Notes on Reductions, Cuts and Matchings

In which we describe how to reduce other problems to the max flow problem, and how to analyze a randomized algorithm for finding the minimum cut in an undirected graph.

1 Variations of the Maximum Flow Problem

There are several variations of the maximum flow problem that are efficiently solvable given an algorithm for the standard maximum flow problem. We discuss an example.

In the max flow problem *with node capacities*, we are given a network in which, besides having a capacity $c_{u,v}$ for every edge (u, v), we are also given a capacity c_v for each node. A feasible flow must satisfy the edge capacity constraints and the conservation constraints as in the standard problem, and in addition it must also satisfy *node capacity* constraints that say that the total flow going into each node v (which will be equal to the total flow going out of the node v) must be at most c_v.

Given an instance of this problem, we can construct an equivalent instance of the standard max flow problem in the following way: for each node v other than s, t in the original instance, we create two nodes v_{in} and v_{out} in the new instance, with an edge (v_{in}, v_{out}) of capacity $c_{v_{in},v_{out}} = c_v$. For each edge (u, v) of the original instance, we create an edge (u_{out}, v_{in}) in the new network, of capacity $c_{u_{out},v_{in}} = c_{u,v}$.

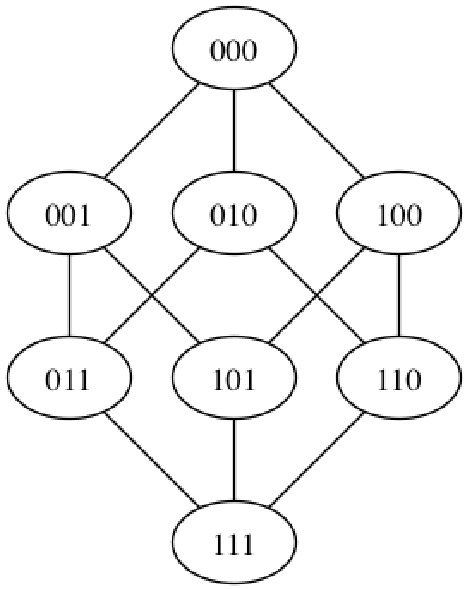
Finally we claim that for every feasible flow of value F in the original network there is a feasible flow of the same value in the new network, and vice versa. To see how, if f is a feasible flow in the original network, we can create a feasible flow f' in the new network by letting $f'_{v_{in},v_{out}}$ be equal to the total flow through the node v according to f', and letting $f'_{u_{out},v_{in}} = f_{u,v}$ for every edge (u, v) of the original network. Conversely, if f is a feasible flow in the new network, we can get a feasible flow f' for the original network by letting $f'_{u,v} = f_{u_{out},v_{in}}$ for every edge (u, v) of the original network.

2 Maximum Matching in Bipartite Graphs

In this section we show applications of the theory of (and of algorithms for) the maximum flow problem to the design an algorithm for finding a maximum matching in a given bipartite graph.

A bipartite graph is an undirected graph $G = (V, E)$ such that the set of vertices V can be partitioned into two subsets L and R such that every edge in E has one endpoint in L and one endpoint in R.

For example, the 3-cube is bipartite, as can be seen by putting in L all the vertices whose label has an even number of ones and in R all the vertices whose label has an odd number of ones.



There is a simple linear time algorithm that checks if a graph is bipartite and, if so, finds a partition of V into sets L and R such that all edges go between L and R: run DFS and find a spanning forest, that is, a spanning tree of the graph in each connected component. Construct sets L and R in the following way. In each tree, put the root in L, and then put in R all the vertices that, in the tree, have odd distance from the root; put in L all the vertices that, in the tree, have even distance from the root. If the resulting partition is not valid, that is, if there is some edge both whose endpoints are in L or both whose endpoints are in R, then there is some tree in which two vertices u, v are connected by an edge, even though they are both at even distance or both at odd distance from the root r; in such a case, the cycle that goes from r to u along the tree, then follows the edge (u, v) and then goes from v to r along the three is an odd-length cycle, and it is easy to prove that in a bipartite graph there is no odd cycle. Hence the algorithm either returns a valid bipartition or a certificate that the graph is not bipartite.

Several optimization problems become simpler in bipartite graphs. The problem of finding a *maximum matching* in a graph is solvable in polynomial time in general graphs, but it has a very simple algorithm in bipartite graphs, that we shall see shortly. (The algorithm for general graphs is beautiful but rather complicated.)

In a undirected graph $G = (V, E)$, a matching is a subset $M \subseteq E$ of the edges such that every vertex of G is an endpoint of at most one edge in M. Said another way, every vertex of the graph (V, M) has degree zero or one. The maximum matching problem is the problem of finding a matching with the largest number of edges. We will describe an algorithm that is based on a reduction to the maximum flow problem. The reduction has other applications, because it makes the machinery of the max flow - min cut theorem applicable to reason about matchings. For example, it would be possible to give a very simple proof of Hall's theorem, a classical result in graph theorem, based on the max flow - min cut theorem.

The problem of finding a *maximum matching* in a graph, that is, a matching with the largest number of edges, often arises in assignment problems, in which tasks are assigned to agents, and almost always the underlying graph is bipartite, so it is of interest to have simpler and/or faster algorithms for maximum matchings for the special case in which the input graph is bipartite.

Consider the following algorithm.

- Input: undirected bipartite graph $G = (V, E)$, partition of V into sets L, R
- Construct a network $(G' = (V', E'), s, t, c)$ as follows:
  - the vertex set is $V' := V \cup \{s, t\}$, where s and t are two new vertices;
  - E' contains a directed edge (s, u) for every $u \in L$, a directed edge (u, v) for every edge $(u, v) \in E$, where $u \in L$ and $v \in R$, and a directed edge (v, t) for every $v \in R$;
  - each edge has capacity 1;
- find a maximum flow $f(\cdot, \cdot)$ in the network, making sure that all flows $f(u, v)$ are either zero or one
- return $M := \{(u, v) \in E \text{ such that } f(u, v) = 1\}$

The running time of the algorithm is the time needed to solve the maximum flow on the network (G', s, t, c) plus an extra $O(|E|)$ amount of work to construct the network and to extract the solution from the flow. In the constructed network, the maximum flow is at most $|V|$, and so, using the Ford-Fulkerson algorithm, we have running time $O(|E| \cdot |V|)$. The fastest algorithm for maximum matching in bipartite graphs, applies a max-flow algorithm called the *push-relabel algorithm* to the above network, and it has running time $O(|V| \cdot \sqrt{|E|})$. It is also possible to solve the problem in time $O(MM(|V|))$, where $MM(n)$ is the time that it takes to multiply two $n \times n$ matrices. (This approach does not use flows.) Using the currently best known matrix multiplication algorithm, the running time is about $O(|V|^{2.37})$, which is better than $O(|V|\sqrt{|E|})$ in dense graphs. The algorithm based on push-relabel is always better in practice.

Remark 1 (Integral Flows) *It is important in the reduction that we find a flow in which all flows are either zero or one. In a network in which all capacities are zero or one, all the algorithms that we have seen in class will find an optimal solution in which all flows are either zero or one. More generally, on input a network with integer capacities, all the algorithms that we have seen in class will find a maximum flow in which all $f(u, v)$ are integers. It is important to keep in mind, however, that, even though in a network with zero/one capacities there always exists an optimal integral flow, there can also be optimal flows that are not integral.*

We want to show that the algorithm is correct that is that: (1) the algorithm outputs a matching and (2) that there cannot be any larger matching than the one found by the algorithm.

Claim 2 *The algorithm always outputs a matching, whose size is equal to the cost of the maximal flow of G'.*

PROOF: Consider the flow $f(\cdot, \cdot)$ found by the algorithm. For every vertex $u \in L$, the conservation constraint for u and the capacity constraint on the edge (s, u) imply:

$$\sum_{r:(u,r)\in E} f(u, r) = f(s, u) \leq 1$$

and so at most one of the edges of M can be incident on u.

Similarly, for every $v \in R$ we have

$$\sum_{\ell:(\ell,v)\in E} f(\ell, v) = f(v, t) \leq 1$$

and so at most one of the edges in M can be incident on v. □

Remark 3 *Note that the previous proof does not work if the flow is not integral*

Claim 4 *The size of the largest matching in G is at most the cost of the maximum flow in G'.*

Proof: Let M^* be a largest matching in G. We can define a feasible flow in G' in the following way: for every edge $(u, v) \in M^*$, set $f(s, u) = f(u, v) = f(v, t) = 1$. Set all the other flows to zero. We have defined a feasible flow, because every flow is either zero or one, and it is one only on edges of G', so the capacity constraints are satisfied, and the conservation constraints are also satisfied, because for every vertex that is not matched in M^* there is zero incoming flow and zero outgoing flow, while for the matched vertices there is one unit of incoming flow and one unit of outgoing flow. The cost of the flow is the number of vertices in L that are matched, which is equal to $|M^*|$.

This means that there exists a feasible flow whose cost is equal to $|M^*|$, and so the cost of a maximum flow is greater than or equal to $|M^*|$. $\square$

So we have established that our algorithm is correct and optimal.