

## Main Steps

Solutions using network flows have a number of common points.

**Step 1: Construct a flow network.** Given the input to the problem you are trying to solve, describe how you will construct your flow network. That is, what are the vertices, what are the edge capacities, what is the source and sink?

**Step 2: Define the correspondence between the flow solution and your solution.** This step is easy to forget, but is very important. You should clearly state how a solution to the constructed flow problem corresponds to a solution to your original problem; that is, show one of the following:

- given a flow in your network, how to construct an assignment in the original problem,
- given a cut in your network, how to construct a partition in your original problem, or
- given a flow in your network, how to construct a set of paths in your original problem.

**Step 3: Prove that the solution you find is correct.** You need to show that any solution found by your algorithm is in fact a feasible (or, optimal) solution. That is, use the properties of your constructed flow/cut to show one of the following:

- the assignment/matching you constructed in step 2 has the properties you want it to,
- the cut you constructed in step 2 has the value and properties you want it to, or
- the paths you constructed in step 2 have the properties you want them to.

**Step 4: Prove that you find all solutions.** Step 3 shows that any solution you find is correct; but this is not enough to show correctness of your whole algorithm. (Namely, an algorithm that never returns anything satisfies step 3, but is probably not going to solve your original problem.) Therefore, it is crucial that you also show that your algorithm finds all solutions to your original problem. Usually this is shown by considering an arbitrary solution to your original problem, and showing how you would construct a flow or cut in your constructed network. In this way, you guarantee that when there is a solution to your original problem there is also a flow in your constructed flow network, and so you don't accidentally overlook any solutions. Note that you have to use the properties of a solution to your original problem to argue that the flow you construct is actually a flow (satisfying capacity constraints and flow conservation).

**Step 5: Running time.** Make sure that the flow network you construct is poly-size (that is, the number of vertices is polynomial in the input size of the original problem), so that the Edmonds-Karp algorithm (or, Ford-Fulkerson, should it be more appropriate) are polynomial in the original input size.

---

## An Example: Bipartite Matching

The original problem is: given an *undirected* bipartite graph  $G = (U \cup V, E)$ , find a matching  $M \subseteq E$  of maximum cardinality. We use network flows as a subroutine to solve this problem.

Given the graph  $G$ , we construct a flow network  $H = (W, F)$  as follows:

- the vertices  $W$  are  $U \cup V \cup \{s, t\}$ .
- the *directed* edges  $F$  are  $\{(s, u) : u \in U\} \cup \{(v, t) : v \in V\} \cup \{(u, v) : \{u, v\} \in E\}$ .
- the capacity of each edge in  $F$  is 1.

We find an (integral) maximum flow  $f$  in  $H$ , from which we construct a matching in  $G$  as follows:

- $M = \{\{u, v\} : f(u, v) = 1 \text{ and } u \in U, v \in V\}$ .

**Claim.** *The set  $M$  constructed by the algorithm is indeed a matching in  $G$ , and  $|M| = v(f)$ .*

*Proof.* For each vertex  $u \in U$ , there is one edge  $(s, u) \in F$ , and it is of capacity 1; therefore,  $f^{in}(u) \leq 1$  and by flow conservation,  $f^{out}(u) \leq 1$ . Since our flow is integral, there is at most one edge out of  $u$  that carries flow, and thus at most 1 edge incident to  $u$  that is contained in our matching  $M$ . Similarly, for each  $v \in V$  there is one edge  $(v, t) \in F$ , and thus  $f^{out}(v) \leq 1$  and by flow conservation  $f^{in}(v) \leq 1$ . Again, this implies that at most 1 edge of  $M$  is incident to  $v$ . Therefore,  $M$  is a matching. Lastly, recall that  $v(f) = f(A, \bar{A})$  for any  $s$ - $t$  cut  $(A, \bar{A})$ . If we take  $A = \{s\} \cup U$ , then  $v(f) = f(A, \bar{A}) = |M|$  and we're done.  $\square$

Thus far, we have shown that the algorithm finds a matching of size equal to the value of the maximum flow  $f^*$  (that is, we've shown the maximum matching  $M^*$  has  $|M^*| \geq v(f^*)$ ). If we claim our algorithm finds the maximum matching then we still need to show that  $|M^*| \leq v(f^*)$ .

**Claim.** *For any matching  $M$ , there exists a flow  $f$  in  $H$  such that  $v(f) = |M|$ .*

*Proof.* Given a matching  $M \subseteq E$ , we construct a flow  $f$  as follows: for each  $\{u, v\} \in M$ , set  $f(s, u) = f(u, v) = f(v, t) = 1$ . We need to show this is a flow of value  $|M|$ .

- If  $u \in U$  is not incident to any edge in  $M$  then  $f(s, u) = 0$  and  $f(u, v) = 0$  for all  $v \in V$ ; thus, both capacity constraints and flow conservation are satisfied for these vertices and edges.
- If  $u \in U$  is incident to one edge  $\{u, v\} \in M$  then  $f(s, u) = 1 = f(u, v)$ ; thus, both capacity constraints and flow conservation is satisfied for these vertices and edges.
- It is not possible for  $u \in U$  to be incident to more than one edge in the matching  $M$ .

We can make the analogous arguments for  $v \in V$  and its incident edges. Lastly, the value of the flow  $f$  is the flow out of the source:  $v(f) = |\{f(s, u) : u \in M\}| = |M|$ .  $\square$

**Theorem 1.** *The algorithm is correct and runs in polynomial-time.*

*Proof.* By the first claim we know that our algorithm finds a matching  $M$  such that  $|M| = v(f^*)$ , where  $f^*$  is a maximum flow in  $H$ . Moreover, the second claim tells us that any matching  $M$  has  $|M| \leq v(f^*)$ , and therefore the matching we return is maximum.

Given the original graph  $G$  on  $n$  vertices, we construct a flow network  $H$  with  $n' = n + 2$  vertices and at most  $m' = O(n^2)$  edges. We can use Ford-Fulkerson to find the maximum flow  $f$  in  $H$  in time  $O(m'C) = O(m'n') = O(n^3)$ . Constructing the matching  $M$  is  $O(n)$  time.  $\square$