CSCE 629-601 Analysis of Algorithms

Fall 2022

Instructor: Dr. Jianer Chen
Office: PETR 428
Phone: (979) 845-4259
Email: chen@cse.tamu.edu
Office Hours: MWF 3:50pm–5:00pm

Teaching Assistant: Vaibhav Bajaj
Office: EABC 107B
Phone: (979) 739-2707
Email: vaibhavbajaj@tamu.edu
Office Hours: T; 2pm-3pm, TR: 4pm-5pm

Course Notes. Graph Matching

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. The Graph Matching problem is to find a maximum matching (i.e., a matching with the maximum number of edges) in a given graph.

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 Let M be a matching in a graph G. An *augmenting path* (with respect to M) is a simple path $P = \langle u_0, u_1, \ldots u_h\rangle$ in G, with u_0 and u_h being unmatched, and the edges going alternatively between edges not in M and edges in M.

By the definition, an augmenting path must start and end at edges not in M. Therefore, the length (i.e., the number of edges) of an augmenting path is always odd.

Figure 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 path $\langle v_6, v_2, v_5, v_1, v_4, v_7\rangle$ is an augmenting path. There are also shorter augmenting paths, such as the path $\langle v_6, v_3\rangle$ consisting of the single edge $[v_6, v_3]$ and the path $\langle v_8, v_5, v_2, v_6\rangle$.

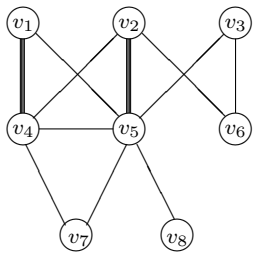

Figure 1: Matching and augmenting paths

Recall that the *symmetric difference* of two sets A and B is defined as $A\Delta B = (A\backslash B)\cup(B\backslash A)$. That is, $A\Delta B$ consists of the elements that belong to exactly one of A and B. If we regard an augmenting path P with respect to a matching M in the graph G as a set of edges, then $P\Delta M$

is the set of edges obtained from M by replacing all edges in $P \cap M$ with all edges in $P \setminus M$. Thus, the number of edges in $P\Delta M$ is 1 plus that in M.

This is straightforward to verify the following lemma.

Lemma 1 *Let M be a matching in a graph G and let P be an augmenting 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 *Let G be a graph and let M be a matching in G. The matching M is the maximum if and only if there is no augmenting path w.r.t. M in the graph G.*

PROOF. We prove the following equivalent statemet: *M is not a maximum matching if and only if there is an augmenting path w.r.t. M in the graph G.*

Suppose that there is an augmenting path P w.r.t. M in the graph G. By Lemma 1, $M' = P\Delta M$ is also a matching in G in which the number of edges is 1 plus that in M. Therefore, M is not a maximum matching.

Conversely, suppose that the matching M is not maximum. Let M_0 be a matching larger than M. Consider the graph $M_0\Delta M$ that consists of the edges in G that belong to exactly one of M_0 and M. No vertex in $M_0\Delta M$ has degree larger than 2. In fact, if a vertex v in $M_0\Delta M$ is incident to three edges, then at least two of the edges incident to v belong to the same matching, which is either M or M_0, contradicting the definition that a matching does not contain two edges sharing a common end. Therefore, each component C of $M_0\Delta M$ must be either a simple path, or a simple cycle. If the component C is a cycle, then the number of edges in $C \cap M$ and the number of edges in $C \cap M_0$ must be equal, because the cycle C must traverse the edges in M and M_0 alternatively and no two edges in a matching share a common end. If C is a simple path, then the number of edges in $C \cap M$ can be either larger than, or smaller than, or equal to the number of edges in $C \cap M_0$. However, since the matching M_0 is larger than the matching M, there are more edges in $(M_0\Delta M) \cap M_0$ than that in $(M_0\Delta M) \cap M$. This means that there must be at least one component C' of $M_0\Delta M$ that is a simple path with more edges in M_0 than that in M. Because the edges in C' must go alternatively between edges in M_0 and edges in M, the simple path C' must start and end with edges in M_0 (thus not in M). Therefore, C' must be an augmenting path w.r.t. M. This completes the proof of the theorem. □

Based on Theorem 2, a general graph matching algorithm can be derived. The algorithm is given in Figure 2. The time complexity of the algorithm depends on how efficiently we can construct an augmenting path for a given matching.

Algorithm. Matching
Input: a graph $G = (V, E)$
Output: a maximum matching M in G
1. $M = \emptyset$;
2. **while** (there is an augumenting path P w.r.t. M in G) **do**
$M = M\Delta P$.

Figure 2: General algorithm for graph matching

2 Matching in bipartite graphs

Although the GRAPH MATCHING problem on general graphs can be solved in polynomial time, the known algorithms for the problem are rather complicated and are out of the scope of the current notes. Instead, here we present an algorithm for the GRAPH MATCHING problem on a special class of graphs, the *bipartite graphs*. We will explain how augmenting paths can be constructed for a matching in a bipartite graph.

Definition 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.

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

Algorithm. Bip-Augment
Input: a bipartite graph G and a matching M in 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 3: Finding an augmenting path in a bipartite graph

Theorem 3 *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, the algorithm **Matching** in Figure 2, the algorithm **Bip-Augment** in Figure 3, and Theorem 3, an algorithm can be developed for the GRAPH MATCHING problem on bipartite graphs, as given in the following theorem.

Theorem 4 *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 3. Therefore, the running time of the algorithm **Bip-Augment** is bounded by $O(m)$.

Mow we consider the algorithm **Matching** in Figure 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, the algorithm **Matching** solves the GRAPH MATCHING problem in time $O(nm)$. □