LECTURES 3 & 4: Matchings

1 Matching

Chapter 10, Section 1

A *matching* in a graph is a subset of the edges such that no vertex is incident to more than one edge in the subset. It is a sort of *mating* or *coupling* of vertices in pairs, often necessarily leaving some vertices uncoupled, unmatched. In case there are no weights on the edges we just like to find a matching with a maximum number of edges, maximum number of couples, minimum number of unmatched vertices.

MATCHING.
Instance: Graph $G = (V, E)$.
Question: Find a largest cardinality matching in G.

As in Max Flow, there is a very nice characterisation of an optimal solution in terms of augmenting paths. (I draw an example on the blackboard of a graph and a matching in the graph.) We call a path an *alternating path* with respect to matching M, if it consist alternatingly of edges in the matching M and edges not in the matching M. We call a vertex *exposed* w.r.t. M if it is unmatched in M. We call an alternating path between two exposed vertices an *augmenting path* w.r.t. M. Clearly, such an augmenting path P starts and ends with an edge not in M, hence it must have an odd number of edges. Given such an augmenting path P it is easy to see how to improve the matching: add to M all edges of $P \setminus M$ (edges on P that are not in M) and delete from M all edges of $M \cap P$ (edges on P that are in M).

Lemma 1. *Given a graph G and a matching M of G. If there exists an augmenting path P in G w.r.t. M, then $M' = (M \setminus P) \cup (P \setminus M)$ is a matching of G with $|M'| = |M| + 1$.*

Proof. As we noticed, P has one more edge not in M than edges in M, which proves $|M'| = |M| + 1$. Remains to prove that M' is indeed a matching. Suppose it is not. Then M' must contain two edges, e and e' say, incident to the same vertex, v say.

These two edges cannot belong both to $M \setminus P$, since M is a matching and therefore does not contain two edges incident to the same node. Since P

is an alternating path, e and e' cannot belong both to $P \setminus M$.

Hence, the only possibility that remains is $e \in M \setminus P$ and $e' \in P \setminus M$. Since v is incident to $e \in M \setminus P$ it is not exposed w.r.t. M. Because $e' \notin M$ and path P is an alternating path, v must be incident to an edge e'' on P which is also in the matching: $e'' \in P \cap M$. Hence, in M, v is incident to $e \in M$ and $e'' \in M$, which is in contradiction with M being a matching. □

Theorem 1. *Given a graph G and matching M. M is a maximum matching if and only if there does not exist an augmenting path w.r.t. M in G.*

Proof. That there does not exist an augmenting path w.r.t. a maximum matching is obvious from Lemma 1. The other way is a beautiful proof that reveals the structural properties of matchings.

Suppose again the opposite: there is no augmenting path w.r.t. to M but M is not maximum. Then there exists an optimal matching M' with $|M'| > |M|$. Consider the graph on V with edges in $M \oplus M' =: (M \setminus M') \cup (M' \setminus M)$; i.e., $G = (V, M \oplus M')$. Since any vertex cannot be incident to two edges in a single matching, each vertex in G has degree 2 or less. If it has degree 2 then one of the edges is in M and the other in M'. Thus, all components of G consist of even length circuits and paths, that alternatingly contain edges of M and M'. Since the circuits have as many edges from M as from M', it must be that there exists a path that contains more edges from M' than from M (remember $|M'| > |M|$). But then this is an augmenting path w.r.t. M, contradicting our assumption. □

Given this characterisation one would guess the following algorithm for MATCHING: Start with an empty matching. In each iteration find an augmenting path w.r.t. to the current matching. If no such path exists, the maximum matching has been found.

Indeed, this is essentially the algorithm that solves cardinality matching problems. However, there is a complication in the algorithm: finding an augmenting path. This complication only occurs in the presence of cycles of odd length, so-called *odd cycles* in the graph. Therefore we concentrate on graphs that do not have such cycles.

1.1 Bipartite Matching

Chapter 10, Sections 2,3

Theorem 2. *A graph is bipartite if and only if it has no odd cycles.*

Exercise 4.1. Prove this theorem.

Given a bipartite graph $G = (V, U, E)$ and a matching M, leaving exposed vertices $V_0 \subset V$ and $U_0 \subset U$. To find an augmenting path we propose to do a form of *Breadth-First-Search*: Start in all vertices of V_0. In each iteration, from the V-vertices just reached, find their neighbours in U, skipping those that have been found before. If one of them happen to be a vertex from U_0 we have found an augmenting path. Otherwise, all vertices U reached are matched, and on any augmenting path the next edge must be an edge from the matching. Thus, from the U vertices we jump immediately to their V-mates in the matching. We call such vertices *outer-vertices.* The U-vertices in this search tree we call *inner-vertices.* (Notice that U-vertices will never be outer-vertices in the search for augmenting paths.)

From these V-mates we do the further search to neighbouring vertices from U, omitting vertices that have been found before. We proceed, until we find an exposed U_0-vertex, in which case we have found an augmenting path, or until no new U-vertices can be reached, in which case we conclude that there is no augmenting path and declare the present matching as the maximum matching.

Theorem 3. *The augmenting path algorithm finds a maximum matching in a bipartite graph in $O(|E| \min\{|U|, |V|\})$.*

Proof. We just need to prove that if there is an augmenting path, the algorithm will detect it. Suppose there is such a path, but it is not found. Then, starting from the exposed V_0-vertex, there must be some earliest U-vertex on the path that is not detected from the preceding V-vertex on the path. But this can only occur if it was detected before, hence already "visited", being adjacent to another matched V-vertex, which in its turn must have been reached before by an alternating path starting at an exposed node. Hence, this would yield an alternative augmenting path.

For the analysis of the running time, see the proof of Theorem 10.2 in [PS]. □

In [PS] the authors construct an auxiliary graph for finding the augmenting paths, but the algorithm is essentially the same as the one I presented.

Read for yourself in [PS] Section 10.3 how the bipartite matching problem can be transformed into a max flow problem.

Like Max Flow had the min-max relation to Min Cut, also Bipartite Matching has a beautiful min-max relation to a famous other problem:

Vertex Cover.
Instance: Graph $G = (V, E)$.
Question: Find a minimum cardinality vertex cover in G; i.e. a minimum cardinality subset of the vertices such that each edge has at least one of its incident vertices in the subset, each edge is *covered.*

Theorem 4. *Given a bipartite graph $G = (V, U, E)$ the size of a maximum matching is equal to the size of a minimum vertex cover.*

In Exercise 1 you are asked to prove this theorem. You should be able to do this. As usual one side is easily argued: the size of a maximum matching is less than or equal to the size of any vertex cover. For the other direction one should think of a similar construction as used in the proof of the max-flow min-cut theorem. If you like, you may be helped by the network flow representation of the problem in Section 10.3 and be able to see that a min-cut there corresponds to a minimum vertex cover.

The reason why this augmenting path search does not work on non-bipartite graphs is best illustrated at the hand of an example, which I draw on the blackboard, and looks like in [PS], Figure 10.8. By such an odd cycle, which is incident to a matched edge not in the cycle, all vertices, not only the ones on the cycle, can both be inner vertices and outer-vertices, for the ones on the cycle depending on the route taken. The BFS that we proposed before will fail, since it may find alternating paths from exposed nodes to exposed nodes that along which no augmentation is possible (see for an example Figure 10-5 in [PS] and the text at page 227). Hence we can only be successful in finding augmenting paths if we keep all the possibilities open for using any of the vertices on the odd cycle as potential outer-vertex. That is indeed what is proposed in the so-called blossom-algorithm. This algorithm and its analysis is very technical and we skip it.

Instead we will study *weighted* bipartite matching. There are non-negative weights on the edges of the graph and the objective is to find a matching of maximum total weight.

1.2 Weighted Bipartite Matching

Chapter 11, Section 1 and Chapter 13, Section 2

Weighted Bipartite Matching:
Instance: Bipartite graph $G = (V, U, E)$ and weights $w : E \to \mathbb{R}_+$.
Question: Find a matching $M \subset E$ in G of maximum total weight.

We start by studying an essential property of the linear programming relaxation of the obvious integer 0-1 programming formulation of this problem.

$$\begin{aligned} &\max \textstyle\sum_{e\in E} w_e x_e \\ &\text{s.t. } \textstyle\sum_{e\ni v} x_e \le 1,\ \forall v \in V; \\ &\textstyle\sum_{e\ni u} x_e \le 1,\ \forall u \in U; \\ &x_e \ge 0 \qquad \forall e \in E. \end{aligned} \tag{1}$$

As I will show, all basic feasible solutions are integral. Thus, this problem can be solved by solving the LP problem. Solving LP's by the Simplex algorithm is not a theoretically efficient method. Instead, for solving the weighted bipartite matching problem we will use a primal-dual algorithm.

A matrix is totally unimodular (TUM) if each of its square submatrices has determinant -1, 0 or 1.

Theorem 5. *Any basic feasible solution of the system $\{Ax = b,\ x \ge 0\}$ is integral if A is TUM and b is an integral vector.*

Exercise 4.2. Prove this theorem using your knowledge of basic feasible solutions of LP's combined with Linear Algebra, in particular solutions of systems of (independent) linear equations.

If I am right TUM is a property that is mentioned in the course Bedrijfseconometrie 1, where sufficient conditions for a matrix being TUM are given. I will prove here directly that the left-hand side matrix of (1) is TUM.

The matrix is the incidence matrix of the bipartite graph with the edges for the columns and the first n rows corresponding to the vertices of V and the second n rows corresponding to the vertices of U. Each column corresponds to a variable x_{ij} and has entries 1 in the row corresponding to vertex $i \in V$ and a 1 in the row corresponding to vertex $j \in U$. The other entries are all 0.

Theorem 6. *The incidence matrix of a bipartite graph is TUM.*

Proof. We prove that the matrix is TUM by induction on the size of the square submatrices. From the description of the matrix above it is clear that all 1×1 submatrices have determinant 0 or 1. Suppose now that all $(n-1)\times(n-1)$ square submatrices have determinants -1, 0 or 1. Consider a $n \times n$ square submatrix B. We distinguish 3 cases:

- If B contains a column with only 0-entries, then $\det(B) = 0$.
- If B contains a column with exactly one 1 entry, then we use the induction hypothesis to assert that $\det(B) = 1 \cdot \pm 1 \det B'$, with B' the matrix obtained by deleting the row and column of this 1-entry.
- If B has in every column two 1-entries, then in each column it has a 1-entry in a V-row and a 1 entry in a U-row. Summing the V-rows of B gives an all-1-vector, and summing the U-rows of B gives an all-1-vector. Hence the rows of B are linearly dependent, implying that $\det(B) = 0$. $\square$

Exercise 4.3. Prove Theorem 4 again using the TUM-property and strong LP-duality.

A primal-dual algorithm

Chapter 11, Sections 1,2 and Chapter 7, Section 4

The *Hungarian algorithm* for this problem is a primal-dual algorithm. It is easy to see that we may concentrate on complete bipartite graphs, since missing edges can be added with weight 0. Similarly we may concentrate on bipartite graphs with $|V| = |U| = n$, since otherwise the smaller side is supplemented with vertices that are connected to all vertices on the other side with edges of weight 0. In doing so we may then restrict to finding a maximum weight *perfect matching.* (A matching is perfect if all vertices are matched.)

Now we are in a position to transform the problem into a minimization problem: Choosing a large enough W, e.g. $W = \max_{\{i,j\}\in E} w_{ij}$, and setting *costs* $c_{ij} = W - w_{ij}$ the problem becomes finding a perfect matching minimizing total costs. This problem is known under the name ASSIGNMENT PROBLEM.

We will solve this problem by a so-called *primal-dual algorithm*. The minimum weight perfect matching problem has the following LP formulation:

$$\begin{aligned}
&\min \textstyle\sum_{i=1}^n \sum_{j=1}^n c_{ij}x_{ij} \\
&\text{s.t. } \textstyle\sum_{j=1}^n x_{ij} = 1, \qquad i = 1,\dots,n; \\
&\qquad \textstyle\sum_{i=1}^n x_{ij} = 1, \qquad j = 1,\dots,n; \\
&\qquad x_{ij} \geq 0 \qquad\qquad i = 1,\dots,n,\ j = 1,\dots,n.
\end{aligned} \tag{2}$$

This algorithm works with dual feasible and primal infeasible solutions, always respecting the complementary slack conditions. The dual problem of LP (2) is

$$\begin{aligned}
&\max \textstyle\sum_{i=1}^n a_i + \sum_{j=1}^n b_j \\
&\text{s.t. } a_i + b_j \leq c_{ij}, \qquad i = 1,\dots,n,\ j = 1,\dots,n
\end{aligned} \tag{3}$$

The dual variables are unrestricted in sign. Variable a_i corresponds to vertex $i \in V$, and variable b_j corresponds to vertex $j \in U$. The complementary slack conditions are

$$(c_{ij} - a_i - b_j)x_{ij} = 0,\ i = 1,\dots,n,\ j = 1,\dots,n \tag{4}$$

We could start with the dual feasible solution $a_i = b_j = 0\ \forall i,\ \forall j$. In order to satisfy complementary slack all x_{ij} must be 0. Now we raise each of the b_j values, leaving all $a_i = 0$, until $a_i + b_j$ becomes equal to c_{ij} for some i, j combination. Thus, we set $b_j = \min_i c_{ij},\ \forall j$.

In that case we obtain at least n edges that become *admissible*, in the sense that $c_{ij} - a_i - b_j = 0$ and therefore complementary slack allows to raise the x_{ij} values of these edges. Call the set of admissible edges J. Then we like to find a maximum matching, unweighted, in the graph $G_J = (V, U, J)$. This can be done by using the (unweighted) bipartite cardinality matching algorithm just explained before.

.................Insert here an example ....#

If the maximum matching in G_J happens to be a perfect matching then we have a primal feasible solution for the original problem. But in that case we have a primal and a dual feasible solution pair that satisfy the complementary slack conditions and therefore must be an optimal pair.

So let us assume that the maximum matching in G_J is not a perfect matching. Thus, at some point the search for an augmenting path starting in exposed nodes in V stops unsuccessfully. At this moment we label all vertices on alternating paths starting in exposed V-vertices. We call the set of labeled vertices in V and U, respectively V^* and U^*. Then we like to make more edges admissible and preferably edges that would create augmenting paths together with edges already in J.

We wish to raise some of the a_i and b_j values, taking care that we cannot increase both on edges $\{i, j\} \in J$, since this would give an infeasible dual solution. We wish to do this raising of values to make edges admissible that can enlarge the search tree. In the algorithm the choice is made to increase by a fixed value θ, to be chosen later, the values a_i for all labeled vertices $v_i \in V^*$ and the values b_j for all unlabeled vertices $u_j \in U \setminus U^*$. To keep all essential edges in J, i.e., all edges that are in the current search tree, all the edges in the current matching, and all the edges that could potentially be on an augmenting path w.r.t. the current matching, we decrease a_i for vertices $v_i \in V \setminus V^*$ and decrease the b_j of vertices $u_j \in U^*$ again all with the same value θ.

Let us check that if an edge $\{v, u\} \in J$ is between two vertices in the search tree, then it remains in J. This is because v is labeled, hence its corresponding a-value is increased, and u is labeled, hence its corresponding b-value is decreased, both with the same amount. Similarly, you can check that the same holds for any edge $\{v, u\} \in J$ between two vertices not on the search tree. An edge $\{v, u\} \in J$ such that u is on the search tree (hence matched) and v is not on the search tree, is an edge that will not appear in any augmenting path giving the current matching. Indeed, both its a-value and its b-value will decrease and it will not remain in J. Notice that it cannot happen that that $\{v, u\} \in J$ with v in the search tree and u not in the search tree, since then the search tree could have been extended. These latter edges are the ones that we would like to add to J!

In order to remain dual feasible, we must make sure that dual constraints corresponding to the edges that have an increase both on their a_i and their b_j value keep being satisfied. Therefore we set

$$\theta = \min_{v_i \in V^*, u_j \in U \setminus U^*} \frac{1}{2}(c_{ij} - a_i - b_j). \tag{5}$$

This implies that after the increases (and decreases) at least one extra edge $\{v_i, u_j\}$ with $v_i \in V^*$ and $u_j \in U \setminus U^*$ will become admissible, thus extending the alternating paths from exposed vertices.

Then we repeat the search for an augmenting path in the graph G_J. If such a path is found, augment the matching. If not, at least one new U-node and one new V-node become labeled. Thus after at most n such extensions of J, due to dual modifications, we will find an augmenting path. Let us call the set of dual modifications between any two consecutive augmentations a *stage*. Then we just concluded that any stage takes at most n dual modifications. In each such modification computing θ can be implemented to take $O(n)$ operations. Searching for an augmenting path in G_J can be implemented to take $O(n)$ operations (we do not have to start after each modification from scratch). Altogether, a stage takes $O(n^2)$ operations.

Clearly there are at most $O(n)$ stages, since every stage ends with an augmentation and the perfect matching has n edges.

Once more, verify for yourself that once a perfect matching in G_J is reached, then this perfect matching is a primal feasible solution of the original problem that together with the dual feasible solution satisfies the complementary slack conditions, and hence both primal and dual solutions are optimal.

1.3 Material and Exercises

[PS] Bipartite Cardinality Matching. Chapter 10: Sections 10.1-10.3, Section 10.4 (just read the difficulties with finding augmenting paths because of *blossoms.*

[PS] Bipartite Weighted Matching (the ASSIGNMENT PROBLEM). Chapter 11: Sections 11.1 and 11.2. Here the book gives only a very partial description of the Hungarian Method. The exact description of the algorithm for the more general Hitchcock Problem is found in Chapter 7, Section 7.4.

[PS] Total Unimodularity. Chapter 13: Section 13.2

Exercises:
From Chapter 10: Exercises 1,2,3,14 (if you prefer you may restrict in Exercise 14 to bipartite graphs)

From Chapter 11: Exercises 1,2,7,8