Matching and Assignment problems

MA428, Dr Katerina Papadaki

I. Introduction

In this lecture we will cover the following:

1) Problem definition
2) LP Formulations
3) Bipartite Cardinality Matching Problem
4) Bipartite Weighted Matching Problem - Assignment problem
5) The marriage problem and stable matchings
6) The Matching Algorithm
7) Applications

The material covered in this lecture can be found in chapter 12 of AMO.

II. Problem definition

In practice we often need to pair objects together: for example we might need to assign jobs to machines subject to some constraints like machine A can only perform jobs 3 and 4. The objective could be to maximize the number of pairs or assignments or it could be to minimize a cost that corresponds to each assignment (or maximize a gain/weight that corresponds to each assignment). Matchings are assignments of a pair of objects.

K. Papadaki is with the Department of Management, Operational Research Group at London School of Economics, United Kingdom

E-mail: k.p.papadaki@lse.ac.uk

A. Preliminaries

Consider an undirected graph $G = (N, E)$ made up of $n = |N|$ nodes and $m = |E|$ edges. We adopt the notational convention that edge (i, j) can also be written as (j, i): $(i, j) = (j, i)$.

Definition Given a graph $G = (N, E)$, the *degree* of a node $i \in N$ is the number of edges adjacent to node i.

Definition A *matching* $M = (N, E')$ in $G = (N, E)$, with $E' \subseteq E$, is a sub-graph of G that contains all the nodes in G and with the property that every node in M has degree zero or one.

We often describe a matching M by its edges E', since the edges in E' are enough to characterize the matching.

In a matching M, no two edges are adjacent to the same node. Further, a matching M will contain a number of disconnected components some of them will have one edge connecting two nodes and some will only have one node. Thus, a matching pairs together (matches) two nodes by having an edge between them. An example of a matching M on a graph G is shown in Figure 1, where the edges of M are shown in solid lines. This matching has edges $(1, 4)$, $(3, 5)$ and thus it has assigned node 1 to node 4, and node 3 to node 5. No assignment was made for node 2.

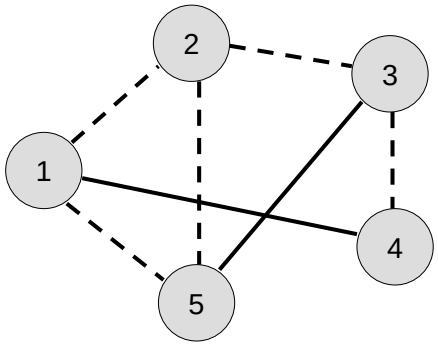


Fig. 1. A matching M on a graph G, where the edges of the matching are shown in solid lines: $M = \{(1, 4), (3, 5)\}$

Definition The *cardinality* of a matching M is the number of edges in M.

For example, the cardinality of the matching shown in Figure 1 is 2.

Suppose we assign a weight w_{ij} on each edge $(i, j) \in E$ of graph $G = (N, E)$. This weight w_{ij} could be a cost that we want to minimize or some benefit that we want to maximize. There are several variations of matching problems. Before we list the special cases that we will consider we need some more definitions.

Definition A *perfect matching* M on a graph G is a matching on G, where each node has degree equal to one.

A perfect matching on G can only exist if n is an even number. Further, even if n is even there might not be enough edges to produce a perfect matching (assuming that G is connected). Consider the example of the tree shown in Figure 2. There are 4 nodes and 3 edges but we cannot pick two edges to produce a matching without violating the degree constraints.

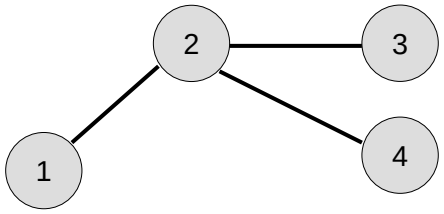


Fig. 2. There is no perfect matching on this graph.

Definition A graph $G = (N, E)$ is *bipartite* if we can partition the nodes in N into two sets N_1, N_2 such that each edge $(i, j) \in E$ belongs to the cut $[N_1, N_2]$.

This means that each edge $(i, j) \in E$ joins a node in N_1 to a node in N_2. Thus, there are no edges that join nodes that are both in N_1 or both in N_2. An example of a bipartite graph is

shown in Figure 3. Bipartite graphs often model situations where we have two sets of different objects, such as a set of machines and a set of jobs, and we would like to assign machines to jobs. Then we have two types of nodes: nodes that denote machines (say N_1) and nodes that denote jobs (say N_2). An edge denotes a possible assignment. Thus, since we cannot assign machines to machines or jobs to jobs, there cannot be an edge connecting two nodes in N_1 or two nodes in N_2.

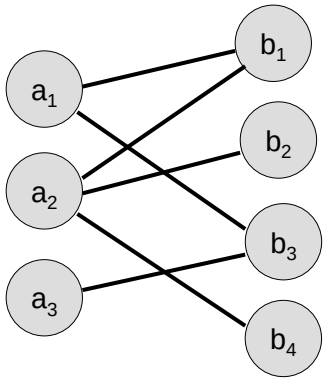


Fig. 3. An example of a bipartite graph: $N_1 = \{a_1, a_2, a_3\}$ and $N_2 = \{b_1, b_2, b_3, b_4\}$. Note that all the edges connect a node in N_1 to a node in N_2.

Definition A *bipartite matching* is a matching in a bipartite graph.

We consider the following special cases of matching problems:

1) *Bipartite Cardinality Matching Problem*: Given a bipartite graph $G = (N_1 \cup N_2, E)$, find a matching M of G with maximum cardinality. Here we can set $w_{ij} = 1$, and find a matching that maximizes the total weight. This problem is often called *bipartite matching problem*.
2) *Bipartite Weighted Matching Problem*: Given a bipartite graph $G = (N_1 \cup N_2, E)$ with $|N_1| = |N_2|$ and edge weights w_{ij}, find a perfect matching M with minimum total weight. This problem is the classic *assignment problem*. We assume that there exists a perfect matching in G.

3) *Stable Matching Problem*: This problem is often called *stable marriage problem* because of the following application: Suppose we want to marry k men to k women. Each person ranks those of the opposite sex according to their preference. We would like to find a perfect matching, where there does not exist a man and a woman that are not married to each other but prefer each other to their current spouses. Such a matching is called a stable matching. It is stable in the sense that no man-woman have an incentive to leave their current partners and run off with each other. This is a feasibility problem since we are not trying to optimize an objective but just identify a stable perfect matching.
4) *Nonbipartite Cardinality Matching Problem* Given a general graph $G = (N, E)$, find a maximum cardinality matching.

Note that the bipartite cardinality matching problem is a special case of the bipartite weighted matching problem when we are maximizing weights. However, due to its special structure, we consider it separately because we can find simpler algorithms to solve it.

Further, in nonbipartite graphs finding matchings becomes a difficult problem and thus we only consider the simpler case of finding a maximum cardinality matching as mentioned above. We describe an algorithm for finding matchings in the bipartite case and we show how this algorithm fails for nonbipartite graphs. Modifications of this algorithm to find matchings in nonbipartite graphs exist but we do not cover them in this lecture. The interested reader can find these at the end of section 12.6 of AMO.

III. LP Formulations

In this section we formulate the general weighted matching problem for general graphs (includes both bipartite and nonbipartite). We also provide special formulations for the bipartite matching problems in sections IV and V.

A. General Weighted Matching Problem

In a general graph $G = (N, E)$, with edge weights w_{ij}, we let variable $x_{ij} = 1$, if edge (i, j) is in the matching and $x_{ij} = 0$ otherwise. The *general weighted matching problem* can be

formulated as follows:

$$\begin{aligned} &\max_{x_{ij}} \sum_{(i,j)\in E} w_{ij}x_{ij} \\ &\text{s.t.} \sum_{j:(i,j)\in E} x_{ij} \leq 1 \quad \text{for all } i \in N \\ &\quad x_{ij} \in \{0,1\} \quad \text{for all } (i,j) \in E \end{aligned} \tag{1}$$

The constraints above ensure that the degree of each node i is either zero or one, thus resulting in a matching. Further, we are maximizing with respect to positive weights $w_{ij} \geq 0$; if any of the weights were negative then the corresponding variable would be zero. If we set $w_{ij} = 1$, then the above formulation becomes a *general maximum cardinality matching problem*.

We can also constraint the problem to a perfect matching and then minimize with respect to some cost c_{ij} (substitute $w_{ij} = -c_{ij}$ in problem (1)). This is the *minimum weight perfect matching problem* and is formulated as follows:

$$\begin{aligned} &\min_{x_{ij}} \sum_{(i,j)\in E} c_{ij}x_{ij} \\ &\text{s.t.} \sum_{j:(i,j)\in E} x_{ij} = 1 \quad \text{for all } i \in N \\ &\quad x_{ij} \in \{0,1\} \quad \text{for all } (i,j) \in E \end{aligned} \tag{2}$$

In the above problem we require a perfect matching and thus the degree of each node $i \in N$ must be equal to one.

Note that both of the above formulations are integer programming formulations. For general weighted matching problems we cannot relax them to LP formulations. However, when the graph is bipartite then we can relax them to LP formulations and still get integer solutions. Thus we study bipartite matching problems separately.

IV. Bipartite Cardinality Matching Problem

Let $G = (N, E)$ be a bipartite graph with node partition $N = N_1 \cup N_2$. We would like to find a matching with the maximum cardinality. To solve this problem we will transform it into a maximum flow problem. First, we need to transform our undirected graph G into a network. For each edge $(i, j) \in E$, with $i \in N_1$ and $j \in N_2$, we define an arc (i, j). We also introduce a source node s and a sink node t. We connect s with an arc (s, i) to each node $i \in N_1$, and

connect every node $j \in N_2$ with an arc (j, t) to sink node t. We also set the capacity of each arc in the network to be 1. The resulting network is denoted by $G' = (N', A')$.

Figure 4 shows the transformed network G' for the bipartite graph G shown in Figure 3.

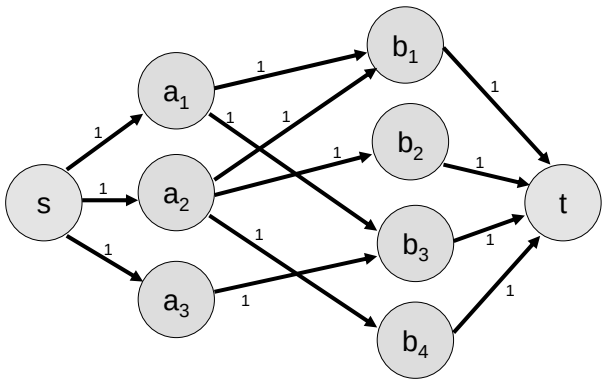


Fig. 4. The transformed network G' for the bipartite graph G shown in figure 3. All arcs have capacity 1.

Lemma 4.1: There exists a one-to-one correspondence between a matching of cardinality k in the original network G, and an integral flow of value k from source s to sink t in the transformed network G'.

Proof: Suppose that $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$ is a matching of G with cardinality k. We will construct the corresponding flow of value k. Set the flow on each arc in M to be 1: $x_{i_r, j_r} = 1$ for $r = 1, \ldots, k$. Set the flow on each arc (s, i_r), for $r = 1, \ldots, k$, to be 1 and the flow on each arc (j_r, t), or $r = 1, \ldots, k$, to be 1. Then this flow is feasible since at each intermediate node the net flow is zero. Further, this flow has value k since k units of flow leave s and k units arrive at t.

Suppose we are given an integral flow x of value k from s to t in the transformed network. Since x is assumed to be integral, we can decompose the k units of flow x into k paths from s to t of unit flow: $s - i_1 - j_1 - t$, $s - i_2 - j_2 - t$, ..., $s - i_k - j_k - t$. By construction of the transformed network we have $\{i_1, \ldots, i_k\} \in N_1$ and $\{j_1, \ldots, j_k\} \in N_2$. Since arcs (s, i) and (j, t) have unit capacity, the k paths cannot overlap in the intermediate nodes and thus no node in N_1 (or N_2) appears in more than one of these paths. Thus, the edges/arcs $\{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$

define a matching. ■

We have shown that each integral flow corresponds to a matching. Further, since the capacities of the transformed network are integral (they are all 1), then by the integrality theorem of maximum flows, the maximum flow x^* is be integral and thus it corresponds to the maximum cardinality matching. This establishes the following theorem:

Theorem 4.2: In a bipartite graph G, the maximum cardinality matching corresponds to the maximum flow in the transformed network G'.

We have shown in the proof of lemma 4.1 how to convert a flow into a matching.

Thus, we can solve the maximum flow problem in the transformed network G' using the augmenting path algorithm (or any other algorithm for finding a maximum flow) and then convert this flow into a matching.

The formulation of the *Bipartite Cardinality Matching Problem* can be written as follows:

$$\begin{aligned} \max_{x_{ij}} \quad & \sum_{(i,j)\in E} x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in E} x_{ij} \leq 1 \quad \text{for all } i \in N_1 \\ & \sum_{j:(i,j)\in E} x_{ij} \leq 1 \quad \text{for all } i \in N_2 \\ & x_{ij} \geq 0 \quad \text{for all } (i,j) \in E \end{aligned} \tag{3}$$

Note that the above formulation does not impose a condition that the variables x_{ij} are binary. Since the bipartite cardinality matching problem is equivalent to the maximum flow problem for a network with integral capacities, we know that there exist integer optimal solutions. In fact we know that the extreme points of the feasible region are integer points. Since we solve the LP using the simplex method, we know that we will always give as an optimal solution that is an extreme point (even if there are multiple optima that are not extreme points). Thus, for these type of problems we can relax the integrality condition on variables x_{ij}.

V. Bipartite Weighted Matching Problem - Assignment Problem

For a bipartite graph $G = (N, E)$ with node partition $N = N_1 \cup N_2$ and $|N_1| = |N_2|$, we would like to identify a perfect matching with minimum cost. For brevity we shall call this the assignment problem. The formulation of the assignment problem can be stated as follows (see

section III):

$$\begin{aligned} \min_{x_{ij}} \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in A} x_{ij} = 1 \quad \text{for all } i \in N_1 \\ & \sum_{k:(k,i)\in A} x_{ki} = 1 \quad \text{for all } i \in N_2 \\ & x_{ij} \in \{0,1\} \quad \text{for all } (i,j) \in A \end{aligned} \tag{4}$$

We will show that the assignment problem is a minimum cost flow problem. In order to do that we first convert the graph $G = (N, E)$ to a directed network $G = (N, A)$, where each edge $(i, j) \in E$, with $i \in N_1$ and $j \in N_2$, is converted to an arc $(i, j) \in A$. Let the capacity $u_{ij} = 1$ for all arcs $(i, j) \in A$. Further, suppose that each node in $i \in N_1$ has a supply $b(i) = 1$ and each node in $j \in N_2$ has a demand of $b(i) = -1$. We want to satisfy all supplies and demands subject to capacity constraints at minimum cost. Figure 5(a) shows a bipartite graph with $|N_1| = |N_2| = 3$; Figure 5(b) shows the conversion of this graph into a network with unit capacities and unit supplies/demands.

(a) (b)

Fig. 5. (a) Find a minimum cost perfect matching in a bipartite graph with $|N_1| = |N_2| = 3$; (b) Find a minimum cost flow that satisfies demand/supply and capacities.

Why does the minimum cost flow on this network correspond to the minimum cost perfect matching? Consider a feasible integral flow x. The demand of 1 unit for node $j \in N_2$ is satisfied by the supply of a single node $i \in N_1$ via arc (i, j), since the flow is integral. Thus, each integral

flow will comprise of $|N_1| = |N_2|$ arcs from N_1 to N_2. These arcs cannot be adjacent to the same node since each node has only 1 unit of supply or 1 unit of demand. Since, these flow arcs are not adjacent to the same nodes and their number is $|N_1| = |N_2|$, they form a perfect matching. Since the minimum cost flow problem that we created has integral capacities and demand/supplies, it has optimal integer solutions, and this is because the extreme points of the feasible region are integral. Thus, the optimal flow of the min cost flow problem corresponds to the minimum cost perfect matching.

The other direction is quite easy. Given a perfect matching the corresponding flow of setting $x_{ij} = 1$ on all the matching arcs will give us a feasible flow.

Now, cosider the formulation of the minimum cost flow problem defined as above on a bipartite graph with $|N_1| = |N_2|$:

$$\begin{aligned} \min_{x_{ij}} \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in A} x_{ij} = 1 \text{ for all } i \in N_1 \\ & -\sum_{k:(k,i)\in A} x_{ki} = -1 \text{ for all } i \in N_2 \\ & 0 \le x_{ij} \le 1 \text{ for all } (i,j) \in A \end{aligned} \tag{5}$$

As we can see the formulation (5) is identical to the assignment problem formulation (4). The only difference is that in the assignment problem we constrained the variables to be binary. We can relax this condition and let $x_{ij} \ge 0$ ($x_{ij} \le 1$ is taken care of by the conservation of flow constraints) in the assignment problem formulation (4). The LP relaxation of (4) will still give us integer solutions.

Thus, we can use network simplex to solve the assignment problem or any other algorithm that solves minimum cost flow problems.

VI. Stable Matching Problem

This problem is also called the stable marriage problem. Suppose we would like to marry n men with n women. Each person ranks all the people of the opposite sex according to their preferences.

Let N_1 be the set of men and N_2 be the set of women. Let M be a perfect matching. An edge $(i,j) \in M$ means that man i marries woman j.

Definition We say that M is an *unstable* matching if there exists a man-woman pair $i - j$ that are not married ($(i, j) \notin M$), but prefer each other to their current spouses.

Definition We say that M is a *stable* matching if it is not unstable.

Our aim is to find a stable perfect matching.

To find a stable perfect matching we use the *propose and reject algorithm*, which is an iterative greedy algorithm. We assume that we are given two matrices of rankings: A is the matrix that gives each man's ranking of the women, and B gives each woman's ranking of the men. The higher the ranking the higher the preference. Thus, A_{ij} = the ranking of woman j by man i, and B_{ji} = the ranking of man i by woman j.

We assume that men propose and women accept or reject. The algorithm can also be applied the other way round.

The algorithm starts as follows: each man proposes to his favorite woman (the one that he ranked the highest). Each woman receives a set of proposals and rejects them all except the one that she ranks the highest. She is then temporarily assigned with this man and we say they are engaged. Some women will get no proposals and remain unassigned (single). Some men will be rejected and they remain unassigned (single). The algorithm keeps a $LIST$ of unassigned men and for each unassigned man it keeps a variable *next-woman*, which is the index of the woman with the highest ranking that has not yet rejected him. Initially, $LIST = N_1$ and the *next-woman* for each man is the woman with the highest ranking. When a woman rejects a man then we assign *next-woman* to be the next woman with the highest ranking.

At the next iteration, the unassigned men in $LIST$ propose to their *next-woman*. If a woman receives a single proposal for the first time then she accepts it. If a woman is unassigned and she receives several proposals for the first time then she accepts the one that she ranks highest. If she is already engaged then she picks the man that has the highest ranking amongst her fiance and all the other men that proposed. When a man proposes he leaves the $LIST$ and if he gets rejected then he returns to the $LIST$. The algorithm terminates when the $LIST$ is empty.

A description of the algorithm is shown in Algorithm 1.

We have the following result:

Lemma 6.1: The propose-and-reject algorithm produces a stable matching.

Proof: Suppose John prefers Mary to his current partner. We will show that Mary prefers

Algorithm 1 Propose-and-Reject Algorithm

begin

Let $LIST = N_1$ be the list of unassigned (single) men.

For each $i \in LIST$, let *next-woman*(i) be the index of the woman ranked highest by man i.

Assume that each woman $j \in N_2$ is unassigned (single).

while $LIST$ is not empty

All men in $LIST$ propose to their *next-woman*. Remove all men from $LIST$.

Each single woman j accepts the proposal of man i that has the highest ranking.

Each already engaged woman j accepts the proposal of man i that has the highest ranking amongst her fiance and all other proposals. If i is not her fiance, the fiance is returned to $LIST$.

All women j that accepted proposals from men i are now engaged.

Each man that made a proposal and got rejected returns to $LIST$.

For each man in $LIST$, we let *next-woman* be the highest ranked woman that he has not yet proposed.

end;

end;

her current partner to John and thus the matching is stable. Since men propose to women in the order of their rankings, John must have proposed to Mary at an earlier iteration and got rejected because Mary preferred another man, say man i, to John. Since no woman switches to a partner she prefers less, Mary prefers her current partner more than i (or equal if i is her current partner) and she prefers i more than John. So the matching is stable. ■

There are several stable matchings and the propose-and-reject algorithm just gives us one of them.

Definition We refer to a pair (i, j), $i \in N_1$, $j \in N_2$, as *stable partners* is there exists a stable matching that assigns them together.

The propose and reject algorithm produces a stable matching such that each man is assigned

to his best possible stable partner. We call such a matching *man-optimal* matching. Before we prove this we show the following:

Lemma 6.2: In the propose-and-reject algorithm, where men propose and women accept/reject, a woman never rejects a stable partner.

Proof: Let M^* be the stable matching produced by the propose-and-reject algorithm. Suppose that women reject stable partners in the propose-and-reject algorithm. Suppose that the first time that a woman rejects a stable partner is when Mary rejects David to be with John. Thus,

(1) Mary prefers John to David.

Let M be the stable matching that produces the stable pair (David, Mary). Since we assumed that the rejection of David by Mary was the first time a woman rejected a stable partner in the propose-and-reject algorithm, we can conclude that John was not earlier rejected by any of his stable partners and thus Mary is the highest ranking stable partner for John (since men propose in the same order as their rankings). Thus,

(2) John prefers Mary to any other stable partner.

Suppose that in M, John was assigned to Kate, which means (John, Kate) is a stable pair. From observation (2) we can conclude that:

(3) John prefers Mary to Kate.

Since M has pairs (David, Mary) and (John, Kate), by observations (1) and (3) we conclude that M is not a stable matching. Thus, we have reached a contradiction and our lemma holds. ■

Since men propose to women in the order or their preferences and since women never reject a stable partner, the propose-and-reject algorithm marries every man to his best stable partner. We have the following theorem:

Theorem 6.3: The propose-and-reject algorithm, produces a man-optimal matching.

It is obvious that the propose and reject algorithm produces the best stable matching for men. It can be shown that in fact this is at the expense of women: in a man-optimal matching each woman obtains the worst possible stable partner.

Stable matchings have many applications. One very important application is the assignment of medical students to hospitals, where a centralized system exists that makes assignments. In this situation, it is desirable that any assignment is stable.

VII. The Matching Algorithm for Maximum Cardinality Matchings

We have shown in section IV that we can covert the problem of finding a maximum cardinality matching in a bipartite graph to a maximum flow problem. In this section we provide an alternative algorithm for finding maximum cardinality matchings in bipartite graphs. This algorithm is useful in practice because it can extend to the nonbipartite case. We do not cover this extension in this lecture but we highlight why it fails in the general case.

A. Preliminaries

Consider a matching M on a general graph $G = (N, E)$. We call edges (i, j) in G as *matched edges* if they are in M and *unmatched edges* otherwise. Further, we call nodes i in G that are incident to matched edges as *matched nodes* and *unmatched nodes* otherwise.

Definition A path P in G is called an *alternating path* with respect to matching M, if every consecutive pair of edges in P contains one matched and one unmatched edge. We call it an *even alternating path* if it contains an even number of edges and an *odd alternating path* if it contains an odd number of edges. An *alternating cycle* is an alternating path that begins and ends at the same node.

For the matching shown in Figure 6(a) path $1 - 3 - 5 - 4$ is an odd alternating path, path $1 - 3 - 5 - 4 - 2$ is an even alternating path and $2 - 4 - 5 - 3 - 2$ is an alternating cycle.

Definition We call an odd alternating path whose first and last nodes are unmatched, an *augmenting path*.

The reason that we are interested in augmenting paths is that we can switch the edges on an augmenting path from matched to unmatched and vice versa and obtain a matching with cardinality $|M| + 1$. Thus, by finding augmenting paths we can increase the cardinality of the matching. Figure 6(b) shows the resulting matching after switching the edges on the augmenting path $1 - 2 - 4 - 3 - 5 - 6$ of Figure 6(a).

B. Augmenting Path Theorem

In this section we show an important theorem for matchings that gives rise to an algorithm using augmenting paths. We first introduce some new concepts that build on ideas from the

(a) (b)

Fig. 6. (a) Path $1-3-5-4$ is an odd alternating path and path $1-3-5-4-2$ is an even alternating path; $2-4-5-3-2$ is an alternating cycle; path $1-2-4-3-5-6$ is an augmenting path; (b) Switching matched with unmatched edges in augmenting path $1-2-4-3-5-6$ results in the matching shown with cardinality 3, one more than the cardinality 2 of the original matching.

previous section.

Definition Let A and B be two sets, we define the *symmetric difference* denoted by $A \oplus B$ to be the set of elements that are members of one of A, B but not both.

If M is a matching and P is an augmenting path with respect to M, then $M \oplus P$ is the resulting matching if we switch the edges of P (from matched to unmatched and vice versa). All the edges of M not in P will remain matched in $M \oplus P$, the common edges of M and P will not be matched in $M \oplus P$, the edges in P but not in M will be matched in $M \oplus P$. Figure 6(b) shows matching $M \oplus P$ that results from matching M shown in Figure 6(a) and augmenting path $1-2-4-3-5-6$. Whenever, P is an augmenting path $M \oplus P$ will increase the cardinality of the matching by 1. Note further that in $M \oplus P$, all the nodes in M remain matched and two more nodes namely the first and the last nodes of P are matched.

Suppose we have two matchings M and M^* of G let us see what happens when we take the symmetric difference $M \oplus M^*$. Consider the graph $G^* = (N, M \oplus M^*)$. If node p in G is unmatched in both M and M^*, then in G^* it will be a disconnected node as shown in Figure 7(a). If node p is matched in both M and M^* by the same edge, then again it will be a disconnected node as shown in Figure 7(a). If it is matched by both M and M^* from different edges then it

will have two edges incident to it in G^* as shown in Figure 7(b). If it is matched by only one of M and M^*, then it will have only that edge incident to it in G^* as shown in Figures 7(c) and (d).

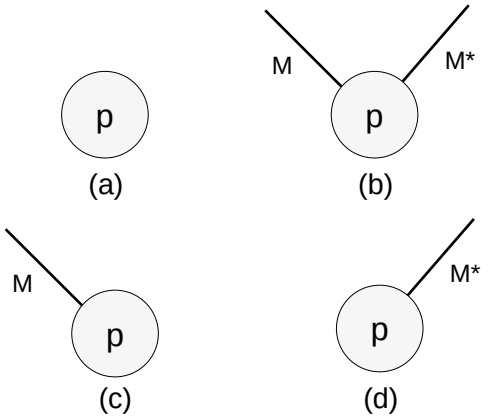


Fig. 7. The Figure shows all the possible cases for the edges that are incident to node p in G^*: (a) node p was unmatched by both M, M^* or matched by both via the same edge; (b) node p was matched by both M, M^* via different edges; (c) node p was matched by M and unmatched by M^*; (d) node p was unmatched by M and matched by M^*.

From Figure 7 we can see that the degree of nodes in G^* can only be 0, 1 or 2. Further, each node with degree 2 will have an edge from M and an edge from M^* and thus any path or cycle will alternate in edges of M and M^*. Therefore, the graph G^* will be composed of disconnected components that are of the six types shown in Figure 8. Components (b) and (e) are even paths starting with an edge from M and M^* respectively. Components (c) and (d) are odd paths starting and ending with edges from M and M^* respectively. Component (f) is an even cycle.

We will use these components to prove the following important theorem:

Theorem 7.1: Augmenting Path Theorem: The matching M^* is a maximum cardinality matching of G if and only if the graph G contains no augmenting path with respect to matching M^*.

Proof: If M^* is a maximum cardinality matching then we cannot have an augmenting path because if we did we could increase the cardinality of M^* and thus violate the optimality assumption.

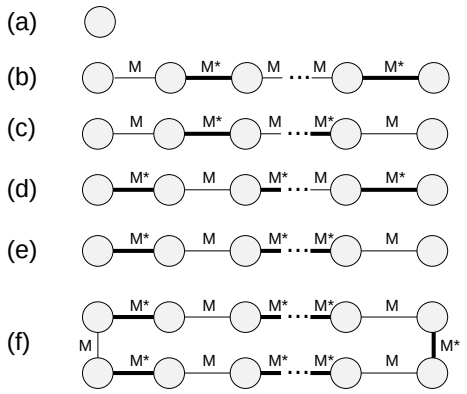


Fig. 8. All possible components of graph $G^* = (N, M \oplus M^*)$.

Suppose G contains no augmenting path with respect to M. Let M^* be a maximum cardinality matching. Consider the graph $G^* = (N, M \oplus M^*)$. G^* cannot have components of type (c), since these imply that M^* had an odd path with the end nodes unmatched, contradicting its optimality. Also, G^* cannot have components of type (d), since these imply that M had an odd path with the end nodes unmatched, contradicting the assumption that M has no augmenting path. Thus we can only have components of type (a), (b), (e) and (f). All of these components contain the same number of edges from M and from M^*. Since, G^* contains the edges that are not common to M and M^*, the edges of M and M^* that are not in G^* are common to both. Thus, $|M| = |M^*|$, and M is a maximal cardinality matching. ■

There is another version of the augmenting path theorem state below:

Theorem 7.2: Alternative version of Augmenting Path Theorem: If node p is unmatched in M and M contains no augmenting path that starts at node p, then there exists a maximum cardinality matching with node p unmatched

We do not proof this version of the theorem. The proof can be found in AMO (page 478).

C. The Matching Algorithm

In this section we describe the matching algorithm. The material we have covered so far in section VII applies to general graphs. The matching algorithm that we describe in this section applies to general graphs if there is a procedure in place for finding augmenting paths. It turns

out that this is quite straightforward for bipartite graphs but more difficult in the nonbipartite case.

The algorithm is quite simple and it is based on the augmenting path theorem. We start with a feasible matching M that could be the null matching (i.e. $M = \emptyset$). For each unmatched node p in G, we identify an augmenting path starting at node p. If such a path P exists, we replace M with $M \oplus P$ which increases the cardinality of the matching by 1. If there exists no augmenting path starting at node p, then delete node p and all the arcs incident to it from G and continue. The description of the matching algorithm is shown in Algorithm 2.

Algorithm 2 The Matching Algorithm

begin

Let M be an initial matching.

for each unmatched node p do

if there exists an augmenting path P starting at node p then

Replace M with $M \oplus P$.

else

Remove p and all the edges incident to it from graph G.

end;

end;

M is a maximal cardinality matching of G.

end;

The algorithm will result in a maximum cardinality matching in a finite number of iterations since it will check all unmatched nodes whose number reduces at every iteration. When there is no augmenting path, we know that the matching is optimal.

Removing a node p that has no augmenting path starting at p will not reduce the cardinality of the matching. By the alternative version of the augmenting path theorem (theorem 7.2), we know that there exists a maximum cardinality matching with node p unmatched. Even if we left node p in the graph, it will never get matched in any future iterations. This is because the only nodes that become matched are the nodes at the ends of an augmenting path. Further, node p

and its incident edges will never be part of any other augmenting path. Thus we have no use for node p and its incident edges and we remove them.

D. Finding an Augmenting Path

We find an augmenting path starting at unmatched node p by using a search algorithm. We define a node i in the graph as *reachable* from node p if there is an alternating path from node p to node i. We use the search algorithm to find reachable nodes and when a reachable node is unmatched then we have found an augmenting path. The search algorithm will create a search tree that we call the *alternating tree*. We label the nodes that are reachable from p with two types of labels: odd and even which we denote with O and E. These labels identify whether the number of edges on the alternating path from node p to reachable node i is even or odd. If the alternating path from node p to node i has odd number of edges then i will be labeled O.

We start by labeling node p with E. The search algorithm maintains a $LIST$ of nodes to examine. We perform a breadth first search (first-in first-out). There are two cases:

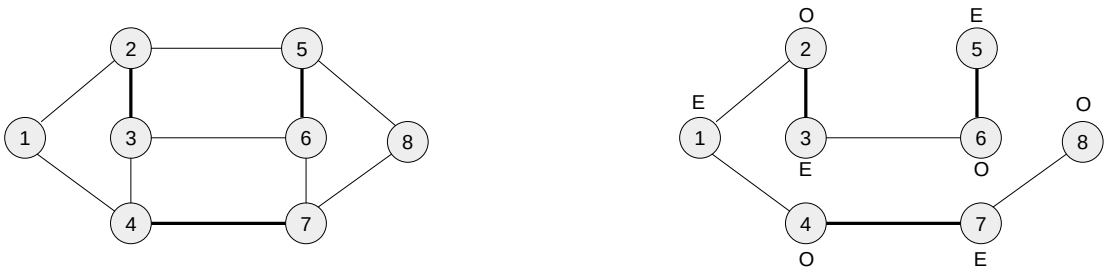
1) When examining a node i labeled E, the algorithm scans the adjacent edges (i, j) and assigns the label O to each j that has not yet been labeled. If node j is unmatched, then we have found an augmenting path and we stop; else we add j to the $LIST$. We assume the nodes are labeled and then added to the $LIST$ in the order that gives preference to the nodes with lower indices.
2) When examining a node i labeled O, then the algorithm selects the unique matched edge (i, j). If j is not already labeled, then it labels it with E and adds it to $LIST$.

The algorithm terminates either when $LIST$ is empty, which means it could not find an augmenting path, or when it assigns an O label to an unmatched node, which means it found an augmenting path.

Consider the example given in Figure 9. Figure 9(a) shows a graph with the current matching shown in bold lines. We would like to find an augmenting path starting at unmatched node 1. We use a breadth first search algorithm. We label node 1 with E. We scan all the edges out of 1 in the order $(1, 2)$, $(1, 4)$ giving priority to the one with the lowest index. We label node 2 with O and since it is a matched node we continue and we add it to $LIST$. Similarly, we label node 4 with O and add it to $LIST$. We have $LIST = \{2, 4\}$. We now examine node 2 which has an O label: we only care about the matched edge $(2, 3)$. Since, 3 is not already labeled we

label it with E and add it to $LIST$. We have $LIST = \{4, 3\}$. We continue by examining node 4 and so on. The nodes are examined in the following order: $1-2-4-3-7-6-8-5$. The complete alternating tree with the labels is shown in Figure 9(b).

However, we could have stopped before completing the tree since when we examined node 7 which is labeled E, we found an unmatched node 8 labeled O. At that point we stop the search algorithm because we found augmenting path $1-4-7-8$.



(a) (b)

Fig. 9. (a) Graph with matching shown in bold lines; (b) The complete alternating tree rooted at node 1 with E and O labels on each node.

How do we know that the search algorithm will find an augmenting path when there exists one? In fact we do not know that. When the graph is bipartite then we can always find the augmenting path using the search algorithm. In the next section we discuss these issues.

E. Why does the Search for an Augmenting Path fail for Nonbipartite Graphs

Consider the graph shown in Figure 10. We are looking for an augmenting path starting at node 1. There exists an augmenting path, namely $1-2-3-4-5-6$. If the search algorithm, reaches node 5 via alternating path $1-2-3-4-5$, then it will assign label E to node 5 and when checking node 5 it identifies node 6 as an O node that is unmatched. In this case, it will identify the augmenting path. However, if the search algorithm reaches node 5 via alternating path $1-2-3-7-8-5$, then node 5 will have the label O. In this case, when checking node 5 it will only scan the matched edge $(5, 4)$ and miss the augmenting path.

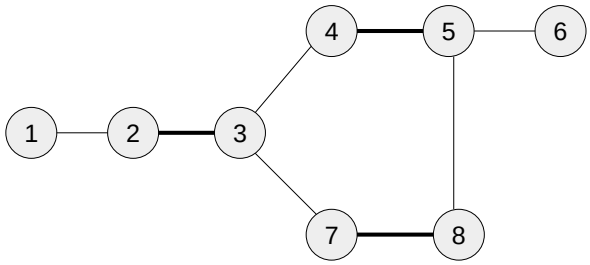


Fig. 10. Path $1-2-3-4-5$ gives 5 an E label and the algorithm discovers the augmenting path; path $1-2-3-7-8-5$ gives 5 an O label and it misses node 6, and does not discover the augmenting path.

The reason for the failure to find the augmenting path is that nodes can have different labels if they are reached from different paths. If each node possessed a unique label irrespective of the alternating path that reached it, then the algorithm would always find the augmenting path. Let us show this: suppose there exists an augmenting path $p-i_1-j_1-i_2-j_2-\ldots-i_k-j_k-q$ from node p to node q with respect to a matching M. If we examined nodes $p, i_1, j_1, i_2, j_2, \ldots, q$ in this order, then we would assign even labels to $p, j_1, j_2 \ldots, j_k$ and odd labels to $i_1, i_2, \ldots, i_k, q$. Given our assumption, any other alternating path should assign the same labels. Thus, the search algorithm will assign an odd label to node q and find an augmenting path (which might be a different one).

Thus, it is important to be able to label the nodes with the same label independent of the path. The problem arises when we have odd cycles as in the example of Figure 10. If we try to reach a node from different sides of the odd cycle then we will end up with different labels. Graphs that have no odd cycles do not have this problem. It turns out that such graphs are in fact bipartite graphs.

Lemma 7.3: A graph is bipartite if and only if it has no odd cycles.

Proof: Suppose graph G is bipartite with $N = N_1 \cup N_2$. Let $i \in N_1$ be a node in a cycle C. Since the graph is bipartite, any path starting at i will alternate between sets N_2 and N_1. The cycle C starts at i and terminates at i and thus it will go back and forth an even number

of times. Thus cycle C has an even number of edges.

Suppose a graph G has no odd cycles. Let T be a spanning tree of G. Pick a node i as the root node. We label each node j with its depth index which is the number of edges on the unique path from node i. We label node i with 0. Let N_1 be the set of nodes with even indices (including i) and let N_2 be the set of nodes with odd indices. Let $p, q \in N_1$, then the unique path on T connecting p and q will have indices that alternate between $even - odd - even - \ldots - even$, which contains an even number of edges. Non-tree edge (p, q) cannot exist because if it did it would create an odd cycle with the unique path from p to q. Similarly, for $p, q \in N_2$, we cannot have non-tree edge (p, q). Thus, no two nodes within N_1 are adjacent and no two nodes within N_2 are adjacent, and thus G is bipartite. ■

Thus when the graph is bipartite we can always find an augmenting path using the search algorithm. For the nonbipartite case, which contains odd cycles, some modifications in the search algorithm for an augmenting path can be made to ensure that we get an augmenting path. We do not cover these in this lecture but the interested reader can check the last part of section 12.6 of AMO.

VIII. Applications

Several other applications of matchings can be found in section 12.2 of AMO.

IX. Exercises

1) For the figure in 12.4 find a matching M of cardinality 1. Start from this matching and use the augmenting path algorithm for max flow to find a maximum cardinality matching in the graph.
2) Solve 12.33 of AMO. Justify why this graph is bipartite. Use the Bipartite Matching algorithm starting from unmatched node 2 to find a maximum cardinality matching.
3) Use the propose-and-reject algorithm for the Stable Marriage problem and solve 12.25.
4) Formulate the problem 12.3 of AMO as an assignment problem. Construct a minimum cost flow network that solves this problem. Explain how you would find an initial feasible solution using matchings. Convert this solution to a spanning tree solution.
5) Show the the propose and reject algorithm produces a woman pessimal matching: this means that each woman gets her worst stable partner.