Chapter 2

Graph Matching

The Graph Matching problem often arises when we are concerned with "effective assignments", which optimally pair up objects based on certain pre-given requirements. For example, consider the following Optimal Course Assignment problem:

> Optimal Course Assignment
>
> Given a set of teachers T and a set of courses C, and a set S of pairs of the form (t_i, c_j) indicating that the teacher $t_i \in T$ can teach the course $c_j \in C$, find a subset A of S that contains the largest number of pairs in S such that each t_i in T and each c_j in C appears at most once in A. The subset A of S represents a course assignment in which (1) each teacher t_i teaches at most one course that the teacher t_i can teach; (2) each course c_j is taught by at most one teacher who can teach c_j, and (3) the assignment maximizes the number of courses that can get taught.

Similar problems arise in worker/job assignment, person/position decision, boy/girl engagement, and so forth.

These problems have been formulated into a general fundamental problem on graphs that has been widely studied. Given a graph G, a *matching* M in G is a subset of edges in G such that no two edges in M share a common end. Assume that the graph G is a weighted graph (i.e., each edge e of G is associated with a real number $wt(e)$ that is called the *weight* of the edge), then the *weight* of a matching M is defined to be the sum of the weights of the edges in M. For unweighted graphs, we simply assume the weigh of every edge is 1. The Graph Matching problem is to find a maximum matching in a given graph. Using our formulation, the Graph Matching problem is given as a 4-tuple.

GRAPH MATCHING $= \langle I_Q, S_Q, f_Q, opt_Q \rangle$, where

I_Q: the set of all weighted undirected graphs G

S_Q: $S_Q(G)$ is the set of all matchings in the graph G

f_Q: $f_Q(G, M)$ is the weight of the matching M in G

opt_Q: max

For example, the OPTIMAL COURSE ASSIGNMENT problem can be converted into the GRAPH MATCHING problem on unweighted graphs as follows. We let each teacher and each course be a vertex in the graph G. There is an edge between a teacher and a course if the teacher can teach the course. A solution to the GRAPH MATCHING problem on the graph G corresponds to an optimal course assignment for the school.

The GRAPH MATCHING problem has attracted attention because of its intuitive nature and its wide applicability. Its solution in the general case involves sophisticated and beautiful combinatorial mathematics. In this chapter, we will concentrate on analysis principles and fundamental algorithms for the problem. Further applications of the problem will be mentioned in later chapters.

Throughout this chapter, we will assume that n is the number of vertices and m is the number of edges in the considered graph. Also, we assume that the graphs in our discussions are weighted graphs. Unweighted graphs will be treated as weighted graphs in which the weight of all edges is 1.

2.1 Augmenting paths

Let M be a matching in a graph $G = (V, E)$. A vertex v is *matched* (with respect to the matching M) if v is an endpoint of an edge in M, otherwise, the vertex is *unmatched*.

Definition 2.1.1 Let M be a matching in a graph G. An *alternating path* (with respect to M) is a simple path $P = \langle u_0, u_1, \ldots u_h \rangle$ in G in which the edges go alternatively between edges in M and edges not in M (the starting edge $[u_0, u_1]$ can be either in M or not in M), such that either

(1) P is a cycle (i.e., $u_0 = u_h$) and h is an even number; or
(2) P is not a cycle. In this case, if the end-edge $[u_0, u_1]$ (resp. $[u_{h-1}, u_h]$) of P is not in M then the end-vertex u_0 (resp. u_h) is unmatched.

Recall that the *symmetric difference* of two sets A and B is defined as $A \Delta B = (A \setminus B) \cup (B \setminus A)$. In particular, if we also regard the alternating

path P with respect to the matching M in the graph G as a set of edges, then $P\Delta M$ is the set of edges obtained from M by removing all edges in $P \cap M$ then adding all edges in $P \setminus M$.

Definition 2.1.2 An alternating path P (w.r.t. M) is an *augmenting path* (w.r.t. M) if the *gain* $g(P)$ of P is positive, where the gain $g(P)$ of the path P is defined as $g(P) = \sum_{e \in P \setminus M} wt(e) - \sum_{e' \in P \cap M} wt(e')$.

Figure 2.1 shows a graph G and a matching M in G, where heavy lines $[v_1, v_4]$ and $[v_2, v_5]$ are edges in the matching M and light lines are edges not in M. The number near each edge is the weight of the edge. The paths $\langle v_6, v_2, v_5, v_1, v_4 \rangle$, $\langle v_2, v_5, v_1, v_4, v_2 \rangle$, and $\langle v_3, v_5, v_2, v_6 \rangle$ are all alternating paths, in which $\langle v_2, v_5, v_1, v_4, v_2 \rangle$ is also an augmenting path while the other two alternating paths are not augmenting paths.

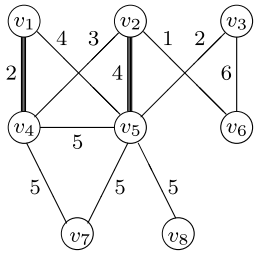

Figure 2.1: Alternating paths and augmenting paths

This is straightforward to see the following fact:

Lemma 2.1.1 *Let M be a matching in a graph G and let P be an alternating path w.r.t. M in G, then $P\Delta M$ is also a matching in the graph G.*

The following theorem serves as a fundamental theorem in the study of graph matching and graph matching algorithms.

Theorem 2.1.2 *Let G be a weighted graph and let M be a matching in G. M is of the maximum weight if and only if there is no augmenting path w.r.t. M in the graph G.*

PROOF. Suppose that there is an augmenting path P w.r.t. M in the graph G. By Lemma 2.1.1, $M' = P\Delta M$ is also a matching in G. Moreover, the weight $wt(M')$ of the matching M' is equal to $wt(M) + g(P)$, where the gain $g(P)$ of the augmenting path P, by definition, is positive. As a result, the

weight $wt(M')$ of the matching M' is strictly larger than the weight $wt(M)$ of M, and the matching M is not of the maximum weight.

Conversely, suppose that the matching M is not of the maximum weight. Let $M_{\max}$ a matching of the maximum weight. Then $wt(M_{\max}) > wt(M)$. Consider the graph $G_0 = M_{\max}\Delta M$ that consists of the edges in G that belong to exactly one of $M_{\max}$ and M. No vertex in G_0 has degree larger than 2. In fact, if a vertex v in G_0 had degree larger than 2, then at least two edges incident on v belong to either M or $M_{\max}$, contradicting the fact that both M and $M_{\max}$ are matchings in G. Therefore, each component of G_0 must be either a simple path, or a simple cycle. Note that each component of G_0 must be an alternating path w.r.t. M, whose edges go alternatively between edges in M (in fact, in $M \setminus M_{\max}$) and edges not in M (i.e., in $M_{\max} \setminus M$). Since $wt(M_{\max}) > wt(M)$, at least one C of the components of G_0 must satisfy $\sum_{e \in C \cap M_{\max}} wt(e) = \sum_{e \in C \setminus M} wt(e) > \sum_{e' \in C \cap M} wt(e')$. Thus, this component C makes an augmenting path w.r.t. M in G. This proves that if M is not of the maximum weight, then there is an augmenting path w.r.t. M in G, thus, completes the proof of the theorem. □

Based on Theorem 2.1.2, a general graph matching algorithm can be derived. The algorithm is given in Figure 2.2. Most algorithms for GRAPH MATCHING are essentially based on this basic algorithm. Naturally, to develop fast algorithms based on this framework, we will focus on reducing the number of times for the **while** loop in step 2 to be executed, and developing efficient algorithms for constructing the augmenting paths in step 2.1.

Algorithm. Matching
INPUT: a weighted graph $G = (V, E)$
OUTPUT: a matching M of the maximum weight in G

1. $M = \emptyset$;
2. **while** (there are augumenting paths w.r.t. M in G) **do**
2.1 construct a set A of vertex-disjoint augmenting paths;
2.2 $M = M\Delta A$.

Figure 2.2: General algorithm for graph matching

2.2 Matching in bipartite graphs

In this section, we present an algorithm for the GRAPH MATCHING problem on a special class of graphs, the *bipartite graphs*. We will explain how aug-

menting paths can be constructed for a matching in a bipartite graph. Then a maximum matching in the bipartite graph can be constructed using the algorithm **Matching** in Figure 2.2. Our discussion will be on unweighted bipartite graphs, which can be extended to weighted bipartite graphs without principle difficulties.

Definition 2.2.1 A graph $G = (V, E)$ is *bipartite* if the vertex set V of G can be partitioned into two disjoint subsets $V = V_1 \cup V_2$ such that every edge in G has one end in V_1 and the other end in V_2.

The bipartiteness of a graph can be tested using a standard graph traversing algorithm such as depth first search or breadth first search, which try to color the vertices of the given graph by two colors such that no two adjacent vertices are colored with the same color. Obviously, a graph is bipartite if and only if it can be colored in this way with two colors, which also implies that a bipartite graph G contains no cycles of odd length.

Let M be a matching in a bipartite graph G. The idea of constructing an augmenting path w.r.t. M is fairly natural: we pick each unmatched vertex v_0, and try to find an augmenting path starting from v_0. For this, we perform a process similar to *breadth first search*, starting from the vertex v_0. The vertices encountered in the search process are classified into *odd-level vertices* and *even-level vertices*, depending upon their distance to the vertex v_0 in the search tree, assuming that the vertex v_0 is at level 0. For an even-level vertex v, the process tries to extend the augmenting path by adding an edge not in M. The vertex v may be incident on several edges not in M and we do not know which is the one we want. Thus, we record all of them — just as in breadth first search we record all unvisited neighbors of the current vertex v. For an odd-level vertex w, the process either concludes with an augmenting path (when w is unmatched) or tries to extend the augmenting path by adding an edge in M (note that if w is a matched vertex then there is a unique edge in M that is incident on w). Note that in case of an odd-level vertex w, the search process is different from the standard breadth first search: the vertex w may have several unvisited neighbors, but we only record the one that matches w in M and ignore the others.

The drawback of the above process is that if the starting vertex v_0 is not an end of an augmenting path, then the process will fail and have to try other unmatched vertices. This wastes time. Instead, we will start from *all* unmatched vertices, and extend the paths from them in the manner as described above. However, once we find out that two of these paths are connected to make an augmenting path, we stop with the augmenting path.

Algorithm. Bip-Augment(G, M) \\M is a matching in bipartite graph G

1. $Q = \emptyset$; \\ Q is a queue
2. **for** (each vertex v) **do** $lev[v] = -1$;
3. **for** (each unmatched vertex v) **do** { $lev[v] = 0$; $Q \leftarrow v$; }
4. **while** ($Q \neq \emptyset$) **do**

   $u \leftarrow Q$;

4.1 **if** ($lev[u]$ is even)

   for (each edge $[u, w]$) **do**

4.1.1 **if** ($lev[w] = -1$) { $lev[w] = lev[u] + 1$; $dad[w] = u$; $Q \leftarrow w$; }

4.1.2 **else if** ($lev[w] = lev[u]$) an augmenting path is found; stop;

4.2. **else** \\ $lev[u]$ is odd

   let $[u, w]$ be the edge in M;

4.2.1 **if** ($lev[w] = -1$) { $lev[w] = lev[u] + 1$; $dad[w] = u$; $Q \leftarrow w$; }

4.2.2 **else** \\ $lev[w] = lev[u]$ an augmenting path is found; stop;

5. return('no augmenting path in G.').

Figure 2.3: Finding an augmenting path in a bipartite graph

A formal description of this search process is given in Figure 2.3.

According to the algorithm, each vertex v is assigned a level number $lev(v) \geq 0$ such that either v is an unmatched vertex and $lev(v) = 0$, or v has a vertex $dad(v)$ at level $lev(v) - 1$ as its parent. In particular, if $lev(v)$ is odd, then the edge $[dad(v), v]$ is an edge not in M while if $lev(v)$ is even then v is the unique child of its parent $dad(v)$ and the edge $[dad(v), v]$ is an edge in M. The level is also used to record whether a vertex v has been visited in the process. A vertex v is unvisited if and only if $lev(v) = -1$.

The algorithm **Bip-Augment** builds a leveled hierarchy H, starting from level 0 that consists of all unmatched vertices. Each vertex w at a level larger than 0 has a parent at level $lev[w] - 1$, given by $dad[w]$. Therefore, starting from a vertex w in the hierarchy H, we can trace by following the array $dad[*]$ a (unique) path from w to an unmatched vertex at level 0.

Theorem 2.2.1 *On a bipartite graph G and a matching M in G, the algorithm* **Bip-Augment** *stops at step 4.1.2 or 4.2.2 if and only if there is an augmenting path in G with respect to M.*

PROOF. We first show that for an odd-level vertex u, if we reach step 4.2.2, then we must have $lev[w] = lev[u]$, where $[u, w]$ is an edge in M. Because of

step 4.2.1, $lev[w] \neq -1$ if we are at step 4.2.2. Also note that $lev[w] \neq 0$ since w is a matched vertex. If $lev[w] > lev[u]$, then w must already have a parent $dad[w] \neq u$ at level $lev[u]$ and $[dad[w], w]$ is an edge in M, contradicting the assumption that $[u, w]$ is an edge in M. Now assume $lev[w] < lev[u]$. If $lev[w]$ is even then since w is matched, $lev[w] > 0$ so $[dad[w], w]$ is an edge in M and the vertex $dad[w]$ is at level $lev[w] - 1 < lev[u]$. If $lev[w]$ is odd then w is matched with its unique child at level $lev[w] + 1 < lev[u]$ (this is because both $lev[w]$ and $lev[u]$ are odd). Thus, in either case, we would get a contradiction that w is matched with a vertex that is not u. In consequence, we must have $lev[w] = lev[u]$ if we are at step 4.2.2.

Thus, if the algorithm **Bip-Augment** stops at step 4.1.2 or 4.2.2, we must encounter an edge $[u, w]$ with $lev[u] = lev[w]$. If $lev[u]$ is even, then the edge $[u, w]$ is not in M because u is either unmatched and at level 0 or matched with its parent at level $lev[u] - 1$. If $lev[u] = lev[w] = 0$, then the single edge $[u, w]$ is an augmenting path w.r.t. M. If $lev[u] = lev[w] > 0$, then both edges $[dad[u], u]$ and $[dad[w], w]$ are in M, and by following the array $dad[*]$, we can go from u and from w to get two paths P_u and P_w, respectively, that go alternatively between edges in M and edges not in M. Note that the paths P_u and P_w cannot share a common vertex v: otherwise, the two subpaths from u to v and from w to v, respectively, plus the edge $[u, w]$ would give a cycle of odd length in the graph G, contradicting the fact that the graph G is bipartite. Thus, the two paths P_u and P_w must be disjoint and ended at two different unmatched vertices at level 0. Now these two paths plus the edge $[u, w]$ give an augmenting path w.r.t. M.

On the other hand, if $lev[u]$ is odd, then the edge $[u, w]$ is in M, and both edges $[dad[u], u]$ and $[dad[w], w]$ are not in M. By following the array $dad[*]$, we can go from u and from w to get two disjoint paths P_u and P_w, respectively, that go alternatively between edges not in M and edges in M, and end at two different unmatched vertices at level 0 (here again the disjointness of the paths P_u and P_w is because of the bipartiteness of the graph G). These two paths plus the edge $[u, w]$ in M give an augmenting path w.r.t. M.

This proves that if the algorithm **Bip-Augment** stops at step 4.1.2 or 4.2.2, then there is an augmenting path in G w.r.t. M. Moreover, the above discussion explained how the augmenting path could be constructed using the array $dad[*]$ in this case.

Now we prove that if there is an augmenting path P w.r.t. M, then the algorithm **Bip-Augment** must stop at step 4.1.2 or step 4.2.2 (and construct an augmenting path). For this, we only need to prove that there is an augmenting path P that contains an edge $[u, w]$ for which the vertices u

and w get their level numbers assigned by the algorithm satisfying $lev[u] = lev[w]$ and the edge is caught at step 4.1.2 or step 4.2.2 by the algorithm.

Let $P = \langle v_1, v_2, \ldots, v_{2k} \rangle$ be a shortest augmenting path, i.e., an augmenting path with the fewest edges among all augmenting paths. So P starts and ends at unmatched vertices v_1 and v_{2k}, respectively, which are at level 0 in the algorithm. Now trace the path P in the leveled hierarchy H, starting from the vertex v_1 at level 0. Since the path P has to go back to the vertex v_{2k} that is also at level 0, there must be an edge $[v_i, v_{i+1}]$ in the path P such that $lev[v_i] \geq lev[v_{i+1}]$. Without loss of generality, assume that $[v_i, v_{i+1}]$ is the first such an edge in P. We show that we must have $lev[v_i] = lev[v_{i+1}]$.

If $[v_i, v_{i+1}]$ is an edge in M, then $lev[v_i]$ is odd. Note that if $lev[v_{i+1}] = -1$ at the time when we examine the edge $[v_i, v_{i+1}]$ at step 4.2 of the algorithm, then step 4.2.1 would set $lev[v_{i+1}] = lev[v_i]+1 > lev[v_i]$, contradicting our assumption that $lev[v_i] \geq lev[v_{i+1}]$. Thus, we must have $lev[v_{i+1}] \neq -1$. Now, as we have shown in the first paragraph in this proof, in this case, we will reach step 4.2.2 and the equality $lev[v_i] = lev[v_{i+1}]$ holds true.

The only case left is that $[v_i, v_{i+1}]$ is an edge not in M and $lev[v_i] > lev[v_{i+1}]$. In this case, $lev[v_i]$ is even and $lev[v_i] > 0$. The level number $lev[v_{i+1}]$ cannot be even: otherwise, we would have $lev[v_i] \geq lev[v_{i+1}] + 2$. In this case, when vertex v_{i+1} is examined at step 4.1, the edge $[v_{i+1}, v_i]$ would make the vertex v_i to have a level number bounded by $lev[v_{i+1}] + 1 < lev[v_i]$. Thus, $lev[v_{i+1}]$ must be odd. Let P' be the path from an unmatched vertex v' at level 0 to the vertex v_{i+1}, obtained by tracing the array $dad[*]$ from v_{i+1}. Then, because $lev[v_i] > lev[v_{i+1}]$, the path P' plus the path $\langle v_{i+1}, v_{i+2}, \ldots, v_{2k} \rangle$ would give a shorter augmenting path w.r.t. M, contradicting the assumption that P is a shortest augmenting path.

Therefore, we must have $lev[v_i] = lev[v_{i+1}]$. Moreover, as shown above, if $[v_i, v_{i+1}]$ is an edge in M, then $lev[v_i]$ is odd, so that the edge $[v_i, v_{i+1}]$ must be caught by step 4.2.2, and if $[v_i, v_{i+1}]$ is an edge not in M, then $lev[v_i]$ is even, and the edge $[v_i, v_{i+1}]$ is caught by step 4.1.2. In conclusion, if there is an augmenting path w.r.t. M, then an augmenting path will be constructed by step 4.1.2 or step 4.2.2 of the algorithm.

This completes the proof of the lemma. □

Based on Theorem 2.1.2, the algorithm **Matching** in Figure 2.2, the algorithm **Bip-Augment** in Figure 2.3, and Theorem 2.2.1, an algorithm can be developed for the GRAPH MATCHING problem on bipartite graphs, as given in the following theorem.

Theorem 2.2.2 *The* GRAPH MATCHING *problem on bipartite graphs can be solved in time* $O(nm)$.

PROOF. We can use an array $M[1..n]$ to represent a matching in the graph such that $M[u] = w$ if and only if $[u, w]$ is an edge in the matching M. Thus, checking whether an edge is in the matching, adding an edge to the matching, and deleting an edge from the matching can all be done in constant time.

The algorithm **Bip-Augment** processes each edge in the graph at most twice, once from each end of the edge, and the process on an edge takes constant time. Moreover, once the algorithm stops at step 4.1.2 or 4.2.2, the found augmenting path can be easily constructed by tracing the array $dad[*]$ from the two vertices u and w of the current edge $[u, w]$ to vertices at level 0, as explained in the proof of Theorem 2.2.1. Therefore, the running time of the algorithm **Bip-Augment** is bounded by $O(m)$.

Mow we consider the algorithm **Matching** in Figure 2.2, which solves the GRAPH MATCHING problem. Each execution of the **while** loop in step 4 of the algorithm **Matching** calls the algorithm **Bip-Augment** to find an augmenting path and constructs a larger matching for the graph G. Since a matching in a graph of n vertices contains no more than $n/2$ edges, we conclude that the algorithm **Matching** runs in time $O(nm)$ if it uses the algorithm **Bip-Augment** as a subroutine to find augmenting paths. By Theorem 2.1.2, the algorithm **Matching** solves the GRAPH MATCHING problem in time $O(nm)$. □

2.3 Matching in general unweighted graphs

We now get back to the GRAPH MATCHING problem on general graphs, i.e., graphs that are not necessarily bipartite. In this section, we will be focused on the GRAPH MATCHING problem on unweighted graphs. Thus, unless indicated explicitly, all graphs in the discussion of this section are assumed to be unweighted graphs. As we mentioned, unweighted graphs are treated as weighted graphs in which all edge have weight 1. As a result, the weight of a matching M in an unweighted graph G is equal to the number of edges in M, and an augmenting path w.r.t. M in G is of the form $P = \langle u_0, u_1, \ldots, u_{2h+1} \rangle$, consisting of an odd number of edges such that $u_0 \neq u_{2h+1}$ and both are unmatched vertices, the edges $[u_{2i-1}, u_{2i}]$ are in M for $i = 1, \ldots h$, and the edges $[u_{2i-2}, u_{2i-1}]$ are not in M, for $i = 1, \ldots h+1$. That is, an augmenting path P w.r.t. M in the unweighted graph G is not a cycle, and is an alternating path starting and ending at unmatched vertices.

As a result, the number of edges in $P \setminus M$ is exactly one larger than that in $P \cap M$, and the matching $M \Delta P$ has one more edge than the matching M.

For a set S of edges, we denote by $|S|$ the number of edges in S. In particular, for a matching M in an unweighted graph, $|M|$ is the weight of M, and for a path P, $|P|$ is the length of P.

Let $P_1, \ldots, P_r$ be a set of vertex-disjoint augmenting paths w.r.t. a maching M in an unweighted graph G. Then it is easy to see that $M \Delta P_1 \Delta \cdots \Delta P_r$ is also a matching in G with $|M| + r$ edges. Therefore, if in step 2.1 of the algorithm **Matching** in Figure 2.2, we are able to construct many vertex-disjoint augmenting paths, then we will result in a faster matching algorithm. In the following, we show that this is possible for unweighted graphs based on the idea of *shortest augmenting paths*, which are the augmenting paths whose length is the shortest over all augmenting paths.

Lemma 2.3.1 *Let M and M' be matchings in a graph G such that $|M| < |M'|$. Then the graph $M \Delta M'$ contains at least $|M'| - |M|$ vertex-disjoint augmenting paths w.r.t. M.*

PROOF. Let $C_1, \ldots, C_r$ be the components of $M \Delta M'$. As explained in the proof of Theorem 2.1.2, each of these components is either a cycle or a simple path. Each of these components either (1) contains the same number of edges in M and in M', or (2) has one more edge in M than that in M', or (3) has one more edge in M' than that in M. Since the matching M' has $|M'| - |M|$ more edges than M, at least $|M'| - |M|$ of the components $C_1, \ldots, C_r$ must be of the structure (3), and each of these components is an augmenting path w.r.t. M. Moreover, all these augmenting paths are vertex-disjoint. □

Lemma 2.3.1 implies an upper bound on the length of the shortest augmenting paths, as given in the following lemma:

Lemma 2.3.2 *Let M be a matching in a graph G and let M_0 be a maximum matching in G, $|M| < |M_0|$. Then the shortest augmenting path w.r.t. M has its length bounded by $2\lfloor |M|/(|M_0| - |M|) \rfloor + 1$.*

PROOF. Lemma 2.3.1 shows there are at least $|M_0| - |M|$ vertex-disjoint augmenting paths w.r.t. M. Since the matching M has $|M|$ edges, at least one of these $|M_0| - |M|$ augmenting paths contains no more than $\lfloor |M|/(|M_0| - |M|) \rfloor$ edges in M. As as result, the length of this augmenting path is bounded by $2\lfloor |M|/(|M_0| - |M|) \rfloor + 1$. □

The following lemma shows that if we repeatedly augment a matching using shortest augmenting paths, then the length of the augmenting paths will not decrease.

Lemma 2.3.3 *Let P_0 be a shortest augmenting path w.r.t. a matching M_0, and let P_1 be an augmenting path w.r.t. the matching $M_1 = M_0 \Delta P_0$, then $|P_1| \geq |P_0| + |P_0 \cap P_1|$. In particular, $|P_1| \geq |P_0|$.*

PROOF. Consider the matching $M_2 = M_1 \Delta P_1$. We have $|M_2| = |M_0|+2$. By Lemma 2.3.1 (and its proof), M_0 has two vertex-disjoint augmenting paths P_0' and P_0'' that are contained in $M_2 \Delta M_0$. Note that $M_2 \Delta M_0 = P_0 \Delta P_1$ (you may want to verify this equality). Thus,

$$2|P_0| \leq |P_0'| + |P_0''| \leq |M_2 \Delta M_0| = |P_0 \Delta P_1| = |P_0| + |P_1| - |P_0 \cap P_1|,$$

where the first inequality is because P_0 is a shortest augment path w.r.t. M_0, and the second inequality is because P_0' and P_0'' are vertex-disjoint paths in the graph $M_2 \Delta M_0$. From this, the lemma is derived. □

Consider the following procedure: for a given graph G, start with the trivial matching $M_0 = \emptyset$, and for each $i \geq 0$, construct a *shortest* augmenting path P_i w.r.t. M_i to get a larger matching $M_{i+1} = M_i \Delta P_i$. This procedure gives a sequence of paths, which stops when a maximum matching is achieve:

$$P_0, P_1, P_2, \ldots \tag{2.1}$$

By Lemma 2.3.3, we have $|P_i| \leq |P_{i+1}|$ for all $i \geq 0$.

Lemma 2.3.4 *For any two paths P_i and P_j in the sequence (2.1), if $|P_i| = |P_j|$, then the paths P_i and P_j are vertex-disjoint.*

PROOF. Assume to the contrary that there are paths P_i and P_j with $|P_i| = |P_j|$ that are not vertex-disjoint, where $i < j$. Without loss of generality, we pick such P_i and P_j so that the value $j - i$ is minimized.

If $j > i+1$, then by Lemma 2.3.3, $|P_i| = |P_{i+1}| = \cdots = |P_{j-1}| = |P_j|$, and each P_h with $i < h < j$ is vertex-disjoint with both P_i and P_j. Therefore, P_j is also an augmenting path w.r.t. $M_{i+1} = M_i \Delta P_i$. Thus, under the assumed conditions, P_j is always an augmenting path w.r.t. M_{i+1}. By Lemma 2.3.3, $|P_j| \geq |P_i| + |P_i \cap P_j|$. Since $|P_i| = |P_j|$, $P_i \cap P_j = \emptyset$. However, this would imply that P_i and P_j share no common vertex: every vertex v in the path P_i becomes matched in the matching $M_{i+1} = M_i \Delta P_i$ and is an end of an

edge e_v in $M_{i+1} \cap P_i$. Thus, if v is also in P_j, which is w.r.t. M_{i+1}, then the edge e_v must be also in P_j, which would contradict the fact $P_i \cap P_j = \emptyset$.

This completes the proof that paths P_i and P_j are vertex-disjoint. □

Now suppose that we divide the sequence (2.1) into segments:

$$s_1 = (P_0, \ldots, P_{h_1}), s_2 = (P_{h_1+1}, \ldots, P_{h_2}), \ldots, \tag{2.2}$$

such that each segment is a maximal subsequence of paths that have the same length. Then we have the following theorem.

Theorem 2.3.5 *Suppose that the size (i.e., the weight) of a maximum matching in the unweighted graph G is s. Then the number of segments in sequence (2.2) is bounded by $2\sqrt{s}+1$.*

PROOF. Let $r = \lfloor s - \sqrt{s} \rfloor$. Then for each $i \leq r$, $|M_i| \leq r$, so by Lemma 2.3.2, $|P_i| \leq 2r/(s-r)+1 \leq 2\sqrt{s}$. Note $|P_i|$ must be an odd number bounded by $2\sqrt{s}$. Thus, the first r paths in the sequence (2.1) can have at most $\sqrt{s}$ different path lengths, thus make at most $\sqrt{s}$ segments. On the other hand, because the maximum matching of the graph G contains s edges, the sequence (2.1) contains exactly s paths. Therefore, the last $s - r \leq \sqrt{s}+1$ paths in the sequence (2.1) can make at most $\sqrt{s}+1$ segments. In conclusion, the number of segments in the sequence (2.2) is bounded by $2\sqrt{s}+1$. □

By Lemma 2.3.4, the paths in each segment in the sequence (2.2) are vertex-disjoint. This fact enables the construction of all paths in each segment in parallel, as shown in the algorithm given in Figure 2.4, where "a maximal set A of vertex-disjoint shortest augmenting paths" is a set to which no further vertex-disjoint shortest augmenting path can be added.

Algorithm. U-Matching
INPUT: an unweighted graph G
OUTPUT: a maximum matching M in G

1. $M = \emptyset$;
2. **while** (there are augmenting paths w.r.t. M in G) **do**
   2.1 construct a maximal set A of vertex-disjoint shortest augmenting paths w.r.t. M;
   2.2 Let $M = M \Delta A$.

Figure 2.4: Maximum matching based on shortest augmenting paths

Let $A = \{P_1, \dots P_h\}$ be the maximal set of vertex-disjoint shortest augmenting paths w.r.t. M constructed in step 2.1 of the algorithm **U-Matching**. Since all paths in A are vertex-disjoint, each path P_i is also an augmenting path w.r.t. the matching $M_i = M\Delta P_1\Delta \cdots \Delta P_{i-1}$. By Lemma 2.3.3, P_i is also a shortest augmenting path w.r.t. M_i. Moreover, let P_{h+1} be a shortest augmenting path w.r.t. the matching $M\Delta A = M\Delta P_1\Delta \cdots \Delta P_h$. Then by Lemma 2.3.3, $|P_{h+1}| \geq |P_h|$. In fact, we must have $|P_{h+1}| > |P_h|$: if $|P_{h+1}| = |P_h|$, then by Lemma 2.3.4, P_{h+1} is vertex-disjoint with all paths in A thus is also a shortest augmenting path w.r.t. M, which would contradict the assumption that the set A is a maximal set of vertex-disjoint shortest augmenting paths w.r.t. M. This enables us to conclude that step 2.1 of the algorithm **U-Matching** constructs an entire segment in the sequence (2.2). By Theorem 2.3.5, the while-loop of step 2 of the algorithm **U-Matching** is executed at most $2\sqrt{s} + 1$ times, where s is the size of a maximum matching in the graph G, thus, $s \leq n/2$. As a result, if step 2.1 runs in time $O(\alpha(n))$, then the algorithm **U-Matching** solves the GRAPH MATCHING problem on unweighted graphs in time $O(\alpha(n)\sqrt{n})$.

Theorem 2.3.5 is due to Hopcroft and Karp [70], who applied the theorem and proposed the algorithm **U-Matching** on bipartite graphs. Essentially, finding a maximal set of vertex-disjoint shortest augmenting paths w.r.t. a maching M in a bipartite graph can be achieved by finding a flow that saturates all shortest paths in a flow-network in which all edges have capacity 1. Using Dinic's algorithm [35], such a flow can be constructed in time $O(m)$. Combining this result with Theorem 2.3.5 and algorithm **U-Matching**, we obtain an algorithm for GRAPH MATCHING on unweighted bipartite graphs that runs in time $O(m\sqrt{n})$. We will give a detailed description of this algorithm in the next chapter on maximum flow, after the discussion on Dinic's maximum flow algorithm on general flow-networks.

GRAPH MATCHING algorithms for general unweighted graphs turn out to be much more difficult. While constructing an augmenting path in a bipartite graph is rather straightforward and intuitive, constructing augmenting paths in a general graph will need to deal with a complicated graph structure called "blossom". Edmond [36] studied the properties of blossoms and proposed a polynomial-time algorithm for GRAPH MATCHING on general unweighted graphs. Much following-up work has been done on improving Edmond's algorithm (e.g., [47]). Eventually, Micali and Vazirani [99] were able to take advantage of Theorem 2.3.5 and algorithm **U-Matching**, and developed an $O(m)$-time algorithm that constructs a maximal set of vertex-disjoint shortest augmenting paths in a general unweighted graph (see also [119]). This implies that GRAPH MATCHING on general unweighted graphs

can also be solved in time $O(m\sqrt{n})$. We summarize the above discussion in the following theorem.

Theorem 2.3.6 *The* GRAPH MATCHING *problem on unweighted general graphs can be solved in time* $O(m\sqrt{n})$.

2.4 Matching in general weighted graphs

Now we consider the general GRAPH MATCHING problem on weighted graphs, which, for the convenience of discussion, will be named WEIGHTED MATCHING. Let G be an undirected and weighted graph in which the weight for each edge e is given by $wt(e)$. Let M be a matching in G. The *weight* of M is defined by $wt(M) = \sum_{e \in M} wt(e)$. The objective of WEIGHTED MATCHING is to construct a *maximum weighted matching* in G, i.e., a matching whose weight is maximized over all matchings in G. We will use the word "size" $|M|$ of a matching M to denote the number of edges in the matching, to distinguish it from the weight of the matching M.

Introducing weights in the matching problem makes the problem much harder. In particular, neither must a maximum weighted matching have the largest size nor must matchings of maximum size be maximum weighted. As a result, augmenting a matching with an augmenting path does not guarantee increasing of the matching size. Therefore, the number of times the **while**-loop of step 2 of the algorithm **Matching** in Figure 2.2 is executed now becomes harder to control. Moreover, Theorem 2.3.5 no longer holds true, thus, algorithm **U-Matching** is no longer applicable. Nevertheless, we will see in this section that the concepts such as augmenting paths and blossoms can still be carried over. Due to the space limit, we will only give a brief introduction to the theory and algorithms for the WEIGHTED MATCHING problem. For more thorough and detailed descriptions, readers are referred to more specialized literature [48, 91, 106, 116].

2.4.1 Theorems and algorithms

Without loss of generality, we can assume that the weighted graph G, which is an instance of the WEIGHTED MATCHING problem, has no edges of weight less than or equal to 0. In fact, any matching M in G minus the edges of nonpositive weight gives a matching in G whose weight is at least as large as $wt(M)$. Again we fix a weighted graph G, and let n and m be the number of vertices and the number of edges in G, respectively.

Definition 2.4.1 For each $k \geq 0$, a *k-matching* in a graph G is a matching of size k. If there are k-matchings in G, then denote by M_k a(ny) k-matching in G whose weight is the maximum over all k-matchings in G, and call M_k a *maximum k-matching*. If there is no k-matching in G, define $M_k = \emptyset$.

Therefore, for some k, the matching M_k is a maximum weighted matching in the graph G. We first characterize the index k such that M_k makes a maximum weighted matching in G.

Lemma 2.4.1 *If the index k satisfies $wt(M_k) \geq wt(M_{k-1})$ and $wt(M_k) \geq wt(M_{k+1})$, then M_k is a maximum weighted matching in the graph G.*

PROOF. Assume the opposite, that the matching M_k is not a maximum weighted matching in G. Let $M_{\max}$ be a maximum weighted matching in G. Consider the graph $G_0 = M_k \Delta M_{\max}$. As we explained in the proof for Theorem 2.1.2, each component of G_0 is either a simple path or a simple cycle, and in each connected component of G_0, the number of edges in M_k and the number of edges in $M_{\max}$ differ by at most 1.

Since $wt(M_{\max}) > wt(M_k)$, there must be one connected component C_0 in the graph G_0 such that

$$\sum_{e \in C_0 \cap M_k} wt(e) < \sum_{e \in C_0 \cap M_{\max}} wt(e)$$

Note that $M' = M_k \Delta C_0$ is also a matching in G and $wt(M') > wt(M_k)$. Now a contradiction is derived: (1) if C_0 contains the same number of edges in M_k and in $M_{\max}$, then the matching M' has exactly k edges, contradicting the assumption that M_k is a maximum k-matching in G; (2) if C_0 contains one more edge in M_k than in $M_{\max}$, then M' is a $(k-1)$-matching with $wt(M') > wt(M_k) \geq wt(M_{k-1})$, contradicting the assumption that M_{k-1} is a maximum $(k-1)$-matching in G; and (3) if C_0 contains one more edge in $M_{\max}$ than in M_k, then M' is a $(k+1)$-matching with $wt(M') > wt(M_k) \geq wt(M_{k+1})$, contradicting the assumption that M_{k+1} is a maximum $(k+1)$-matching in G. □

Thus, if we start with the trivial matching $M_0 = \emptyset$, construct a sequence

$$M_0, M_1, M_2, \ldots,$$

of matchings, and stop at the first M_i such that $wt(M_i) \geq wt(M_{i+1})$, then by Lemma 2.4.1, the matching M_i is a maximum weighted matching in the

graph G. Now the problem remaining is to construct a maximum $(k+1)$-matching M_{k+1} from a maximum k-matching M_k, for each $k \geq 0$. This is given in the following theorem. For this, we define a *maximum augmenting path* w.r.t. a matching M to be an augmenting path P such that its *differential weight* $dw_M(P)$, defined by $dw_M(P) = wt(P \setminus M) - wt(P \cap M)$, is the largest over all augmenting paths w.r.t. M.

Theorem 2.4.2 *Let $k \geq 0$, and suppose $wt(M_{k-1}) < wt(M_k)$. Let P be a maximum augmenting path w.r.t. M_k. Then $M' = M_k \Delta P$ is a maximum $(k+1)$-matching in G.*

PROOF. Since P is an augmenting path w.r.t. M_k, $dw_{M_k}(P) > 0$. Note that we must have $|P \setminus M_k| = |P \cap M_k| + 1$: if $|P \setminus M_k| = |P \cap M_k|$, then $M_k \Delta P$ would be a k-matching whose weight is larger than the maximum k-matching M_k; while if $|P \setminus M_k| = |P \cap M_k| - 1$, then $M_k \Delta P$ would be a $(k-1)$-matching whose weight is larger than the maximum $(k-1)$-matching M_{k-1}. Thus, $M' = M_k \Delta P$ is a $(k+1)$-matching. Moreover, the maximum $(k+1)$-matching M_{k+1} exists and its weight is larger than that of M_k.

Consider the graph $G_0 = M_k \Delta M_{k+1}$, in which each component is an alternating path w.r.t. M_k, (as well as w.r.t. M_{k+1}), which is either a simple path or a simple cycle. Since $|M_{k+1}| = |M_k| + 1$, there must be a component C_0 of G_0 such that $|C_0 \cap M_{k+1}| = |C_0 \cap M_k| + 1$. Let $G'_0 = G_0 \setminus C_0$. Note that G'_0 contains the same number edges in M_k and in M_{k+1}. That is, $|G'_0 \cap M_k| = |G'_0 \cap M_{k+1}|$.

We claim that $wt(G'_0 \cap M_k) = wt(G'_0 \cap M_{k+1})$. In fact, if $wt(G'_0 \cap M_k) < wt(G'_0 \cap M_{k+1})$, then $(G'_0 \cap M_{k+1}) \cup (C_0 \cap M_k) \cup (M_k \cap M_{k+1})$ would give a k-matching whose weight is larger than that of the maximum k-matching M_k, while if $wt(G'_0 \cap M_k) > wt(G'_0 \cap M_{k+1})$, then $(G'_0 \cap M_k) \cup (C_0 \cap M_{k+1}) \cup (M_k \cap M_{k+1})$ would give a $(k+1)$-matching whose weight is larger than that of the maximum $(k+1)$-matching M_{k+1}. As a consequence, we have

$$\begin{aligned} wt(M_{k+1}) - wt(M_k) &= wt(C_0 \cap M_{k+1}) - wt(C_0 \cap M_k) \\ &= wt(C_0 \setminus M_k) - wt(C_0 \cap M_k) > 0. \end{aligned}$$

Thus, C_0 is an augmenting path w.r.t. M_k. Since P is a maximum augmenting path w.r.t. M_k, we have

$$\begin{aligned} dw_{M_k}(P) &= wt(P \setminus M_k) - wt(P \cap M_k) \\ &\geq dw_{M_k}(C_0) = wt(C_0 \setminus M_k) - wt(C_0 \cap M_k) \\ &= wt(M_{k+1}) - wt(M_k). \end{aligned}$$

This immediately gives

$$wt(M') = wt(M_k \Delta P) = wt(M_k) + dw(P) \geq wt(M_{k+1}),$$

i.e., the matching $M' = M_k \Delta P$ is a maximum $(k+1)$-matching in G. □

Lemma 2.4.1 and Theorem 2.4.2 suggest the algorithm in Figure 2.5 for the WEIGHTED MATCHING problem. Theorem 2.4.2 guarantees the constructed matching M_k to be a maximum k-matching in G, and Lemma 2.4.1 ensures the matching M_k returned by the algorithm to be a maximum weighted matching in G. Also note that if there is no $(k+1)$-matching in the graph G, then by definition $M_{k+1} = \emptyset$, so $wt(M_k) \geq wt(M_{k+1})$, and the algorithm stops with the maximum weighted matching M_k.

Algorithm. W-Matching
INPUT: an undirected and weighted graph G
OUTPUT: a maximum weighted matching M in G

1. $M_0 = \emptyset$; $k = 0$;
2. **while** there is an augumenting path w.r.t M_k **do**
   find a maximum augmenting path P w.r.t. M_k;
   $M_{k+1} = M_k \Delta P$;
   if $wt(M_k) \geq wt(M_{k+1})$
   then return(M_k); \\ M_k is the maximum weighted matching
   else $k = k + 1$;

Figure 2.5: An algorithm for WEIGHTED MATCHING

By the algorithm **W-Matching**, the problem WEIGHTED MATCHING is reduced to the problem of finding a maximum augmenting path w.r.t. a maximum k-matching. If the graph G is bipartite, then finding such an augmenting path can be reduced to solving the SHORTEST PATH problem, using the well-known Dijkstra's algorithm [30]. However, finding a maximum augmenting path in a general graph is, though still possible [36], much harder because of, not surprisingly, the existence of the blossom structure.

The best algorithm for the WEIGHTED MATCHING problem is due to Gabow, with his recently published article [48], in which he showed how to construct a maximum augmenting path w.r.t. a maximum k-matching in time $O(m + n \log n)$. This algorithm, combined with the algorithm **W-Matching** in Figure 2.5, gives the following result.

Theorem 2.4.3 *The* WEIGHTED MATCHING *problem on general weighted graphs can be solved in time* $O(nm + n^2 \log n)$.

Theorem 2.4.3 is the best known bound for the WEIGHTED MATCHING problem on general weighted graphs.

2.4.2 Minimum perfect matchings

The WEIGHTED MATCHING problem has a number of interesting variations that have nice applications in computational optimization. In this subsection, we discuss one of these variations, which will be used in our later discussion.

Let M be a matching in a graph G. The matching M is *perfect* if every vertex in G is matched. In other words, the matching M contains exactly $n/2$ edges in G if G has n vertices. The MIN PERFECT MATCHING problem is to look for a *minimum* weighted perfect matching in a given weighted graph, formally defined as follows.

MIN PERFECT MATCHING $= \langle I_Q, S_Q, f_Q, opt_Q \rangle$

I_Q: the set of all undirected weighted graphs G

S_Q: $S_Q(G)$ is the set of all perfect matchings in the graph G

f_Q: $f_Q(G, M)$ is the weight $wt(M)$ of the matching M

opt_Q: min

Of course, not all graphs have perfect matchings. In particular, a graph of odd number of vertices has no perfect matchings. Therefore, the MIN PERFECT MATCHING problem is only defined on weighted graphs that have perfect matchings. By Theorem 2.3.6, whether a graph, regarded as an unweighted graph, has perfect matchings can be detected in time $O(m\sqrt{n})$. Thus, the validity of the problem instances can be checked in time $O(m\sqrt{n})$.

We show how the MIN PERFECT MATCHING problem is reduced to the WEIGHTED MATCHING problem. Let G be an instance of the MIN PERFECT MATCHING problem. For each edge e in G, let $wt(e)$ be the weight of e in G. We construct a new weighted graph G' as follows. Let w_0 be the maximum $|wt(e)|$ over all edges e of G (here $|wt(e)|$ is the absolute value of $wt(e)$). The graph G' has the same vertex set and edge set as G. For each edge e in G', the weight $wt'(e)$ of e in G' is equal to $wt'(e) = (m + n)w_0 - wt(e)$.

Lemma 2.4.4 *Let M' be a maximum weighted matching in the graph G'. Then the same set of edges in M' constitutes a minimum perfect matching in the graph G.*

PROOF. By our assumption, the graph G has perfect matchings. Thus, the number n of vertices of G is an even number.

First we show that any imperfect matching in G' has weight strictly less than that of any perfect matching in G'. Let M_i be an imperfect matching in G' and let M_p be a perfect matching G'. We have

$$\begin{aligned} wt'(M_i) &= \sum_{e\in M_i} wt'(e) = \sum_{e\in M_i} ((m+n)w_0 - wt(e)) \\ &\leq \sum_{e\in M_i} (m+n+1)w_0 \\ &\leq (\frac{n}{2}-1)(m+n+1)w_0, \end{aligned}$$

where the first inequality is because of $w_0 \geq |wt(e)|$ for all edges e, and the last inequality is because M_i is imperfect thus contains at most $\frac{n}{2}-1$ edges. On the other hand,

$$\begin{aligned} wt'(M_p) &= \sum_{e\in M_p} wt'(e) = \sum_{e\in M_p} ((m+n)w_0 - wt(e)) \\ &= \sum_{e\in M_p} ((m+n-1)w_0 + (w_0 - wt(e))) \\ &\geq \sum_{e\in M_p} (m+n-1)w_0 \\ &= \frac{n}{2}(m+n-1)w_0, \end{aligned}$$

where the inequality is because $w_0 - wt(e) \geq 0$ for all edges e. Since $\frac{n}{2}(m+n-1) > (\frac{n}{2}-1)(m+n+1)$, we derive that the perfect matching M_p has weight strictly larger than that of the imperfect matching M_i. In consequence, every maximum weighted matching in G' is a perfect matching.

Now for any perfect matching M_p in the graph G' we have

$$\begin{aligned} wt'(M_p) &= \sum_{e\in M_p} wt'(e) = \sum_{e\in M_p} ((m+n)w_0 - wt(e)) \\ &= \frac{n}{2}(m+n)w_0 - \sum_{e\in M_p} wt(e) \\ &= \frac{n}{2}(m+n)w_0 - wt(M_p). \end{aligned}$$

Thus, for any two perfect matchings M_p and M'_p in G', $wt'(M_p) \geq wt'(M'_p)$ if and only if $wt(M_p) \leq wt(M'_p)$. In conclusion, the maximum weighted

matching M' in the graph G' must constitute a minimum weighted perfect matching in the graph G. $\square$

Combining Theorem 2.4.3 and Lemma 2.4.4, we obtain

Theorem 2.4.5 *The* MIN PERFECT MATCHING *problem can be solved in time* $O(nm + n^2 \log n)$.