we'd need to remove to separate s from t. (This is known as Menger's theorem.)