1 Lectures on March 3 and 10, 2004

We describe Edmond's algorithm for finding matchings of maximum cardinality in a graph. For bipartite graphs with edge weights, we also show how to find matchings of maximum weight. It is recommended that the reader draws pictures of the various constructs that we use (such as alternating trees). I was too lazy to incorporate pictures in these lecture notes.

We recall that bipartite matchings are intersections of two matroids. Some algorithms presented here can serve as examples for the more general case of optimizing over intersections of two matroids.

To a large extent, these lecture notes are based on lecture notes that Laci Lovasz distributed in a course on Combinatorial Optimization in Princeton, 1990.

1.1 Matchings in bipartite graphs

A *matching* in a graph is a set of edges, no two of which share an endpoint. In this section we show how to find maximum matchings in bipartite graphs. The algorithm presented is perhaps not the simplest one possible, but incorporates ideas that will be useful for finding maximum matchings in general graphs.

Recall that a *vertex cover* is a set of vertices that touches all edges. The rest of the vertices in a graph form an independent set. Finding a vertex cover of minimum cardinality is NP-hard. Clearly, in any graph the size of the minimum vertex cover is at least as large as the size of the maximum matching, because the vertex cover must contain at least one vertex from every matched edge. The size of the minimum vertex cover is at most twice as large as the size of any maximal matching (a matching that cannot be extended by any other edge), because taking all vertices involved in a maximal matching is a vertex cover. We note that for complete graphs on an odd number of vertices indeed the size of the minimum vertex cover is twice the size of the maximum matching.

For bipartite graphs, our algorithm will find a matching together with a vertex cover of the same cardinality. This will simultaneously certify the maximality of the matching and the minimality of the vertex cover. This also implies that the size of the minimum vertex cover is equal to the size of the maximum matching in bipartite graphs, and that in bipartite graphs one can find a minimum vertex cover in polynomial time.

Our algorithm will build the matching in phases. Each phase ends when the size of the matching is increased by one. Unlike greedy algorithms, this increase is not simply the consequence of adding an edge to the existing matching, but rather to the addition of a group of edges and the removal of a smaller group of edges. Hence edges that are in the matching at initial phases of the algorithm may end up not being in the final matching.

The mechanism for adding edges is through the use of *alternating paths.*

Definition 1 *Given a matching in a graph, a vertex is* exposed *if it does not participate is the matching. An* alternating path *is a simple path connecting two exposed vertices, such that edges along the path alternate in being out or in the matching. In particular, an edge connecting two exposed vertices is an alternating path.*

A phase ends in our algorithm when an alternating path is found. Then we can flip the edges along the path, removing those that where in the matching from the matching, and putting in the matching the other edges. The size of the matching inceases by 1.

To search for alternating paths, we construct alternating forests.

Definition 2 *A rooted tree in a graph G is an* alternating tree *with respect to a matching M if the root vertex is an exposed vertex, and the edges of every root to leaf path alternate as being out or in M, ending with an edge in M. Vertices at even distance from the root (which in particular include the root itself and all leaves) will be called* outer *vertices, and the other vertices will be called* inner *vertices. An* alternating forest *is a set of disjoint alternating trees that contains all exposed vertices (as the roots of the trees). In particular, the set of all exposed vertices is an alternating forest.*

The algorithm for finding maximum bipartite matchings will have phases. Each phase will have steps. In every step, the algorithm maintains a matching and an alternating forest. A phase ends when an alternating path is found and the size of the maching increases by 1. A step within a phase ends when the number of vertices in the forest increases by 2. In between phases, the size of the alternating forest decreases. Now we are ready to describe the algorithm.

1. Initiallly, the matching M is the empty matching and the alternating forest contains all vertices (they are exposed) as roots of n alternating trees.

2. Repeat as long as possible:

   (a) If there is an edge connecting two outer nodes u and v of the alternating forest, then its endpoints must lie in two different alternating trees (because bipartite graphs have no odd cycles). Let r_1 and r_2 be the roots of these trees. Then the path connecting r_1 to u in the first tree, then following the edge (u, v), and then connecting v to r_2 in the second tree is an alternating path. Use it to increase the size of the matching by 1, and restart a new phase with the set of remaining exposed vertices as the new alternating forest.

   (b) Else, if there is an edge e connecting an outer node u with a node v not in the alternating forest, then there must be a matching edge connected to v, say the edge (v, w). Add the edges (u, v) and (v, w) to the alternating forest. Its cardinality increased by 2.

3. Output the current matching.

The algorithm ends in polynomial time, because the number of phases is at most $n/2$, and the number of steps within a phase is at most $n/2$. Clearly, the output of the algorithm is a matching. It remains to show that this matching is of maximum cardinality. We shall do this by exhibiting a vertex cover of the same cardinality.

Consider the alternating forest that remains at the time that the algorithm ends. Let t denote the number of trees in this forset, i the number of *inner* vertices, o the number of *outer* vertices, and m the number of *other* vertices. Then the size of the matching found is exactly $i + m/2$.

Consider only the subgraph induces on the *other* vertices. It is a bipartite graph, and hence has a vertex cover of size at most $m/2$ (by taking the smaller side of the bipartite graph). Adding to this vertex cover all the *inner* vertices we get a set of $i + m/2$ vertices that form a vertex cover of the whole graph. This is a consequence of the fact that when the algorithm stops, there are no edges connecting *outer* vertices to each other, nor to the set of *other* vertices.

1.2 General graphs

Using an algorithm similar to that of bipartite graphs, there is one more case to consider when the graph may have odd cycles.

- There is an edge (u, v) connecting two outer notes in the same alternating tree. This edge closes an odd cycle in this tree, say of length $2k+1$. Contract all vertices of this cycle to a single vertex w, within the alternating tree and within the original graph, to obtain a new tree T', a new graph G', and a new matching M'. It can be verified that M' is indeed a matching, that $|M'| = |M| - k$, that T' is an alternating tree in G' with respect to the matching M', and that w is an outer vertex in T'.

Due to the above case, we need to change also case 2(a) of the bipartite matching algorithm. When an alternating path is found, we increase by 1 the size of the maximum matching in a graph G', but G' might not be our original graph. We *lift* this matching to a matching in G by reopening vertices w of G' to the odd cycle (of length $2k+1$) from which they were contracted, in reverse order. When doing so, we note that as w was connected to at most one matching edge in the matching for G', we can add k additional edges from the cycle to the matching. As a consequence, we see that also the size of the matching in G increases by 1. We restart a phase *on the original graph G* (forgetting all contractions), with the set of remaining exposed vertices as the alternating forest.

We now show that the algorithm produces a maximum cardinality matching. Consider the alternating forest that remains when the algorithm stops. Each inner vertex is an original vertex of the graph G. The outer vertices may either be original vertices, or contracted vertices. But in any case, they represent connected components of G *with an odd number of vertices.*

Let S denote the set of inner vertices. Let $C_1, C_2 \dots C_c$ denotes the outer vertices, where each outer vertex may correspond to a set of vertices in G. (The number of alternating trees must be $c - |S|$.) Observe that by our stopping condition, if we remove S from G, then each set C_i becomes a connected component of odd cardinality that is disconnected from the rest of the graph. Every matching in the graph must loose at least one vertex from an odd component C_i, unless it connects a vertex from C_i with a vertex of S. As such a connection can be done at most $|S|$ times, every matching misses $c - |S|$ vertices. Observe that the matching we found matches all vertices, except for one exposed vertex in each alternating tree, meaning that it misses only $c - |S|$ vertices, and hence it is optimal.

1.3 Gallai-Edmonds decomposition

The algorithm for maximum matching ends with an alternating forest. Let S (for *Seprator*) denote the set of inner vertices in this forest, let C (for odd *Components*) denote the set of outer vertices, where if an outer vertex is a contracted vertex, then all vertices from which it is derived are in C, and let R denote the set of other vertices (the *Rest*). The arguments of the previous section establish that no vertex from either S or R can be exposed in any maximum matching of G. On the other hand, every vertex from C is exposed in some maximum matching. This can be seen by flipping the edges along the path from the root to this vertex in its alternating tree. If the vertex is lies within a contracted vertex, a similar flipping process may be performed when lifting the contracted vertex to the odd cycle from which it was made.

As a consequence, we can characterize C as the set of all vertices exposed in some matching, S as the neighbor set of C, and R as the rest of the vertices of the graph. Hence in *every* maximum matching, the vertices of R are matched to each other, and the vertices of S are matched to C. The decomposition of the graph into these sets S, C and R is known as the Gallai-Edmonds decomposition of a graph.

We have also implicitly proved Berge's formula which says that the number of exposed vertices in a maximum matching is equal to the maximum of $c - |S|$ over all sets of vertices S, where c is the number of odd components in the graph that remains when S is removed from G.

1.4 Weighted bipartite graphs

Consider a bipartite graph in which edges have arbitrary real weights. Let w_{uv} denote the weight of edge (u, v). We shall show an algorithm for finding a matching M of maximum weight. That is, M is a matching that maximizes $\sum_{(u,v)\in M} w_{u,v}$.

In fact, we shall show an algorithm for a related problem. A *perfect matching* is a matching that touches all vertices. Let $G(U \cup V, E)$ be a complete bipartite graph with equal sides $|U| = |V| = n$ and with edge weights. We shall show an algorithm for finding a perfect matching of minimum weight. As the problem of finding a matching of maximum weight is easily reducible to the problem of finding perfect matchings of minimum weight, and the reduction can be made to preserve bipartiteness, this also gives an algorithm for finding maximum weight matchings in bipartite graphs. (The reduction itself is left as homework. We note that there also is a simple reduction in the reverse direction – from perfect matchings of minimum weight to maximum weight matchings.)

The technique that we shall use for solving the minimum weight perfect matching problem in bipartite graphs is that of weight shifting. This technique will be relatively easy to employ because all perfect matchings have the same cardinality. An important observation that we shall use is that if all edge weights are nonnegative, and there is a perfect matching that uses only 0-weight edges, then this is necessarily a minimum weight matching.

Given an input graph $G(U \cup V, E)$ and edge weights $w : E \longrightarrow R$, our algorithm proceeds as follows.

Step 1: Let $w_{min} = \min_{e\in E} w_e$. Subtract w_{min} from the weight of every edge. Now all edge weights are nonnegative. The weight of every perfect matching changed by exactly $-nw_{min}$. Hence a minimum weight perfect matching in the new graph is also a minimum weight perfect matching in the original graph.

Step 2: As long as there is a vertex $v \in U \cup V$ such that all edges incident to it have positive weight, do the following. Let ϵ_v denote the minimum of these weights. Subtract ϵ_v from the weight of every edge incident with v. All edge weights remain nonnegative, and at least one edge incident with v now has 0-weight. The weight of every perfect matching decreased by exactly ϵ.

When step 2 ends, every vertex has an edge of weight 0 incident with it.

Step 3: Let G' be an unweighted bipartite graph on U and V that contains only the 0-weight edges. Find a maximum cardinality matching in G'. If it is a perfect matching, output this matching as the minimum weight matching and stop.

Step 4: If there is no perfect matching in G', consider its Gallai-Edmonds decomposition into C, S, R as explained in the previous section. Note that for bipartite graphs, all components of C are individual vertices (because there are no odd cycles to contract). Let $\epsilon > 0$ denote the smaller of the following two values: the minimum weight (in G) of an edge from C to R, or half the minimum weight (in G) of an edge with two endpoints in C. We note that $\epsilon > 0$, because G' does not contain edges from C to $R \cup C$. For every vertex in C, decrease by ϵ the weight of every edge incident with it. For every vertex in S, increase by ϵ the weight of every edge incident with it. We note that the weight of edges between C and S does not change, that no edge weight becomes negative, and that at least one edge between C and $C \cup R$ now has 0 weight. The weight of every perfect matching changed by exactly $\epsilon(|S| - |C|)$. Go back to step 3.

This completes the description of the algorithm. All weight shifts that we did had exactly the same effect on the total weight of every perfect matching in G. It follows that when the algorithm stops, the matching that is output is necessarily a perfect matching of minimum weight in the original graph.

It remains to show that the algorithm does indeed stop, and in polynomial time. For this we need to show that step 4 cannot be repeated forever. First we note that the cardinality of the matching found in two consecutive applications of step 4 does not decrease. The reason is that in the first of these applications, all vertices of S were matched to vertices in C. As the weight of these matched edges does not change in the second application, they can be taken also as part of the second matching. All other matched edges in the first application were within R, and they too can be taken in the second application because their weight did not change either. This argument also shows that if the cardinality of the maximum matching did not grow, then the set C of the second application (that of vertices that are exposed in some maximum matching) contains the set C of the first application. And as now C has a fresh neighbor, the size of the set S grows by at least one. As $|S| < n$, it takes at most n invocations of step 4 until the cardinality of the matching grows, showing that altogether step 4 is repeated at most n^2 times.

1.5 A dual problem

Given a complete bipartite graph $G(U \cup V, E)$ with edge weights $w : E \longrightarrow R$, a cost function on the vertices $c : U \cup V \longrightarrow R$ is *feasible* if for every edge (u, v), $c(u) + c(v) \leq w_{u,v}$. We are interested in finding a feasible cost function that maximizes $\sum_{v \in U \cup V} c(v)$.

Let M be an arbitrary perfect matching in G. By summing the feasibility condition $c(u) + c(v) \leq w_{u,v}$ over all edges of M, we obtain that

$$\sum_{v \in U \cup V} c(v) \leq \sum_{e \in M} w_e.$$

Hence the maximum over all feasible cost functions of the sum of costs of vertices is at most the weight of the minimum weight perfect matching.

We exhibit a cost function c with sum of costs equal to the weight of the minimum weight matching. Initially, all vertex costs are set to 0. Follow the steps of the algorithm of the previous section, and in each step update the costs of vertices as follows. In step 1, give every vertex cost $w_{min}/2$. In step 2, add ϵ_v to the cost of vertex v. In step 4, add ϵ to the cost of every vertex of C, and subtract ϵ from the cost of every vertex of S.

In every step, the weight decrease of an edge is equal to the sum of increase in cost of its two endpoints. As the minimum perfect matching has final weight 0, it follows that the costs we added to the vertices exactly equal the original weight of this matching. Moreover, as every edge has final nonnegative weight, the cost function is feasible.

Given a complete bipartite graph $G(U \cup V, E)$ with edge weights $w : E \longrightarrow R$, we established that for feasible cost functions c and perfect matchings M,

$$\max_c \sum_{v \in U \cup V} c(v) = \min_M \sum_{e \in M} w_e.$$

This was first proved by Egervary in 1931.

1.6 A note on minimum maximal matchings

The following problem appears to be related to that of finding a minimum weight perfect matching. The input is a graph (edges are unweighted), the output is a maximal matching (one that cannot be extended), and the goal is to minimize the cardinality of the maximal matching found. However, unlike the other problems addressed in this lecture, this problem is NP-hard.

We show a reduction from the problem of deciding whether a graph $G(V, E)$ has a vertex cover with k vertices. Add to G a set I of k vertices, a set F of k vertices, and all edges between F and $I \bigcup V$. This new graph G' has a maximal matching of cardinality k iff the G has a vertex cover of size at most k.

Let $S \subset V$ be a vertex cover of size k for G. Then a perfect matching between F and S is a maximal matching for G'. Conversely, let M be a maximal matching in G'. Then it must cover at least one endpoint of every edge between I and F, requiring at least k vertices. Hence at most k vertices participating in the matching are from V. These vertices must form a vertex cover in G, as otherwise an edge of G can extend the matching.

The NP-hardness of minimum maximal matching holds also for bipartite graphs, though the proof is more complicated (or at least I do not see a proof as simple as the above).