CSC5160: Combinatorial Optimization and Approximation Algorithms
Topic: General Matching **Date:** 24/01/2008
Lecturer: Lap Chi Lau **Scribe:** Jennifer X.M. WU

In this lecture, we discuss *matchings in general graph*. Following the same structure of *bipartite matching*, we first introduce the min-max theorem for *general matching*. Then we present Edmonds' polynomial-time algorithm to find a maximum-size matching. Finally, we solve the Chinese postman problem using general matching.

We have considered matchings in bipartite graph, now we will consider matchings in general graph. Recall that a graph is bipartite if and only if it contains no odd cycles. Given a graph with weighted edges, the goal of the Maximum Matching problem is to find a matching (a set of vertex-disjoint edges) with maximum total weight. In this lecture, we focus on the cardinality version where the goal is to find a matching with the maximum number of edges.

5.1 Min-Max Theorem for General Matching

In bipartite matching, we have König's theorem as the min-max theorem, which says that the maximum size of a matching is equal to the minimum size of a vertex cover in a bipartite graph. However, this is not true for general graphs. Consider a complete graph of n vertices. The maximum matching is of size $n/2$ but a minimum vertex cover needs $n-1$ vertices.

There is a min-max theorem for general matching. Tutte in 1947 gave a necessary and sufficient condition characterizing graphs that have a perfect matching. Berge in 1958 observed that it implies a min-max formula for the maximum size of a matching in a graph, now known as the Tutte-Berge formula, which is a generalization of Tutte's theorem.

5.1.1 Tutte Theorem

Definition 5.1.1 An *odd* component in a graph G is a connected component of a graph which has odd number of vertices.

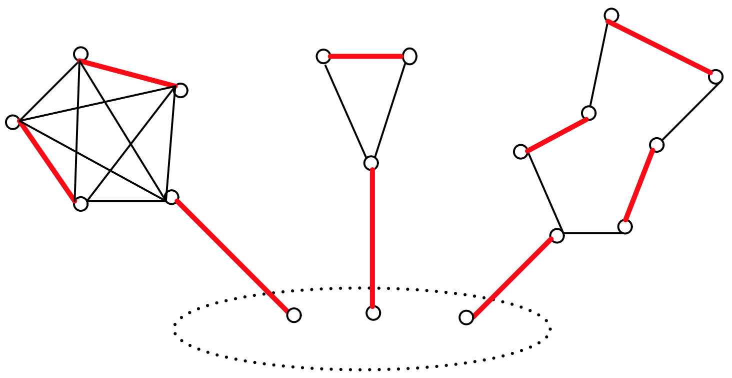


Figure 5.1.1: Perfect Matching in a General Graph

In Figure 5.1.1, suppose we remove all the vertices in the dotted set U, and there are some odd components in $G-U$. To find a perfect matching, each odd component must have at least one vertex which matches with one vertex in U. So if there are more odd components in $G-U$ than the number of vertices in U, then G does not have a perfect matching. So for any subset U, in order to have a perfect matching, we must have $o(G-U) \leq |U|$, where $o(G-U)$ denotes the number of odd components of $G-U$. Tutte proved that it is also a sufficient condition.

Theorem 5.1.2 *(Tutte's theorem). A graph $G=(V,E)$ has a perfect matching if and only if $G-U$ has at most $|U|$ odd components, for each $U \subseteq V$.*

It is a good exercise to see how Tutte's theorem implies Hall's theorem for the existence of a perfect matching in a bipartite graph.

5.1.2 Tutte-Berge Formula

Now suppose $o(G-U) \geq |U|+2k$. By a similar argument as above, there will be at least $2k$ unmatched vertices, one from each "extra" odd component in $G-U$. Tutte-Berge formula makes this precise. In the following $o(G)$ denotes the number of odd components of G and $\nu(G)$ denotes the maximum size of a matching.

Theorem 5.1.3 (TUTTE-BERGE FORMULA) *For each graph $G=(V,E)$,*

$$\nu(G) = \min_{U \subseteq V} \frac{1}{2}(|V|+|U|-o(G-U)). \tag{5.1.1}$$

Proof: To see $\leq$, consider an arbitrary $U \subset V$. Divide the edge set of a maximum matching M into two subset M_1 and M_2, where M_1 consists of the edges which use vertices in U, and M_2 consists of the edges which do not use vertices in U.

$$\nu(G) = |M| = |M_1|+|M_2| \leq |U|+\nu(G-U) \leq |U|+\frac{1}{2}(|V\setminus U|)-o(G-U)) = \frac{1}{2}(|V|+|U|-o(G-U)).$$

It is clear that $|M_1| \leq |U|$ and $|M_2| \leq \nu(G-U)$ and hence the first inequality. For the second inequality, observe that each odd component in $G-U$ must have an unmatched vertex, and so $|M_2| \leq \frac{1}{2}(|V \setminus U| - o(G-U))$.

We will prove $\geq$ after we obtain an algorithm for the maximum matching problem. ■

5.2 Maximum Matching Algorithm in General Matching

5.2.1 Augmenting Path Algorithm?

Recall that we use the concept of an M-augmenting path to find maximum matchings in bipartite graphs. See figure 5.2.2. Given a matching M, an M-alternating path is a path that alternates between edges in M and edges not in M. An M-alternating path whose end points are unsaturated by M is an M-augmenting path. If M is a matching and P is an M-augmenting path, then $M \oplus P$ is a bigger matching, $|M \oplus P| = |M|+1$ (see Figure 5.2.2). If there is no M-augmenting path, then M is a maximum matching. So we have the following polynomial time algorithm for maximum bipartite matching.

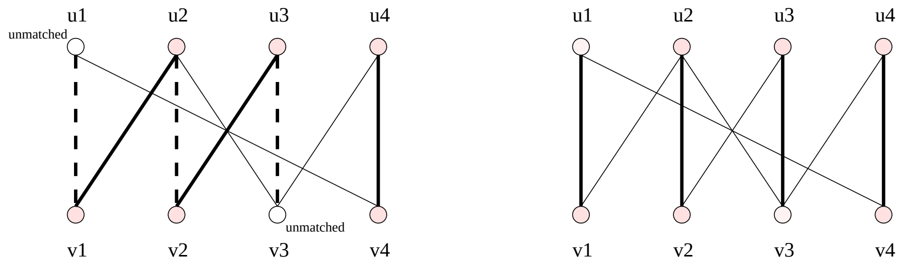


Figure 5.2.2: An example of the augmentation procedure for finding M^*

Maximum Bipartite Matching Algorithm

```
1. Initialization $M := \emptyset$.
2. While there is an $\underline{M\text{-augmenting path}}$ $P$
       set $M := M \oplus P$
3. Return $M$.
```

Figure 5.2.3: Maximum Bipartite Matching Algorithm

Can we use the same algorithm for general matching? The bipartite matching algorithm is based on the following result. Is the same result still true in general graphs?

Theorem 5.2.1 *A matching M in general graph is maximum if and only if there is no M-augmenting path.*

The idea of finding an M-augmenting path to increase a matching M is fundamental in finding a maximum-size matching. The theorem giving optimality condition in bipartite matching still holds for general matching; the reader is encouraged to check the proof of this result in lecture 3. So apparently the same algorithm should be applied to general matchings as well. But why does it take decades for researchers to find an efficient algorithm for general matchings? It turns out the difficulty is in finding an M-augmenting path efficiently, which is often assumed to be an easy step in our previous discussion on bipartite matchings.

5.2.2 Finding M-augmenting path

In bipartite graph, since there is no odd cycle in the graph, we can orient the edges so that an M-augmenting path corresponds to a directed path. Hence the M-augmenting path can be found by BFS or DFS in linear time $O(m)$. When it comes to general graph, which contains odd cycles, we simply don't know how to orient the edges so that an M-augmenting path corresponds to a directed path between two unmatched vertices. How about we don't orient the edges, but simply find an alternating path without repeated vertices? It may looks not so difficult. However, the alternating path may run into a loop that we cannot simply delete it. We just don't know how to find it efficiently. See Figure 5.2.4.

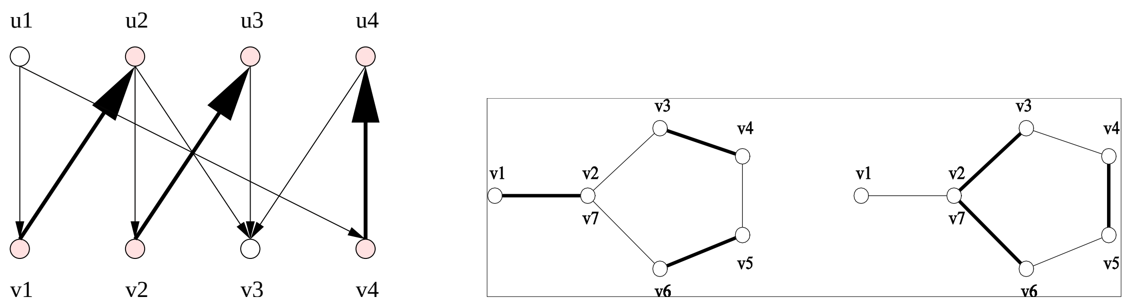
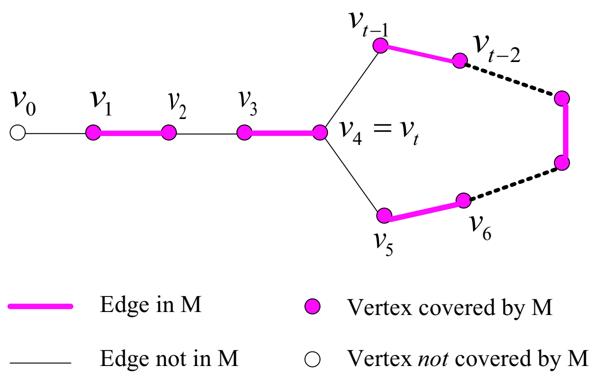


Figure 5.2.4: Finding M-augmenting path in bipartite and general graph

5.2.3 Edmonds' polynomial-time algorithm

It was Jack Edmonds, in his famous paper "Paths, trees, and flowers", who found the trick to resolve this problem. In order to introduce Edmonds' polynomial-time algorithm, we first introduce the term "Blossom".

Definition 5.2.2 *Let $G = (V, E)$ be a graph, let M be a matching in G, and let X be the set of vertices missed(don't covered by M) by M. An M-alternating walk $P = (v_0, v_1, ..., v_t)$ is called an M-flower if t is odd, $v_0, ..., v_{t-1}$ are distinct, $v_0 \in X$, and $v_t = v_i$ for some even $i < t$. Then the circuit $(v_i, v_{i+1}, ..., v_t)$ is called an M-*blossom*(associated with the M-flower).*



The core of the algorithm is based on the following observation. Let $G = (V, E)$ be a graph and let B be a subset of V. Denote by G/B the graph obtained by *contracting(or shrinking)* B to one new vertex, called B. That is, G/B has vertex set $(V \setminus B) \bigcup B$, and for each edge e of G an edge obtained from e by replacing any end vertex in B by the new vertex B. (We ignore loops that may arise.) We denote the new edge again by e. (So its ends are modified, but not its name.) We say that the new edge is the image(or projection) of the original edge.

For any matching M, let M/B denote the set of edges in G/B that are images of edges in M not *spanned* by B, i.e., the set of edges in M with at least one endpoint not in B. For a blossom B,

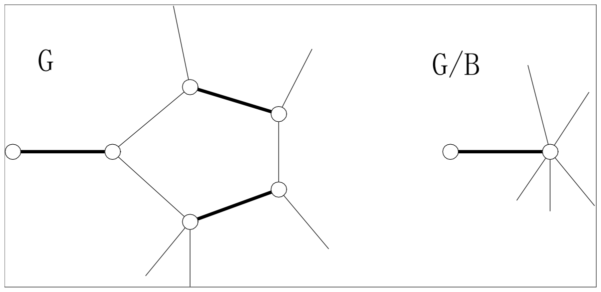


Figure 5.2.5: Shrink Blossom

since M intersects $\delta(B)$ in at most one edge, M/B is a matching in G/B. The following is the key of Edmonds' algorithm.

Theorem 5.2.3 *Let B be an M-blossom in G. Then M is a maximum-size matching in G if and only if M/B is a maximum-size matching in G/B.*

Proof: Let $B = (v_i, v_{i+1}, ..., v_t)$.

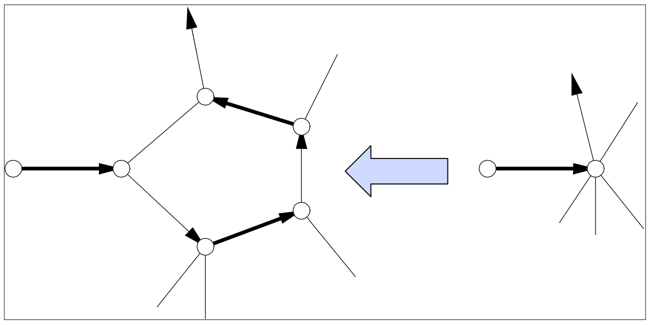


Figure 5.2.6: An M/B-augmenting path $\Rightarrow$ an M-augmenting path

First assume that M/B is not a maximum-size matching in G/B. Let P be an M/B-augmenting path in G/B. If P does not traverse vertex B of G/B, then P is also an M-augmenting path in G. If P traverses vertex B, we may assume that it enters B with some edge uB that is not in M/B. Then $uv_j \in E(G)$ for some $j \in \{i, i+1, ..., t\}$.

- If j is odd, replace vertex B in P by $v_j, v_{j+1}, ..., v_t$.
- If j is even, replace vertex B in P by $v_j, v_{j-1}, ..., v_i$.

In both cases we obtain an M-augmenting path in G. So M is not maximum-size. See Figure 5.2.6.

Conversely, assume that M is not maximum-size. We may assume that $i = 0$, that is, $v_i \in X$, since replacing M by $M \bigoplus EQ$, where Q is the path $(v_0, v_1, ..., v_i)$, does not modify the theorem. That is, we may assume that B is an unmatched vertex in M/B. Let $P = (u_0, u_1, ..., u_s)$ be an M-augmenting path in G. If P does not intersect B, then P is also an M/B-augmenting path in

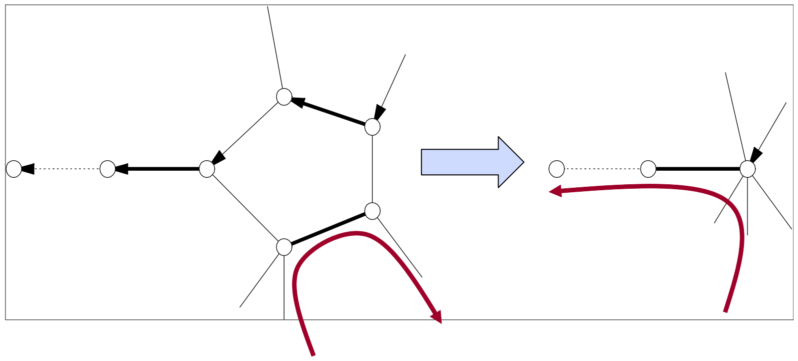


Figure 5.2.7: An M-augmenting path $\Rightarrow$ an M/B-augmenting path.

G/B. If P intersects B, we may assume that u_0 is not in B. (Otherwise replace P by its reverse.) Let u_j be the first vertex of P in B. Then $(u_0, u_1, ..., u_{j-1}, B)$ is an M/B-augmenting path in G/B. So M/B is not of maximum-size. ■

Another useful observation is:

Theorem 5.2.4 *Let $P = (v_0, v_1, ..., v_t)$ be a shortest M-alternating $X - X$ walk. Then either P is an M-augmenting path or $(v_0, v_1, ..., v_j)$ is an M-flower for some $j \leq t$.*

Proof: Assume that P is not a path. Choose $i < j$ with $v_j = v_i$ and with j as small as possible. So $v_0, ..., v_{j-1}$ are all distinct. If $j - i$ would be even, we can delete $v_{i+1}, ..., v_j$ from P so as to obtain a shorter M-alternating $X - X$ walk. So $j - i$ is odd. If j is even and i is odd, then $v_{i+1} = v_{j-1}$ (as it is the vertex matched to $v_i = v_j$), contradicting the minimality of j. Hence j is odd and i is even, and therefore $(v_0, v_1, ..., v_j)$ is an M-flower. ■

We can now describe an algorithm for the problem of finding an M-augmenting path:

The Matching-Augmenting Algorithm

1. Given: a matching M;
   Find: an M-augmenting path, if any.
   Denote the set of vertices missed by M by X.

2. If there is no M-alternating $X - X$ walk of positive length, there is no M-augmenting path.

3. If there exists an M-alternating $X - X$ walk of positive length, choose a shortest one, $P = (v_0, v_1, ..., v_t)$ say.

   Case 1: *P* is a path. Then output P.

   Case 2: *P* is not a path. Choose j such that $(v_0, ..., v_j)$ is an M-flower, with M-blossom B. Apply the algorithm (recursively) to G/B and M/B, giving an M/B-augmenting path P in G/B. Expand P to an M-augmenting path in G (as in the proof of 5.2.3).

Figure 5.2.8: The Matching-Augmenting Algorithm

$O(n^2m)$	Edmonds [1965d] (cf. Witzgall and Zahn [1965])
$O(n^3)$	Balinski [1969] (also Gabow [1973, 1976a], Karzanov [1976], Lawler [1976b])
$O(nm\alpha(m,n))$	Gabow [1976a]
$O(n^{\frac{5}{2}})$	Even and Kariv [1975], Kariv [1976] (also Bartnik [1978])
$O(\sqrt{n}m\log n)$	Even and Kariv [1975], Kariv [1976]
$O(\sqrt{n}m\log\log n)$	Kariv [1976]
$O(\sqrt{n}m + n^{1.5+\varepsilon})$	Kariv [1976] for each $\varepsilon > 0$
$O(\sqrt{n}m)$	announced by Micali and Vazirani [1980], full proof in Blum [1900], fVazirani [1990,1994], and Gabow and Tarjan [1991] (cf. Gabow and Tarjan [1983,1985])
$O(\sqrt{n}m\log_n \frac{n^2}{m})$	Goldberg and Karzanov [1995]

Table 1: Faster Algorithms[1].

With an efficient algorithm for finding M-augmenting path, we now have a polynomial time algorithm for finding maximum matching in general graphs. The following result is by Edmonds.

Theorem 5.2.5 *Given a graph, a maximum-size matching can be found in time $O(n^2m)$.*

Proof: The algorithm directly follows from the above matching-augmenting algorithm, since, starting with $M = \emptyset$, one can iteratively apply it to find an M-augmenting path P and replace M by $M \bigoplus EP$. It terminates if there is no M-augmenting path, whence M is a maximum-size matching.

An M-alternating walk can be found in time $O(m)$. By Theorem 5.2.4, either there is an M-augmenting path (in which case we are done) or there is a blossom B. The shrinked graph G/B can be constructed in time $O(m)$, and it has fewer vertices than B. So the recursion has depth at most n, and hence an M-augmenting path can be found in time $O(nm)$. Since the number of augmentations is at most n, the time bound follows. ∎

5.3 Deriving Tutte-Berge Formula

We now show to how to derive Tutte-Berge formula from Edmonds' algorithm. The strategy is to first find a maximum matching using Edmonds' algorithm, and then construct the set U in the Tutte-Berge formula. For this we need the concept of an M-alternating forest. See Figure 5.3.9.

Definition 5.3.1 *Let $G = (V, E)$ be a simple graph and let M be a matching in G. Define X to be the set of vertices missed by M. An M-**alternating forest** is a subset F of E satisfying: F is a forest with $M \subset F$, each component of (V, F) contains either exactly one vertex in X or consists of one edge in M, and each path in F starting in X is M-alternating.*

For any M-alternating forest F, define

$$\text{even}(F) := \{v \in V \mid F \text{ contains an even-length } X - v \text{ path}\}$$

$$\text{odd}(F) := \{v \in V \mid F \text{ contains an odd-length } X - v \text{ path}\}$$

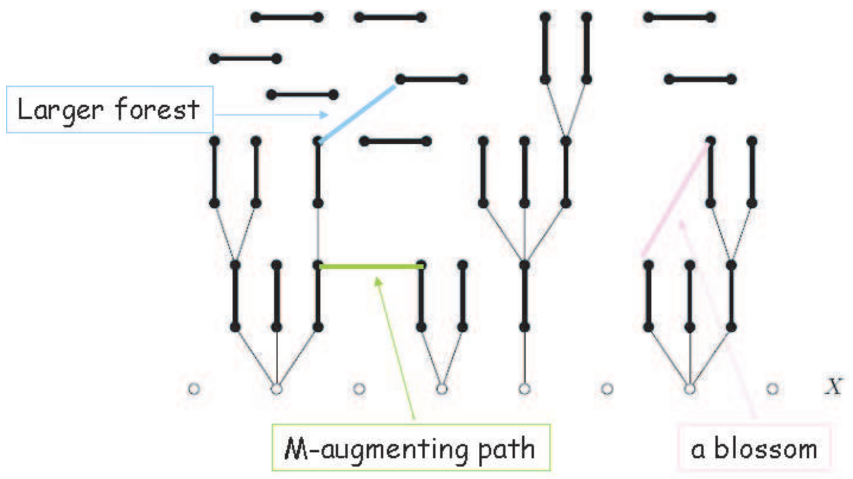


Figure 5.3.9: An M-alternating Forest

$$\text{free}(F) := \{v \in V \mid F \text{ contains no } X - v \text{ path}\}$$

Then each $u \in \text{odd}(F)$ is incident with a unique edge in $F \setminus M$ and a unique edge in M. We use the following algorithm to find an M-alternating forest. Initially, $F := M$. We can make progress in the following cases:

1. **Case 1:** $u \in$ **even**(F) **and** $v \in$ **free**(F). Add uv to F. We obtain a bigger forest.
2. **Case 2:** $u \in$ **even**(F) **and** $v \in$ **even**(F). Find the $X - u$ and $X - v$ paths P and Q in F.
   (a) **Case 2a:** P **and** Q **are disjoint.** Then P and Q form with uv an M-augmenting path. We can find a bigger matching.
   (b) **Case 2b:** P **and** Q **are not disjoint.** Then P and Q contain an M-blossom B. We can shrink the blossom to obtain a smaller graph.

Now suppose none of the above cases apply. This implies that there is no edges connecting even(F) and $v \in \text{even}(F) \cup \text{free}(F)$ (see Figure 5.3.10). We proceed to argue that the current matching is maximum, by constructing a set U in the Tutte-Berge formula. Indeed, if there is no such edge, even(F) is a stable set in $G - \text{odd}(F)$. Hence, by setting $U := \text{odd}(F)$, we have

$$o(G - U) \geq |\text{even}(F)| = |X| + |\text{odd}(F)| = (|V| - 2|M|) + |U|.$$

The first equality holds because each vertex in odd(F) has a unique neighbour in even(F) by following the unique edge in the matching, and so $|\text{odd}(F)| = |\text{even}(F)| - |X|$. The second inequality holds because we find a matching which matches every vertex except those in $|X|$, and so $|M| = \frac{1}{2}(|V| - |X|)$. Therefore, $U := \text{odd}(F)$ is a set that satisfies Tutte-Berge formula, which also shows that M is a maximum matching. This gives a constructive proof of Tutte-Berge theorem.

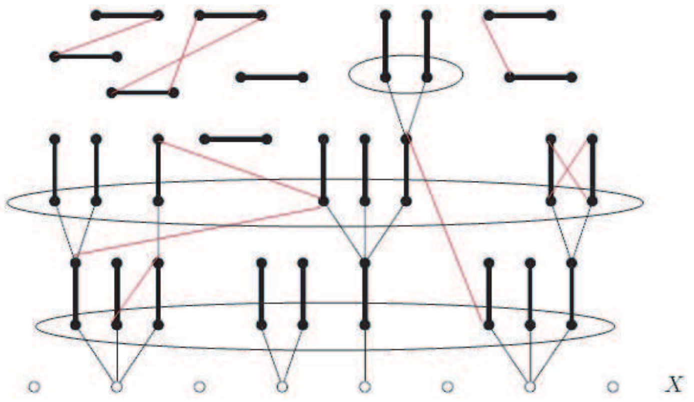


Figure 5.3.10: $U := \text{odd}(F)$ is a set that satisfies Tutte-Berge formula.

5.3.1 The Edmonds-Gallai Decomposition

In fact, the above construction can be used to obtain a lot of useful information about matchings. Let $G = (V, E)$ be a graph. The Edmonds-Gallai decomposition of G is the partition of V into $D(G)$, $A(G)$, and $C(G)$ defined as follows (recall that $N(U) := \{v \in V \backslash U | \exists u \in U : uv \in E\}$):

$D(G) := \{v \in V|$ there exists a maximum-size matching missing $v\}$,
$A(G) := N(D(G))$,
$C(G) := V \backslash (D(G) \cup A(G))$.

Theorem 5.3.2 *$U := A(G)$ attains the minimum in 5.2.3, $D(G)$ is the union of the odd components of $G - U$, and (hence) $C(G)$ is the union of the even components of $G - U$.*

5.4 The Chinese Postman Problem

Now we present a famous application of weighted general matching (that we have not studied yet). The Chinese Postman Problem is a typical example of a problem in discrete mathematics. It states as follows:

Problem: (Chinese Postman Problem) A postman has to travel along *every* road in a given road network to deliver letters. He starts from the post office and needs to return to the post office after the delivery. The problems is to find a shortest tour for the postman. ■

This can be easily translated into a graph problem. Given an undirected graph, a vertex v, a length $l(e)$ on every edge e, find a shortest tour to start from v, visit every edge at least once and come back to v.

Definition 5.4.1 *Circuit: a path that ends at the vertex at which it begins.*

Definition 5.4.2 *Eulerian circuit: a graph is said to be Eulerian if there is a route through it that starts and finishes at the same vertex, traversing each edge exactly once.*

Theorem 5.4.3 *A graph contains an Eulerian circuit if and only if it is connected and each vertex is of even degree.*

For the Chinese postman problem, ff every vertex in a network is of an even degree, then we can find a Eulerian tour and this is clearly optimal.

However, it may well be the case that the network we are dealing with does indeed contain odd degree vertices. In this circumstance, in order to visit all edges and return to the same start point, we may have to retrace some of the edges in order to traverse all edges at least once.

Let's say a postman finish his tour in such a network. We then add those repeated edges to the graph (if it repeats n times, we add n the same edges to the graph), then all the vertexes in the new graph is even, since the postman's trail in the new graph must forms an Eulerian circuit. So the problem reduces to finding the shortest edges to add to the network to make it Eulerian.

Definition 5.4.4 *Let the set of odd degree vertices be T. A T-join is a set of edges which have odd degree in T and even degree on $V - T$ (could be zero degree).*

So the problem now reduces to finding a minimum weight T-join. If there are only 2 odd vertices in the network, a minimum weighted T-join is simply a shortest path between the 2 odd vertices in G. This idea can be generalized. Given a set of odd degree vertices T, we construct a weighted complete graph with vertex set T, and the weight of each edge (u, v) in H is set to be the length of a shortest path between u and v in G. Now we compute a minimum weighted perfect matching M of H. We consider the union of the shortest paths for each $(u, v) \in M$. It can be shown that the union is a minimum weighted T-join. Therefore, we can add this T-join to G, and find an optimal tour for the postman.

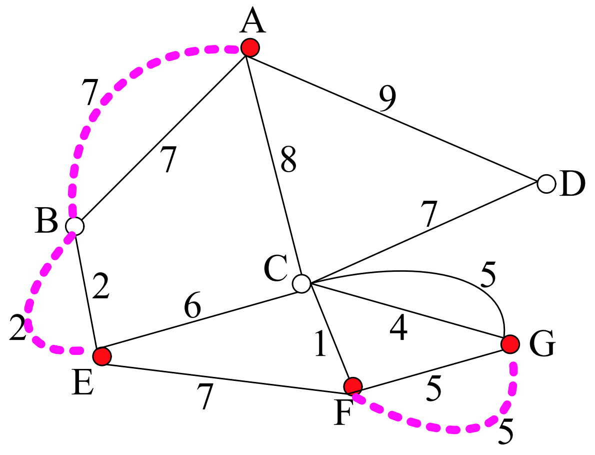


Figure 5.4.11: Perfect Matching in General Graph

Let's see an example in Figure 5.4.11:

1. First, we identify the odd vertices: A, E, F and G, indicated by the solid red dots in the above.
2. Next, we have to list all possible pairings of these odd degree vertices: $\{A, E\}$ and $\{F, G\}$; $\{A, F\}$ and $\{E, G\}$; $\{A, G\}$ and $\{E, F\}$.

3. For each of these three sets of pairings, we have to find the shortest distances between each pair of odd vertices. We may use Dijkstra's algorithm, but since it is a small network, we can compute the distance by inspection.

4. For $\{A, E\}$ and $\{F, G\}$, we have $(7 + 2) + 5 = 14$.

5. For $\{A, F\}$ and $\{E, G\}$, we have $(8 + 1) + (6 + 4) = 19$.

6. For $\{A, G\}$ and $\{E, F\}$, we have $(8 + 4) + 7 = 14$.

7. Comparing all possible sets of pairings, we find that $\{A, E\}$ and $\{F, G\}$ is the set of pairings that should be repeated in the solution. This gives the network shown in Figure 5.4.11. Where the pink dashed lines indicate the additional edges we add to the network to make it Eulerian.

8. Now the network is Eulerian, it's easy to find an Eulerian tour.

References

[1] Alexander Schrijver, *Combinatorial Optimization*, 290, Springer, 2003.

[2] Douglas West, *Introduction to Graph Theory*, Prentice Hall, 2000.