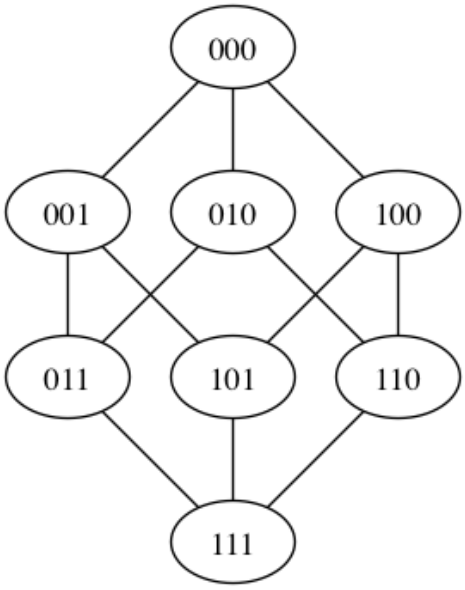
In which we show how to solve the maximum matching problem and the minimum vertex cover problem in bipartite graphs.

Lecture 14

In this lecture we show applications of the theory of (and of algorithms for) the maximum flow problem to the design of algorithms for problems in bipartite graphs.

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.) The algorithm 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. We are going to see a very simple proof of Hall's theorem, a classical result in graph theorem, which uses the max flow - min cut theorem.

As another application, we are going to show how to solve optimally the minimum vertex cover problem in bipartite graphs using a minimum cut computation, and the relation between flows and matchings. In general graphs, the minimum vertex cover problem is NP-complete.

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.

We will describe a way to r*reduce* the maximum matching problem in bipartite graphs to the maximum flow problem, that is, a way to show that a given bipartite graph can be transformed into a network such that, after finding a maximum flow in the network, we can easily reconstruct a maximum matching in the original graph.

1 Maximum Matching in Bipartite Graphs

Recall that, in an undirected graph $G = (V, E)$, a *matching* is a subset of edges $M \subseteq E$ that have no endpoint in common. In a bipartite graph with bipartition (L, R), the edges of the matching, like all other edges, have one endpoint in L and one endpoint in R.

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, which applies the push-relabel algorithm to the network, 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 $|M^*|$. □

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

2 Perfect Matchings in Bipartite Graphs

A *perfect* matching is a matching with $|V|/2$ edges. In a bipartite graph, a perfect matching can exist only if $|L| = |R|$, and we can think of it as defining a bijective mapping between L and R.

For a subset $A \subseteq L$, let us call $N(A) \subseteq R$ the *neighborhood* of A, that is, the set of vertices $\{r \in R : \exists a \in A.(a,r) \in E\}$ that are connected to vertices in A by an edge

in E. Clearly, if there is a perfect matching in a bipartite graph $G = (V, E)$ with bipartition (L, R), then we must have $|A| \leq |N(A)|$, because the edges of the perfect matching match each vertex in A to a distinct vertex in $N(A)$, and this is impossible if $|N(A)| < |A|$.

A classical result in graph theory, Hall's Theorem, is that this is the only case in which a perfect matching does not exist.

Theorem 5 (Hall) *A bipartite graph $G = (V, E)$ with bipartition (L, R) such that $|L| = |R|$ has a perfect matching if and only if for every $A \subseteq L$ we have $|A| \leq |N(A)|$.*

The theorem precedes the theory of flows and cuts in network, and the original proof was constructive and a bit complicated. We can get a very simple non-constructive proof from the max flow - min cut theorem.

PROOF: We have already seen one direction of the theorem. It remains to prove that if $|A| \leq |N(A)|$ for every $A \subseteq L$, then G has a perfect matching.

Equivalently, we will prove that if G does not have a perfect matching, then there must be a set $A \subseteq V$ such that $|A| > |N(A)|$.

Let us construct the network (G', s, t, c) as in the algorithm above, an let us call $n = |L| = |R|$. If G does not have a perfect matching, then it means that the size of the maximum matching in G is $\leq n - 1$, and so the size of the maximum flow in G' is $\leq n - 1$, and so G' must have a cut of capacity $\leq n - 1$. Let us call the cut S.

Let us call $L_1 := S \cap L$ the left vertices in S, and $L_2 := L - S$ the remaining left vertices, and similarly $R_1 := S \cap R$ and $R_2 := R - S$.

In the network G', all edges have capacity one, so the capacity of the cut S is the number of edges that go from S to the complement of S, that is

$$capacity(S) = |L_2| + |R_1| + edges(L_1, R_2)$$

where $|L_2|$ is the number of edges from s to the complement of S, $|R_1|$ is the number of edges from S into t, and $edges(L_1, R_2)$ is the number of edges in E with one endpoint in L_1 and one endpoint in R_2.

This means that we have

$$n - 1 \geq |L_2| + |R_1| + edges(L_1, R_2)$$

and, recalling that $|L_1| = n - |L_2|$,

$$|L_1| \geq |R_1| + edges(L_1, R_2) + 1$$

We can also see that

$$|N(L_1)| \leq |R_1| + edges(L_1, R_2)$$

because the neighborhood of L_1 can at most include $edges(L_1, R_2)$ vertices in R_2. Overall, we have

$$|L_1| \geq N(L_1) + 1$$

and so we have found a set on the left that is bigger than its neighborhood. □

3 Vertex Cover in Bipartite Graphs

The work that we have done on matching in bipartite graphs also gives us a very simple polynomial time algorithm for vertex cover.

- Input: undirected bipartite graph $G = (V, E)$, partition of V into sets L, R
- Construct a network $(G' = (V', E'), s, t, c)$ as before
- Find a minimum-capacity cut S in the network
- Define $L_1 := L \cap S$, $L_2 := L - S$, $R_1 := R \cap S$, $R_2 := R - S$
- Let B be the set of vertices in R_2 that have neighbors in L_1
- $C := L_2 \cup R_1 \cup B$
- output C

We want to show that the algorithm outputs a vertex cover, and that the size of the output set C is indeed the size of the minimum vertex cover.

Claim 6 *The output C of the algorithm is a vertex cover*

PROOF: The set C covers all edges that have an endpoint either in L_2 or R_1, because C includes all of L_2 and all or R_1. Regarding the remaining edges, that is, those that have endpoint in L_1 and the other endpoint in R_2, all such edges are covered by B. □

Claim 7 *There is no vertex cover of size smaller than $|C|$.*

PROOF: Let k be the capacity of the cut. Then k is equal to

$$|L_2| + |R_1| + edges(L_1, R_2)$$

and so

$$k \geq |L_2| + |R_1| + |B| = |C|$$

but k is equal to the capacity of the minimum cut in G', which is equal to the cost of the maximum flow in G' which, by what we proved in the previous section, is equal to the size of the maximum matching in G. This means that G has a matching of size k, and so every vertex cover must have size $\geq k \geq |C|$. $\square$