Chapter 8

Matching Problems

In this section we will study problems to do with matching things together: matching people into sports teams, matching people to tasks they are qualified to perform, matching men to women they wish to marry, and so on. For example, we will consider questions such as the following:

Example 8.1. A factory has m employees, each of whom is qualified to do some of n different jobs. Is it possible to assign each employee to a job they can do, such that no two employees are doing the same job?

Many of these problems can be described entirely in terms of set theory, but using graph theory often gives a convenient way to visualise them. We will begin with some elementary counting problems, and go on to see how these can be generalised to more complex scenarios with the aid of graphs.

8.0.5 Pairings

Definition 8.1. Let S be a set of size $2n$. A *pairing* of S is a partition of S into n subsets $X_1, X_2, \dots X_n \subseteq S$, with $|X_i| = 2$ for all $i = 1, 2, \dots, n$ and $X_i \cap X_j = \emptyset$ when $i \neq j$ (the X_i are *pairwise disjoint*).

Example 8.2. If S is the set $\{a, b, c, d, e, f, g, h\}$, then $\{\{a, d\}, \{e, h\}, \{f, b\}, \{g, c\}\}$ is a pairing of S.

Theorem 8.1. *The number of possible pairings of a set of size* $2n$ *is*

$$\frac{(2n)!}{(2!)^n n!}$$

Proof. From a listing $s_1, s_2, s_3, s_4, \dots, s_{2n-1}, s_{2n}$ of the elements of S we obtain the pairing $\{\{s_1, s_2\}, \{s_3, s_4\}, \dots, \{s_{2n-1}, s_{2n}\}\}$. Reordering the subsets of this pairing or reordering the two elements in any of the subsets of this pairing produces the same pairing. Thus different listings of S can result in the same pairing of S. We now determine the number of different listings which give the same pairing.

Firstly the n subsets in the pairing can be reordered in $n!$ ways without changing the pairing. Secondly each subset of this pairing can be reordered in $2!$ ways to give the same pairing. Since there are n subsets in the pairing, the elements within the subsets of the pairing can be reordered in $(2!)^n$ ways to give the same pairing. Since there are $(2n!)$ different ways of listing the elements of S, it follows that the number of different pairings of S is $\frac{(2n)!}{(2!)^n n!}$. $\square$

Example 8.3. A tennis club has 10 members. In how many ways can the players pair up to play each other for their Saturday morning singles match?
Solution. By Theorem 8.1, the number of ways is given by $\frac{10!}{2!^5 5!} = 945$.

This result can be generalised to sets of size $m \geq 2$; the proof is similar.

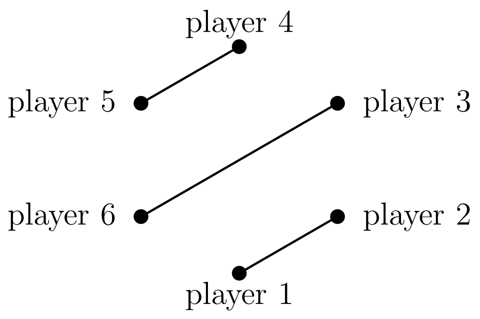
Theorem 8.2. *Let S be a finite set with mn elements. Then the number of ways of partitioning S into n pairwise disjoint sets of size m is given by*

$$\frac{(mn)!}{(m!)^n n!}$$

Example 8.4. The 10 club members decide they are sick of tennis and would rather play 5-a-side football. How many ways are there of dividing them into two teams of size 5?
Solution. The answer is given by $\frac{(10)!}{(5!)^2 2!} = 126$.

Since a pairing of a set consists of subsets of size two, we can represent a pairing in terms of a graph by letting the subsets correspond to edges.

Example 8.5. Suppose 6 players are to be paired up to play tennis matches. We can represent this scenario by a graph in which the players correspond to vertices, joining vertices by edges to indicate which players are playing together:



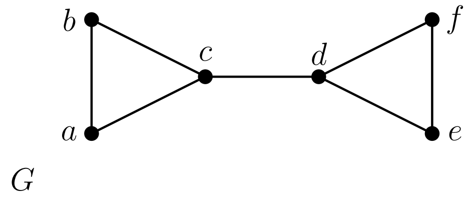
8.1 Matchings in Graphs

In many practical cases, pairing problems are more complicated than those considered in the previous section. For example, in trying to pair off $2n$ students to share rooms in college accommodation there may be several restrictions, such as the sex of the student, whether the student is a smoker, the interests of the student, which prevent certain students sharing the same room. This problem is much more difficult to solve and often the question is not how many pairings there are, but whether there exists a pairing satisfying all of the restrictions. This situation can be modelled using a simple graph with $2n$ vertices in which two vertices are joined by an edge if and only if they can be paired off. In graph theory terminology, we are asking whether the graph has a *perfect matching.*

Definition 8.2. Let G be a simple graph. A *matching* in G is a set M of edges of G such that no two distinct edges of M have an endpoint in common. That is, if e_1 and e_2 are distinct edges of M with $e_1 = u_1v_1$ and $e_2 = u_2v_2$ then $\{u_1, v_1\} \cap \{u_2, v_2\} = \emptyset$. A matching M is a *maximal matching* if no further edges can be added to M to give a larger matching; it is a *maximum matching* if it is at least as large as any other matching. A matching is a *perfect matching* if every vertex of G is the endpoint of some edge in M.

We observe that for a graph to have a perfect matching it must have an even number of vertices.

Example 8.6.



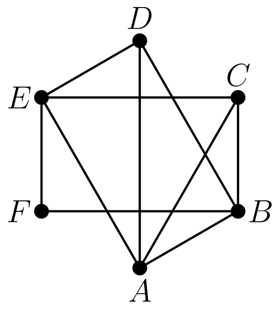
An example of a perfect matching for G is given by the set of edges $M_1 = \{ab, cd, ef\}$. (Note that every perfect matching is necessarily a maximum matching, and every maximum matching is a maximal matching.) The matching given by the set of edges $M_2 = \{bc, de\}$ only has 2 edges and hence is not a maximum matching. However, it *is* a maximal matching, since every other edge shares a vertex with one of these two edges and thus M_2 is not contained in any larger matching.

Example 8.7. A pairing of a set with $2n$ elements corresponds to a perfect matching of the complete graph on $2n$ vertices. A perfect matching of a graph G is also known as a *1-factor* of G. Reinterpreting Theorem 8.1 in graph theoretic notation tells us that the number of distinct 1-factors of the complete graph on 2n vertices is given by

$$\frac{(2n)!}{(2!)^n n!}$$

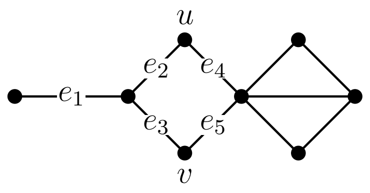
Example 8.8. Alice and Bob, their two children Charlie and Dave, and Alice's cousin Emma and her daughter Frances are going on a canoe trip. Each canoe holds two people. Each of the three children must be accompanied by an adult, Emma doesn't like Bob, and Alice and Frances both get seasick unless they sit in the front of a canoe. Is it possible to find a way of assigning people to canoes so that everyone is happy?

Solution. The following graph represents the pairs of people who can share a canoe:



This graph has a perfect matching $\{EF, AD, BC\}$. Thus, if Emma and Frances share a canoe, as do Alice and Dave, and Bob and Charlie, everyone will be happy.

Example 8.9. The following graph does not have a perfect matching.



To see this, observe that any perfect matching must contain e_1. Hence it cannot contain either of e_2 or e_3. Since u and v both have to be endpoints of some edge in a perfect matching, it follows that e_4 and e_5 would have to be in any perfect matching, which is not possible. Thus the graph has no perfect matching.

Exercise 8.10.

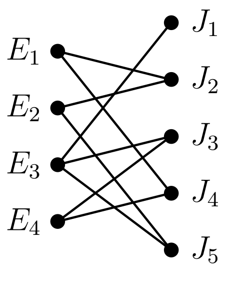
1. What is the size of a maximum matching in $K_{2,4}$? Is it a perfect matching? What about $K_{4,4}$?
2. Show that each cycle graph C_{2n} with an even number of vertices has a perfect matching. Use this to show that a Hamiltonian graph with an even number of vertices has a perfect matching. *(Hint: consider C_4, then C_6, then try and generalise your findings.)*

8.2 Complete Matchings in Bipartite Graphs

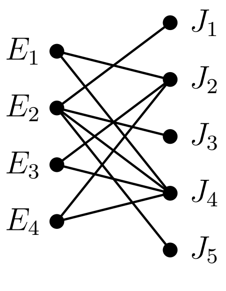
Many matching problems involve trying to pair elements of one set with elements of a another set, and these can be described conveniently in terms of bipartite graphs.

Example 8.11. Consider the problem of assigning jobs mentioned in Example 8.1. We can model this problem using a bipartite graph with m vertices that represent the employees and n vertices that represent the jobs. We add an edge from vertex E_i to vertex J_j when employee i is qualified to do job j. Then we can assign an employee to each job such that all the employees are doing different jobs if and only if the resulting bipartite graph has a matching of size m.

Consider the following graphs:



G_1



G_2

G_1 has a matching of size 4, for example $\{E_1J_2, E_2J_5, E_3J_1, E_4J_3\}$, but G_2 does not. To see this, consider the vertices E_1, E_3 and E_4. Between them they are joined to only two vertices, namely J_2 and J_4. Thus in any matching at least one of E_1, E_3 or E_4 will not be the endpoint of an edge and so G_2 has no matching of size 4.

Definition 8.3. Let $G = (V_1, V_2, E)$ be a bipartite graph and let V_1 and V_2 be the two sets of the partition of the vertices, with $|V_1| = m$, and $|V_2| = n$, where $m \leq n$. Then a *complete matching* for for G is a matching that contains m edges.

A complete matching of a bipartite graph is also a maximum matching, but it is only a perfect matching in the case $m = n$. Using this terminology, we can say that the problem of assigning jobs as discussed in Example 8.11 can be solved precisely when the corresponding bipartite graph has a complete matching.

8.2.1 Hall's Theorem

How can we tell when a bipartite graph has a complete matching? The answer to this is given by a theorem often known as *Hall's Marriage Theorem* because it provides a solution to the following problem:

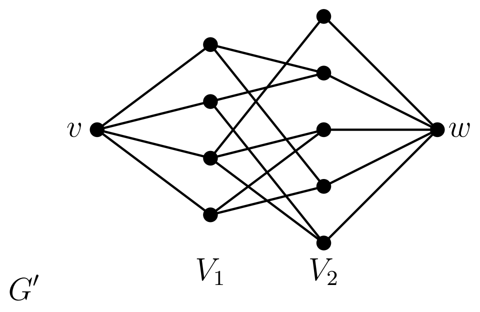
> Given a (finite) set of men, each of whom know a (finite) number of women, is it possible for each man to marry a woman that he knows?

We can represent this scenario by a bipartite graph by letting the vertices of V_1 correspond to the men, the vertices of V_2 be the women, with an edge joining a man's vertex to a woman's vertex if the man knows the woman. This question is then asking whether the resulting graph has a complete matching; Hall's theorem gives us a precise answer:

Theorem 8.3 (Hall's Marriage Theorem)**.** *Let $G = (V_1, V_2, E)$ be a bipartite graph. Then G has a complete matching if and only if for each subset $S \subseteq V_1$, the set $N(S) \subseteq V_2$ of vertices of V_2 that are adjacent to vertices in S satisfies $|N(S)| \geq |S|$. (We refer to this as the* Hall condition *on G.)*

In Example 8.11 we saw that graph G_2 failed to have a complete matching precisely because the Hall condition was not satisfied. Similar reasoning shows that the Hall condition is a necessary condition for a bipartite graph to have a complete matching. To prove Hall's theorem, we also need to show that it is a sufficient condition; this can be done with the aid of the vertex form of Menger's theorem.

Proof. Let $G = (V_1, V_2, E)$ be a bipartite graph in which the Hall condition is satisfied, with $|V_2| \geq |V_1| = m$. Consider the graph G' obtained by adding a vertex v to G that is adjacent to all the vertices in V_1, as well as a vertex w that is adjacent to all the vertices in V_2.



A matching in G gives rise to a set of vertex-disjoint vw-paths in G' and *vice versa*. Thus, to show the existence of a matching of size m, it suffices to show the existence of a set of m vertex-disjoint vw-paths in G'. Menger's theorem tells us that such a set exists precisely when the smallest vw-separating set has size m.

Suppose S is a vw-separating set, and let $S \cap V_1 = A$ and $S \cap V_2 = B$. Then there are no edges of G joining vertices of $V_1 \setminus A$ to $V_2 \setminus B$, so $N(V_1 \setminus A) \subseteq B$. We have $|N(V_1 \setminus A)| \geq |V_1 \setminus A|$, and hence $|B| \geq |V_1 \setminus A|$. But this implies that

$$\begin{aligned}|S| &= |A| + |B| \\ &\geq |A| + |V_1 \setminus A| \\ &= m.\end{aligned}$$

Thus any vw-separating set of G' has size at least m, and the result follows. □

In general a bipartite graph will not have a complete matching. In this case we look for the next best thing, namely a maximum matching.

Definition 8.4. Let $G = (V_1, V_2, E)$ be a bipartite graph. The *deficiency* d of G is given by

$$d = \max_{S \subseteq V_1} \{|S| - |N(S)|\}.$$

Theorem 8.4. *Let $G = (V_1, V_2, E)$ be a bipartite graph with deficiency d. Then the size of a maximum matching M is given by $|M| = |V_1| - d$.*

This result can be proved by the technique that was used to prove Theorem 8.3.

Exercise 8.12. Alice, Bob, Charlie and Dave are helping me to landscape my garden. I need to do some paving, build decking, remove weeds, plant trees, and dig a fishpond. Alice can dig ponds, remove weeds and build decking. Bob knows how to dig ponds, remove weeds, and plant trees. Charlie is good at paving. Dave can dig ponds and remove weeds. Can I assign a different task to each of my friends?

8.3 Stable Matchings

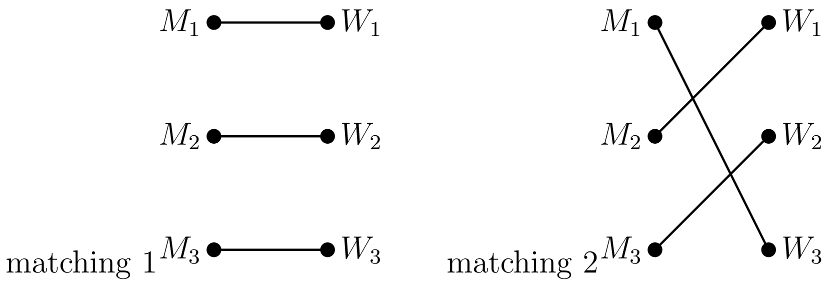
We have seen that we can always find a perfect matching in the regular complete bipartite graph $K_{n,n}$. In "marriage" terminology, if we have n men and n women, it is always possible to pair them off so that each man marries one woman. For instance, we could number the men from 1 to n, and the women from 1 to n, then simply pair the first man with the first woman and so on. However, this takes no account of whether man i and woman i actually want to get married!

Suppose each man ranks the women from the one he would most like to marry, down to the one he would least like to marry, and the women do the same for the men. Now suppose we have a perfect matching between the men and the women. If it happens that there are two couples (M_i, W_i) and (M_j, W_j) such that the M_i prefers W_j to his current wife, and W_j prefers M_i to her current husband, then there is the risk that M_i and W_j will leave their spouses and run away together! We refer to a pair of couples with this property as an *unstable pair*. If, however, there are no unstable pairs then the matching is said to be *stable*.

Example 8.13. Suppose there are three men and three women, and their lists of preferences are as follows:

$$\begin{array}{ll} M_1{:}W_1, W_2, W_3 & W_1{:}M_2, M_3, M_1 \\ M_2{:}W_1, W_2, W_3 & W_2{:}M_3, M_1, M_2 \\ M_3{:}W_2, W_3, W_1 & W_3{:}M_2, M_1, M_3 \end{array}$$

Consider the two following matchings:



Matching 1 is not stable, as M_3 ranks W_2 ahead of his current partner W_3, and in turn W_2 prefers M_3 to her current partner M_2 so $\{(M_3, W_3), (M_2, W_2)\}$ is an unstable pair.

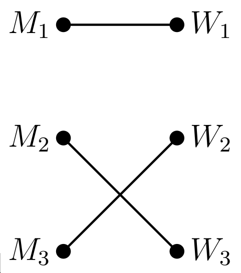
However, matching 2 *is* stable: M_2 and W_1 as well as M_3 and W_2 are all married to their favourite partners, and hence none of them are interest in swapping partners with each other, or with M_1 or W_3. (Poor M_1 is married to his least favourite woman, but none of the other women rank him highly enough to want to swap and become his partner.)

For any lists of preferences for n men and n women there exists a stable matching. In 1962, Gale and Shapley gave an algorithm to find a stable matching given lists of preferences; we will explore this algorithm shortly. First, however, we note that for given lists, there often exist more than one possible stable matching.

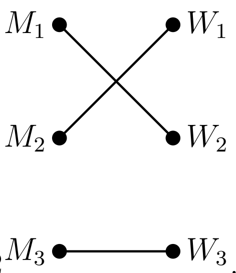
Example 8.14. Given the lists of preferences

$$\begin{array}{ll} M_1{:}W_1, W_2, W_3 & W_1{:}M_2, M_3, M_1 \\ M_2{:}W_3, W_1, W_2 & W_2{:}M_1, M_3, M_2 \\ M_3{:}W_2, W_3, W_1 & W_3{:}M_3, M_2, M_1, \end{array}$$

two possible matchings are



matching 1



matching 2.

Both these matchings are stable: in the first case, each man is married to his highest ranked woman, and so will not be prepared to change partners. In the second case, each woman is married to her favourite man, and thus will not be willing to swap.

Theorem 8.5 (The Gale-Shapley Algorithm). *Given n men and n women who have ranked each other in order of preference, a stable matching can be found using the following algorithm, which proceeds in a number of rounds. During the process, men become provisionally "engaged" to women, although these engagements may later be broken off.*

1. *Each round begins with each man who is not currently engaged "proposing" to the highest woman on his list to whom he has not already proposed.*

2. *Once the men have proposed, any woman who has received one or more proposals becomes engaged to the highest man on her list who has proposed to her, and rejects the other men who have proposed to her. (If she is already engaged, she remains engaged if she ranks her "fiancé" higher than any of the men who have proposed. Otherwise, she breaks off the previous engagement, and becomes engaged to the highest ranking man who has proposed.)*

3. *The process repeats until all the men are engaged.*

Example 8.15. Consider the following lists of preferences.

$$\begin{array}{ll}
M_1{:}W_2, W_3, W_1, W_5, W_4 & W_1{:}M_3, M_1, M_2, M_5, M_4 \\
M_2{:}W_3, W_1, W_2, W_5, W_4 & W_2{:}M_2, M_4, M_5, M_3, M_1 \\
M_3{:}W_2, W_5, W_3, W_1, W_4 & W_3{:}M_1, M_2, M_5, M_4, M_3 \\
M_4{:}W_4, W_1, W_5, W_3, W_2 & W_4{:}M_4, M_3, M_2, M_1, M_5 \\
M_5{:}W_4, W_1, W_2, W_5, W_3 & W_5{:}M_1, M_2, M_4, M_5, M_3
\end{array}$$

We will use the Gale-Shapley algorithm to find a stable matching.

Round 1 *engaged couples:* $\emptyset$

M_1 proposes to W_2, M_2 proposes to W_3, M_3 proposes to W_2, M_4 proposes to W_4 and M_5 proposes to W_4.

W_2 has received proposals from M_1 and M_3; she prefers M_3, so they become engaged. W_3 receives one proposal from M_2, so they become engaged. W_4 has received proposals from M_4 and M_5; she prefers M_4, so they become engaged.

Round 2 *engaged couples:* $\{(M_3, W_2), (M_2, W_3), (M_4, W_4)\}$

M_1 proposes to W_3 and M_5 proposes to W_1.

W_3 prefers M_1 to her fiancé M_2, so she breaks off her previous engagement, and becomes engaged to M_1. W_1 becomes engaged to M_5.

Round 3 *engaged couples:* $\{(M_3, W_2), (M_1, W_3), (M_4, W_4), (M_5, W_1)\}$

M_2 is no longer engaged, so he proposes to W_1.

W_1 prefers M_2 to her current fiancé M_5, so she breaks off the engagement and becomes engaged to M_2.

Round 4 *engaged couples:* $\{(M_3, W_2), (M_1, W_3), (M_4, W_4), (M_2, W_1)\}$

M_5 is no longer engaged, so he proposes to W_2.

W_2 prefers M_5 to her current partner M_3, so she breaks off the engagement and becomes engaged to M_5.

Round 5 *engaged couples:* $\{(M_5, W_2), (M_1, W_3), (M_4, W_4), (M_2, W_1)\}$

M_3 is no longer engaged, so he proposes to W_5.

W_5 accepts the proposal and becomes engaged to M_3. At this point all the men are engaged, so the algorithm terminates.

The resulting matching is $\{(M_5, W_2), (M_1, W_3), (M_4, W_4), (M_2, W_1), (M_3, W_5)\}$.

Exercise 8.16. Check that the matching we have just obtained is indeed stable, by confirming that for each man there is no woman he ranks higher than his wife who also ranks him higher than her husband.

Exercise 8.17. Apply the Gale-Shapley algorithm to find a stable matching for the preference lists given in Example 8.13.

The Gale-Shapley Algorithm Terminates. From the description of the algorithm, we know that it terminates if it reaches a point at which all the men are engaged. Since the number of men equals the number of women, this is equivalent to requiring all the women to be engaged. As the algorithm is carried out, if a woman is engaged at the end of some round, then she will be engaged at the end of all future rounds, since she only ever breaks off an engagement in order to form a new engagement with a higher ranking partner. Thus the number of women who are engaged at the end of each round is nondecreasing.

If a woman receives one or more proposals during a round, then she will be engaged at the end of the round. While there is at least one man who is not engaged then there will be at least one new proposal per round. Since each man proposes to each woman on his list at most once, then eventually every woman must receive a proposal. (Certainly this must happen before n^2 rounds have occurred.) Thus the algorithm terminates. □

The Gale-Shapley Algorithm Outputs a Stable Matching. It is straightforward to see that the output of the algorithm is a complete matching. To show that this matching is stable, we need to prove that their is no unstable pair.

Suppose there is a pair of couples (M_a, W_b) and (M_c, W_d) that is unstable. Without loss of generality, we can suppose M_a rates W_d more highly than W_b and W_d rates M_a more highly than M_c. As the Gale-Shipley algorithm proceeds, each man proposes to women in the order they occur in his preference list, so M_a must have proposed to W_d before he proposed to W_b. Once W_d receives a proposal from M_a, then she will only reject him if she receives a proposal from a man she ranks more highly. But having become engaged to a higher ranking man, she then will never become engaged to the lower ranking M_c, which contradicts our initial assumption.

Thus we conclude that the matching output by the Gale-Shapley algorithm contains no unstable pairs, and hence is a stable matching. □

This provides a proof of our earlier assertion that for any lists of preferences you can find a stable matching. There is one further interesting property of the matchings produced by the Gale-Shapley algorithm that we have not yet addressed.

Theorem 8.6. *Suppose n men and n women rank each other in terms of preference. If the matching output by the Gale-Shapley algorithm results in man M_i marrying woman W_j, then there is no other stable matching in which man M_i marries a woman he ranks higher than woman W_j.*

In other words, if you consider all the stable matchings that are possible for a given list of preferences, you will find that the matching arising from the Gale-Shapley algorithm results in each man marrying a woman that he ranks at least as highly as the woman he marries in any other matching. Thus we say that the Gale-Shapley matching is *male optimal.* This is quite a strong property, and it is not immediately obvious that a stable matching with this property is guaranteed even to exist.

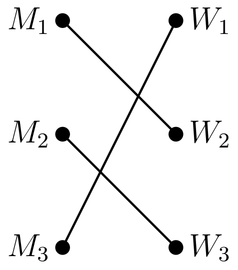
The stable matching output by the Gale-Shapley algorithm is male optimal. Suppose n men and n women rank each other in terms of preference. For any given man, we define his *optimal partner* to be the highest-ranking woman that he marries in any stable matching for those preference lists.

Suppose that the Gale-Shapley algorithm does not produce a male optimal stable matching. Then we conclude that at least one man is rejected by his optimal partner during the execution of the algorithm. Suppose that man M_1 is the first man to be rejected by his optimal partner, woman W_1. This rejection must occur as the result of a proposal from some man M_2 that woman W_1 ranks more highly than M_1. As M_1 is the first man to be rejected by his optimal partner, it follows that M_2 must rank woman W_1 at least as highly as his optimal partner.

Now, since woman W_1 is the optimal partner of M_1, it follows that there is some stable matching $\mathcal{M}$ in which M_1 marries W_1. Denote by W_2 the wife of M_2 in $\mathcal{M}$. Since M_2 ranks W_1 at least as high as his optimal partner, he must prefer W_1 to W_2. Additionally, we have already observed that W_1 prefers M_2 to M_1. Hence the couples (M_1, W_1) and (M_2, W_2) form an unstable pair for $\mathcal{M}$, contradicting the fact that it is a stable matching.

Thus we conclude that the Gale-Shapley matching is indeed male optimal. □

Exercise 8.18. In Example 8.14, two examples of stable matchings for the given preference lists are provided. There is precisely one more stable matching possible for these lists:



1. Apply the Gale-Shapley algorithm to the preference lists to find which of these stable matchings is male optimal. Can you see a way to argue directly that this stable matching is male optimal without having to compare it with the other stable matchings?

2. There are three other possible matchings for these six people. Write down these matchings, and in each case find an unstable pair of couples to show that it is not a stable matching.

A male optimal stable matching gives the best possible result of any stable matching for all the men involved, but what about the women? Unfortunately for them, it turns out that a male optimal matching results in each woman marrying a partner that is ranked at least as low as her partner in any other stable matching. In other words, it is *female pessimal.*

Theorem 8.7. *Any stable matching that is male optimal is female pessimal.*

Proof. Suppose $\mathcal{M}_1$ is a male optimal stable matching that is not female pessimal. Then for some woman W_1 who is married to a man M_1 in $\mathcal{M}_1$, there is a matching $\mathcal{M}_2$ in which she marries a man M_2 whom she likes less than M_1. Let W_2 be the wife of M_1 in $\mathcal{M}_2$. Then $\{(M_1, W_2), (M_2, W_1)\}$ is an unstable pair, as woman W_1 prefers M_1 to M_2, and M_1 prefers W_1 to W_2 (as she is his optimal partner). This contradicts the assumption that $\mathcal{M}_2$ is stable. $\square$

If we decide that we prefer to have a female optimal matching, we simply have to reverse the roles of the men and the women in the Gale-Shapley algorithm.

The whole question of stable marriages may seem a bit contrived. However, there are more serious uses for these ideas, such as matching junior doctors to hospitals in which they will do their training. In this case the doctors could be represented by the men and the hospitals by the women, or *vice versa.*

Exercise 8.19. Four women and four men rank each other as follows:

$$\begin{array}{ll}
M_1{:}W_2, W_4, W_1, W_3 & W_1{:}M_3, M_2, M_4, M_1 \\
M_2{:}W_4, W_2, W_3, W_1 & W_2{:}M_2, M_1, M_4, M_3 \\
M_3{:}W_3, W_1, W_4, W_2 & W_3{:}M_4, M_1, M_3, M_2 \\
M_4{:}W_2, W_4, W_1, W_3 & W_4{:}M_4, M_2, M_3, M_1
\end{array}$$

1. Use the Gale-Shapley algorithm to find a male-optimal stable matching.

2. Now find a female-optimal stable matching.

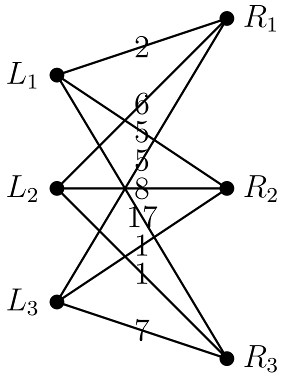
8.4 Optimal Assignments and the Hungarian Algorithm

We will now consider one last class of matching problems, that of finding *optimal assignments.*

Definition 8.5. Let G be a weighted complete bipartite graph. An *optimal assignment* for G is a perfect matching in which the total weight of all the edges is as small as possible for any perfect matching. The *assignment problem* is the problem of finding an optimal assignment for G.

(In some contexts you will be interested in finding assignments for which the total weight of the edges is as *large* as possible; we will touch on this later, but for now we will stick to the case where we are trying to minimise the total weight.)

Example 8.20. Let G be the following weighted bipartite graph:



The matching $\{(L_3, R_2), (L_2, R_3), (L_1, R_1)\}$ has total weight 4, and it is clear in this case that no other perfect matching has lower total weight. Hence this is an optimal assignement.

The reason for the name “optimal assignment” comes from applications like that described in the following example.

Example 8.21. Suppose I have four tasks T_1, T_2, T_3 and T_4 that need to be completed, and I wish to employ four workers W_1, W_2, W_3 and W_4 to carry out the tasks. If the amount each worker charges for performing each task is given in the following table, how can I assign a different worker to each task so that my total wage bill is minimised?

	T_1	T_2	T_3	T_4
W_1	1	5	3	2
W_2	3	4	4	2
W_3	4	3	2	2
W_4	2	3	3	4

We could represent this information as a complete bipartite graph, with a vertex representing worker W_i joined to a vertex representing task T_j by an edge whose weight is the amount that worker charges to perform that task.

There is a technique for finding optimal assignments that is known as the *Hungarian algorithm.* (It is sometimes also known as the Kuhn-Munkres algorithm after the people who came up with it, but Kuhn gave it the name ‘Hungarian algorithm’ because he based it on the work of two Hungarians, Kőnig and Egerváry.)

We will describe how this algorithm can be carried out by working with the entries in the table of costs. First, however, we will discuss some basic motivation for the algorithm.

Suppose there are n workers and n tasks. We wish to assign precisely one worker to each task, and to minimise the associated total cost. In terms of the representation of the problem as a bipartite graph, we wish to find a perfect matching such that the total weight of edges in the matching is minimised. If we consider the table of costs, the entry in row i, column j represents the cost for worker W_i to carry out task T_j. Thus, what we are looking for is a set of n entries in the table with precisely one in each row (as each worker is only assigned one task) and one in each column (since we don’t want any two workers performing the same task,) such that the sum of the entries is minimised. We call a set of entries an *independent set* if no two of them lie in the same row or the same column.

How can we find an independent set of entries whose sum is minimised? There is one case in which this is straightforward: suppose all entries in the matrix are nonnegative, and there is an independent set of entries that are each equal to zero. Such a set clearly has the minimum possible sum. The basic strategy of the Hungarian algorithm is to modify the matrix of costs associated with an assignment problem until it is in this form. The fact that we can do this relies on the following two results.

Lemma 8.8. *Let M be a matrix of costs for an instance of the assignment problem. Suppose we define a new matrix M' by picking a row M and adding a constant λ to each element in that row. Then an optimal assignment for the assignment problem defined by M' is an optimal assignment for the assignment problem defined by M and vice versa, and the total cost of an optimal assignment for M' is equal to the cost of an optimal assignment for M plus λ.*

Proof. Suppose M is an $n \times n$ matrix. An independent set containing n entries has precisely one entry in any given row, so the sum of the entries in that set for M' is equal to the sum of the corresponding entries in M plus γ. From this we see immediately that the entries corresponding to an optimal assignment for M also give rise to an optimal assignment for M' and *vice versa.* □

This means that when trying to solve an assignment problem, we can add or subtract a fixed amount to/from the entries of any row of the matrix of costs, and a solution for the new matrix will give a solution for the original problem. Similarly, we can add or subtract a fixed amount to/from the entries of any column.

Lemma 8.9 (The Kőnig-Egerváry Theorem)**.** *Let M be a square matrix. The size of the largest independent set of entries that are equal to zero is equal to the smallest number of rows and/or columns that between them contain all zero entries of the matrix.*

Example 8.22. Consider the matrix

$$M = \begin{pmatrix} \mathbf{0} & 1 & 1 & 9 \\ 1 & 0 & \mathbf{0} & 0 \\ 2 & \mathbf{0} & 17 & 5 \\ 1 & 0 & 2 & 3 \end{pmatrix}$$

The first two rows, together with the second column between them contain all the zero entries of M. Thus the size of any independent set of zero entries is at most 3. One example of an independent set of zeros of size 3 is shown in bold type. Note that for this lemma, we are not concerned with the values of any nonzero entries, merely with the question of which entries are zero and which are not.

Proof. Although this lemma deals with entries in matrices, it is in fact closely related to results we have seen previously regarding matchings in bipartite graphs. The easiest way to see this is to construct a bipartite graph G from the matrix M: we have a set of n vertices that correspond to the rows of the matrix, and a second set of n vertices that correspond to the columns. We add an edge joining the vertex corresponding to row i to the vertex corresponding to row j when the ij^{th} entry of M is a zero. Then an independent set of zeros in M corresponds to a matching in G.

A set of k rows and/or columns that between them contain all zero entries of M corresponds to a set of k vertices of G with the property that each edge of G is incident with at least one of these vertices. Recall that when using Menger's theorem to prove Hall's theorem we converted a bipartite graph G into a graph G' by adding extra vertices v and w. We perform the same construction in this case, and observe that a set of vertices with the property that every edge of G is incident with at least one of those vertices must be a vw-separating set in G'. As we observed in the proof of Hall's theorem, the minimum size of such a set is equal to the size of the largest matching in G, by the vertex form of Menger's theorem. This proves the desired result. □

The basic approach that we will take to solve the assignment problem is thus the following:

> We repeatedly select a row or column of the cost matrix and add some constant to all the entries in the row column such that
>
> 1. all the entries in the matrix remain nonnegative;
> 2. the maximum possible size of an independent set of zero entries gradually increases
>
> until we can find an independent set of zeros of size n. Once we have such a set, the corresponding entries of the matrix give us the optimal assignment.

We will now provide an outline of the Hungarian algorithm for performing this task, before giving a more detailed description of the individual steps involved in the algorithm.

Theorem 8.10 (Outline of the Hungarian Algorithm). *Let M be a matrix giving the costs associated with an instance of the assignment problem. An optimal assignment can be found by the following method:*

preliminaries *For each row of the matrix, we find the smallest element in the row and subtract that value from each of the entries in that row. Then we repeat this process for the columns, finding the smallest element in each column and subtracting that element from all entries in the column.*

Having carried out these steps, we now have at least one zero in each row and each column. Sometimes this is enough to let us find an independent set of n zero entries and thus solve the assignment problem. If not, we then have to continue to alter the matrix to create some new zero entries. However, at this point we cannot simply continue to subtract values from rows/columns as this would lead to the matrix having negative entries. Instead, we repeat the following steps:

Step 1 *Find the smallest possible set S_1 of rows and/or columns that between them contain all the zeros of M. Suppose $|S_1| = n_1$.*

Step 2 *If $n_1 < n$, we find the smallest entry h_1 that is not in one of the rows or columns of S_1. We subtract h_1 from every element of M, then for each row or column of S_1 we add h_1 back to each element of that row or column. (Note that this prevents any of the entries from becoming negative.)*

Step 3 *Repeat the previous two steps until a matrix with an independent set of n zeros is obtained. (We note that if n_k is the size of the largest independent set of zeros after k repetitions of this process, then during step 2 the sum of all the entries in M is reduced by $n(n - n_k)h_k$. Since the sum of the entries is necessarily nonnegative, this implies that the process must terminate.)*

Example 8.23. The following matrix gives the amount that workers A_1 to A_5 charge to perform jobs j_1 to j_5.

	j_1	j_2	j_3	j_4	j_5
A_1	3	2	4	5	7
A_2	2	1	2	5	2
A_3	2	3	4	7	8
A_4	4	7	2	3	5
A_5	5	2	4	6	4

The smallest element in the first row is 2, so we subtract 2 from each entry in the row. Similarly, the smallest entry in row 2 is 1, so we subtract 1 from all entries in this row. Carrying out this process for all five rows gives rise to the following matrix:

	j_1	j_2	j_3	j_4	j_5
A_1	1	0	2	3	5
A_2	1	0	1	4	1
A_3	0	1	2	5	6
A_4	2	5	0	1	3
A_5	3	0	2	4	2

Now we turn our attention to the columns. The first three columns contain zeros, but the smallest element in column 4 is 1, hence we subtract 1 from all elements of that column. Similarly, the smallest element in column 5 is 1 so we subtract 1 from all elements of column 5.

	j_1	$\mathbf{j}_2$	j_3	j_4	j_5
A_1	1	**0**	2	2	4
$\mathbf{A}_2$	**1**	**0**	**1**	**3**	**0**
$\mathbf{A}_3$	**0**	**1**	**2**	**4**	**5**
$\mathbf{A}_4$	**2**	**5**	**0**	**0**	**2**
A_5	3	**0**	2	3	1

We observe that rows 2, 3 and 4, together with column 2 (all marked in bold) contain all the zeros of this matrix, and in fact the largest independent set of zeros has size 4. Thus we must progress to step 2 of the algorithm.

The smallest entry of the matrix that doesn't lie in row 2, 3, 4 or column 2 is 1. Thus we subtract 1 from all elements of the matrix. Then we add 1 to all the elements in rows 2, 3, and 4, and we add 1 to all the elements in column 2. This gives the following matrix:

	$\mathbf{j}_1$	$\mathbf{j}_2$	j_3	j_4	$\mathbf{j}_5$
A_1	**0**	**0**	1	1	**3**
A_2	**1**	**1**	1	3	**0**
A_3	**0**	**2**	2	4	**5**
$\mathbf{A}_4$	**2**	**6**	**0**	**0**	**2**
A_5	**2**	**0**	1	2	**0**

However, row 4 together with columns 1,2 and 5 (marked in bold) contain all the zeros of this matrix, so there is still no independent set of size 5. Hence we repeat step 2, which yields

	j_1	j_2	j_3	j_4	j_5
A_1	0	0	0	0	3
A_2	1	1	0	2	0
A_3	0	2	1	3	5
A_4	3	7	0	0	3
A_5	2	0	0	1	0

Now this matrix *does* have an independent set of 5 zeros. In fact, it has three such sets:

$$\{A_1j_2, A_2j_3, A_3j_1, A_4j_4, A_5j_5\}$$
$$\{A_1j_3, A_2j_5, A_3j_1, A_4j_4, A_5j_2\}$$
$$\{A_1j_4, A_2j_5, A_3j_1, A_4j_3, A_5j_2\}$$

Returning to the original matrix, we see that each of these gives an assignment of total cost 13.

Having seen an overall description of the Hungarian algorithm, together with an example of how it can be applied, we now consider in more detail some of the individual steps that are necessary for carrying out the algorithm.

Exercise 8.24. Find a maximum independent set of zero entries in the following matrix:

$$\begin{array}{ccccc} 1 & 0 & 2 & 2 & 4 \\ 1 & 0 & 0 & 3 & 0 \\ 0 & 1 & 2 & 4 & 5 \\ 2 & 5 & 0 & 0 & 2 \\ 3 & 0 & 2 & 3 & 1 \end{array}$$

Hence find a smallest possible set of rows and/or columns that between them contain all the zero entries of the matrix.

8.4.1 Maximising the Sum of Edge Weights in an Assignment

So far we have been considering an optimal assignment to be one that minimises the total weight of the edges in a perfect matching. There are some situations, however, in which we would like to *maximise* the total weight.

Example 8.25. Suppose the entries in the following table indicate how suitable each of 5 workers are for performing 5 separate tasks (a higher number indicates a greater suitability.)

	j_1	j_2	j_3	j_4	j_5
W_1	5	6	2	2	5
W_2	4	6	3	5	4
W_3	2	4	5	6	4
W_4	4	6	4	4	4
W_5	4	3	5	5	5

In this scenario, we would like to be able to assign the tasks to the workers so that the total suitability is maximised.

Fortunately, we can use a simple trick to transform a problem such as this into one that we can already solve: just let x be the largest entry in the matrix, and replace each entry M_{ij} by $x - M_{ij}$. Then we can apply the Hungarian Algorithm to the new matrix to obtain an independent set of entries whose sum is as small as possible, and this will give us a set whose sum is as large as possible in the original matrix.

Exercise 8.26. Use this trick to find the most suitable assignment for Example 8.25.

8.4.2 Unbalanced Assignment Problems

The Hungarian algorithm is designed to work for square cost matrices, and so can be used to solve the assignment problem when the number of workers is equal to the number of tasks. Sometimes, however, we may be interested in finding an assignment for the case where these two numbers are not equal. In this case, an optimal assignment is a complete matching of lowest cost in the corresponding bipartite graph, rather than a perfect matching. For example, we may have 5 workers, but only 3 tasks. In this case we would like to choose 3 of the workers to complete the tasks as cheaply as possible. We refer to this as an *unbalanced assignment problem.*

If we attempt to apply the Hungarian Algorithm to a matrix that is not square, then it may not work successfully. An approach that does work, however, is to convert the cost matrix into a square matrix by adding extra rows or columns as needed. We refer to these as *dummy rows* (or dummy columns) and can simply let all the entries in these rows be 0. We then apply the Hungarian Algorithm to this square matrix. Once we have obtained an optimal assignment, we simply ignore entries coming from the dummy rows/columns.

Example 8.27. Suppose we have three workers and we want to assign each of them to do a different task, with five possible tasks to choose from. The amount each worker charges to complete each task is given in the following table, and we would like to find the cheapest possible way of assigning the tasks to the workers.

	j_1	j_2	j_3	j_4	j_5
W_1	3	7	2	2	7
W_2	5	4	2	6	3
W_3	2	3	6	4	3

In order to be able to apply the Hungarian algorithm here, we need to insert two dummy rows in order to obtain a square matrix:

	j_1	j_2	j_3	j_4	j_5
W_1	3	7	2	2	7
W_2	5	4	2	6	3
W_3	2	3	6	4	3
D_1	0	0	0	0	0
D_2	0	0	0	0	0

We can now apply the Hungarian algorithm directly to this matrix.

The motivation behind this approach is that the fact that the entries in the dummy rows are all the same means that they won't have any effect on which combinations of entries from the "genuine" rows lead to the cheapest assignment.

Exercise 8.28. Use the Hungarian algorithm to solve the unbalanced assignment problem presented in Example 8.27

8.4.3 The Bottleneck Problem

We have considered assignment problems in which the goal was either to maximise or to minimise the total weight of the assignment. In some circumstances, however, we may have slightly different goals. Suppose we wish to assign employees to jobs, and we are told the rate at which each employee can complete each job. If we want to have all the jobs completed as soon as possible, then we want to find an assignment for which the smallest rate is as large as it can be, since a low rate means the task will take a long time to complete, leading to a *bottleneck*. This is known as the *bottleneck problem*, and we can find a solution to it by means of the following algorithm.

Theorem 8.11. *Given an $n \times n$ rate matrix M, an assignment whose smallest value is as large as possible can be found using the following steps:*

1. *Choose any assignment (by selecting an independent set of n entries of M).*
2. *Find the smallest entry in this assignment, say Θ.*
3. *Create a new $n \times n$ matrix T by placing a 0 in any position where the entry of M is greater than Θ, and a 1 otherwise.*
4. *If the largest independent set of zero entries of T has size less than n then the assignment is optimal. If T has an independent set of size n, then the assignment given by these entries has a larger smallest value than the previous assignment. We repeat this process from Step 2, using this new assignment.*

Example 8.29. Suppose we are given the following rate matrix:

$$\begin{array}{ccccc}
4 & 10 & 3 & 8 & 3 \\
7 & 8 & 7 & 6 & 2 \\
5 & 6 & 5 & 3 & 2 \\
5 & 2 & 7 & 3 & 2 \\
6 & 3 & 9 & 5 & 2
\end{array}$$

Suppose we take our initial assignment to consist of all entries on the main diagonal. The smallest entry in this assignment is $\Theta = 2$, hence we construct the following matrix T:

$$\begin{array}{ccccc}
0 & 0 & 0 & 0 & 0^* \\
0 & 0 & 0 & 0^* & 1 \\
0 & 0 & 0^* & 0 & 1 \\
0^* & 1 & 0 & 0 & 1 \\
0 & 0^* & 0 & 0 & 1
\end{array}$$

This matrix *does* have an independent set of zero entries of size n (marked with stars).

We return to the original matrix M and consider this new assignment:

$$\begin{array}{ccccc}
4 & 10 & 3 & 8 & 3^* \\
7 & 8 & 7 & 6^* & 2 \\
5 & 6 & 5^* & 3 & 2 \\
5^* & 2 & 7 & 3 & 2 \\
6 & 3^* & 9 & 5 & 2
\end{array}$$

Now we have $\Theta = 3$, so we construct a new matrix T:

$$\begin{array}{ccccc}
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1
\end{array}$$

As column 5 of this matrix contains no zeros, it is impossible to find an independent set of 5 zero entries. Thus we conclude that this current assignment has the largest possible value of Θ.

Note that for each new assignment we find, the value of Θ is strictly greater than in previous assignments. Therefore the number of zeros in subsequent matrices T gradually decreases, until a point is reached at which there is no independent set of zero entries with size n.

Exercise 8.30. Let M be the following matrix.

$$\begin{array}{ccccc}
4 & 8 & 2 & 4 & 2 \\
2 & 4 & 12 & 4 & 3 \\
2 & 3 & 2 & 2 & 3 \\
3 & 4 & 3 & 7 & 5 \\
2 & 3 & 2 & 3 & 5
\end{array}$$

Solve this bottleneck problem by finding an assignment whose smallest value is as large as possible.

Learning Outcomes

After completing this chapter and the related problems you should be able to:

- Count the number of pairings of a set;
- Know the definitions of a matching, maximal matching, maximum matching, and perfect matching in a graph;
- State a sufficient condition for a general graph G to have a perfect matching;
- Know the definition of a complete matching of a bipartite graph;
- State Hall's Theorem, and apply the Hall condition to determine whether a bipartite graph has a complete matching;

- Understand the concept of the deficiency of a bipartite graph;
- Know the definition of a stable matching;
- Apply the Gale-Shapley Algorithm to find a stable matching;
- Recognise that the matching output by the Gale-Shapely algorithm is male optimal;
- Prove that a male-optimal stable matching is female pessimal'
- Apply the Hungarian Algorithm to solve the assignment problem in both the balanced and unbalanced cases;
- Apply the Kőnig-Egerváry theorem to show that a given independent set of zero entries of a matrix is as large as possible;
- Solve the Bottleneck Problem.