Lectures 4 and 5 – Matchings

Uriel Feige
Department of Computer Science and Applied Mathematics
The Weizman Institute
Rehovot 76100, Israel
uriel.feige@weizmann.ac.il

November 30 and December 7, 2017

1 Overview

A *matching* in a graph is a set of edges, no two of which share an endpoint. 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 observe that bipartite matchings are intersections of two (partition) matroids. Some algorithms presented here can serve as examples for the more general case of optimizing over intersections of two matroids. Finding the maximum intersection of three matroids includes the problem of 3-dimensional matching as a special case, and hence is NP-hard.

Some algorithmic principles that we shall use are alternating paths and primal-dual algorithms.

2 Maximum cardinality matchings in bipartite graphs

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 in general graphs is NP-hard.

Definition 1 *A maximization problem A and a minimization problem B are* weak duals *of each other if on every input instance the value of A is not larger than the value of B. They are* strong duals *of each other if on every input instance the value of A is equal to value of B.*

Duality is a useful concept. If max-A and min-B are duals to each other (whether weak or strong), then any feasible (not necessarily optimal) solution for A gives a lower bound on the optimal solution for B, and any feasible (not necessarily optimal) solution

for B gives an upper bound on the optimal solution for A. Moreover, given two feasible solutions of equal value, one for A and the other for B, they must both be optimal. If two NP-problems are strong duals of each other then both of them lie in $NP \cap co - NP$. As a rule of thumb (though this is not a formal theorem), in such cases both problems can be solved in polynomial time.

Proposition 1 *Maximum matching and minimum vertex cover are weak duals of each other. Namely, in every graph the size of the minimum vertex cover is at least as large as the size of the maximum matching.*

Proof: A vertex cover must contain at least one vertex from every matched edge. □

Maximum matching and minimum vertex cover are not strong duals of each other. However, 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.

Theorem 2 *In bipartite graphs, maximum matching and minimum vertex cover are strong duals of each other. Moreover, both a maximum matching and a minimum vertex cover can be found in polynomial time.*

The rest of this section contains an algorithm proving Theorem 2.

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 due to the addition of 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 2 *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.*

Proposition 3 *Given a matching that is not of maximum cardinality in a graph (whether bipartite or not), the graph contains an alternating path.*

Proof: Given a graph $G(V, E)$, let M be a maximum cardinality matching and let M' be a matching of smaller cardinality. Let $G(V, E')$ be the subgraph of G that contains all vertices of G and only those edges that participate in exactly one of the two matchings M and M'. As every vertex has degree at most 2 in G', it decomposes into paths and cycles. All cycles must be of even length (as edges from M and M' need to alternate). As E' has more edges coming from M' than from M, at least one of the paths contains more edges from M than from M', and hence it is an alternating path in the sense of Definition 2. □

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

To search for alternating paths, we construct alternating forests.

Definition 3 *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 matching 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. Initially, 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 forest, 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 every bipartite graph on m vertices 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.

3 Maximum cardinality matching in general graphs

The algorithm presented for bipartite graphs might fail to find a maximum matching in a graph that is not bipartite. As an example, consider a graph on four vertices in which vertices v_1, v_2, v_3 form a triangle, and in addition there is the edge (v_1, v_4). The maximum matching is $(v_1, v_4), (v_2, v_3)$. However, the bipartite algorithm might in the first phase find the edge (v_1, v_2), and in the second phase end with an alternating forest composed of the alternating tree $v_3 - v_1 - v_2$ and the exposed vertex v_4, and then get stuck.

Consequently, for graphs that have odd cycles, we add to the algorithm an additional type of step (within a phase):

- Suppose that there is an edge (u, v) connecting two outer nodes 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 (whose neighbors are those vertices that are neighbors of the original odd cycle), 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 Step 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 C_w (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 C_w to the matching. This argument can be applied recursively: if some vertex w' of C_w is by itself a contracted vertex representing an odd cycle $C_{w'}$, then as w' is connected to at most one matching edge, the rest of the vertices on $C_{w'}$ can be matched along the cycle. 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.

In the new algorithm, the number of steps within a phase is at most n: at most $n/2$ steps within a phase connect an outer node to a node not in the alternating forest, and at most $n/2$ steps contract a cycle. The number of phases is at most $n/2$. Hence the algorithm runs in polynomial time.

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 (perhaps with recursive contractions). 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 lose 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.

3.1 Gallai-Edmonds decomposition

The algorithm for maximum matching ends with an alternating forest. Let S (for *Separator*) 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 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 establishes a strong dual for maximum matchings (technically, for the equivalent problem on minimizing the number of exposed vertices) in general graphs.

Theorem 4 *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.*

4 Maximum weight matching in 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 first present a dual problem to the minimum bipartite perfect matching 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 $\sum_{v \in U \cup V} c(v)$.

Proposition 5 *Given a complete bipartite graph $G(U \cup V, E)$ with edge weights $w : E \longrightarrow R$ and with $|V| = |U|$, the minimum weight perfect matching problem and the maximum feasible vertex cost problem are weak duals of each other.*

Proof: 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

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

□

Egervary in 1931 established strong duality.

Theorem 6 *Given a complete bipartite graph $G(U \cup V, E)$ with edge weights $w : E \longrightarrow R$ and with $|V| = |U|$, the minimum weight perfect matching problem and the maximum feasible vertex cost problem are strong duals of each other. Namely, for feasible cost functions c and perfect matchings M,*

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

Egervary's equality is a special case of duality between covering (a perfect bipartite matching covers all vertices) and packing problems, and moreover, establishes also that the primal problem has an integer optimal solution (which is not the case for more general covering problems).

We shall present a polynomial time algorithm that proves Egervary's theorem. Before presenting the algorithm, we provide some useful observations. Given a feasible cost function c for the vertices, the *slackness* of edge (u, v) is defined as $w(u, v) - c(u) - c(v)$. Namely, the slackness measures how for being tight is the constraint on the vertex costs implied by the edge. An edge with slackness 0 is referred to as *tight*. The following lemma is a special version of a more general framework of *complementary slackness* conditions that apply to pairs of primal and dual problems.

Lemma 7

- *Suppose that c is a feasible cost function of value equal to the weight of the minimum weight perfect matching. Then every minimum weight perfect matching is composed only of edges that are tight with respect to c.*
- *Suppose that there is a feasible cost function c and a perfect matching that is composed only of edges that are tight under c. Then the cost function must be of maximum weight and the matching must be of minimum weight.*

Proof: The first item follows by considering a minimum weight perfect matching M in the proof of Proposition 5, and noticing that to attain equality in $\sum_{v \in U \cup V} c(v) \leq \sum_{e \in M} w_e$ we must have equality in all constraints $c(u) + c(v) \leq w_{u,v}$ for edges of M. The second item follows because it exhibits a cost function and a matching of equal value (summing up costs of vertices along the edges of the matching), and then weak duality implies that they both must be optimal. □

The algorithm that we shall present is an example of a primal-dual algorithm: updating the weights of the vertices, the dual problem, helps us decide which edge to put in the matching, the primal problem. The primal-dual algorithm that we shall use for solving the minimum weight perfect matching problem in bipartite graphs involves iteratively constructing feasible dual solutions of increasing costs, and in the process tracking the slackness of the constraints for the primal problem. We may think of this slackness as new weights for the edges with respect to which we seek a minimum weight perfect matching. 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. Hence the candidate edges to participate in a perfect matching are the 0-weight edges, or in our context where weight corresponds to slackness, those edges for which the corresponding inequality for the dual solution is tight. This is very much related to Lemma 7.

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

1. Let $w_{min} = \min_{e \in E} w_e$.

   Dual problem. Give every vertex cost $w_{min}/2$. This is a feasible cost function of total value nw_{min}.

   Primal problem. Subtract w_{min} from the weight of every edge. The resulting weight of an edge (u, v) is its *slackness*, namely, measures the slackness in the inequality $c(u) + c(v) \leq w_{uv}$. 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.

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 v be an arbitrary such vertex and let ϵ_v denote the minimum weight of an edge incident with v.

   Dual problem. Add ϵ_v to the cost of vertex v. The cost of the dual increased by ϵ_v and the dual remains feasible.

   Primal problem. 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_v.

3. Let G' be an unweighted bipartite graph on U and V that contains only the 0-weight edges. This graph spans all of U and V (by the end of Step 2 every vertex has an edge of weight 0 incident with it). Find a maximum cardinality matching in G'. If it is a perfect matching, output this matching and stop.

4. If there is no perfect matching in G', consider its Gallai-Edmonds decomposition into C, S, R as explained in Section 3.1. Note that for bipartite graphs, all components of C are individual vertices (because there are no odd cycles to contract). Let ϵ denote the minimum of weight of edges (in G) from C to R, or half the minimum weight of edges from C to C, whichever is smaller. We note that $\epsilon > 0$, because G' does not contain edges from C to $C \cup R$.

Dual problem. Add ϵ to the cost of every vertex of C, and subtract ϵ from the cost of every vertex of S. The cost of the dual increases by $\epsilon(|C| - |S|)$. By the absence of edges in G' from C to $C \cup R$ and the choice of ϵ, this dual remains feasible.

Primal problem. For every endpoint of an edge incident with C, decrease by ϵ the weight of the respective edge. For every endpoint of an edge incident with S, increase by ϵ the weight of the respective edge. 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 or between C and 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.

Let us first show that the algorithm ends in polynomial time. For this we need to bound the number of times that Step 4 is executed. Note that the cardinality of the matchings found in two consecutive applications of Step 3 does not decrease. To see this, consider Step 4 that was executed in between the two applications of Step 3. 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 Step 4, 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. Given that in the second application of Step 3 there is at least one new edge that previously connected C to $C \cup R$, then either the cardinality of the maximum matching grows (if an alternating path is found), or the size of the alternative forest grows by at least two vertices (the worst case being when the new edge is between C and R and no more edges can be added to the forest). Hence 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.

We now show that the matching returned by the algorithm is optimal. This follows from the second item in Lemma 7, because the algorithm returns a feasible cost function c for the vertices and a perfect matching that is composed only of tight edges.

(Alternatively, optimality can be proved without explicitly referring to the dual solution. When the algorithm ends, the matching found is of minimum weight (weight 0) in the end graph. All weight shifts that we did had exactly the same effect on the total weight of every perfect matching in G. Hence the final output is necessarily a perfect matching of minimum weight in the original graph.)