Combinatorial Optimization Jan – May, 2025

Lecture 1: Bipartite Matching

Lecturer: T. Kavitha *Scribe: Soham Chatterjee*

A graph $G(V, E)$ is bipartite if the vertex set is partitioned into two sets $V = L \sqcup R$ and the edges are between the two partitions i.e. $E \subseteq L \times R$. Here we will look two main problems in Bipartite graphs: Maximum Matching and Minimum cost Perfect Matching.

1 Maximum Matching

Bipartite Maximum Matching
Input: Graph $G = (L \sqcup R, E)$
Question: Find a maximum matching $M \subseteq E$ of G

First we will solve finding maximum matching in bipartite graphs first. Then we will extend the algorithm to general graphs. We will

1.1 Using Max Flow

One approach to find a maximum matching is by using the max-flow algorithm. For this we introduce 2 new vertices s and t where there is an edge from s to every vertex in L and there is an edge from every vertex in R to t and all edges have capacity 1. Let the constructed graph is $G' = (V', E')$ where $V' = L \cup R \cup \{s, t\}$ and $E' = E \cup \{(s, v) : v \in L\} \cup \{(v, t) : v \in R\}$.

Then the max-flow for this directed graph is the maximum matching of the bipartite graph. In the following claim we will prove that this indeed gives the maximum matching.

Lemma 1.1.1

For a max-flow the flow through any edge is either 0 or 1.

Proof: The Edmonds-Karp algorithm takes a $s \rightsquigarrow t$ path in the residual graph and send the flow equal to the minimum of all the capacities of edges in that path. Since the capacities are all 1 the flow also equals to 1. Therefore at each iteration of Edmonds-Karp the amount of flow added is also integral. Therefore in the final max-flow the flow through each edge is integral. Now since the flow in any edge is always less than or equal to the capacity it is either 0 or 1. ■

Therefore the max-flow of the modified graph is always some non-negative integer. Now we have a lemma that value of max-flow gives a maximum matching.

Lemma 1.1.2

There exists a max-flow of value k in the modified graph $G' = (V', E')$ if and only there is a maximum matching of size k in $G'(L \cup R, E)$.

Proof: Suppose G' has a matching M of size k. Let $M = \{(u_i, v_i) : i \in [k]\}$ where $u_i \in L$ and $v_i \in R$ for all $i \in [k]$. Then we have the flow f, $f(s, u_i) = f(u_i, v_i) = f(v_i, t) = 1$ for all $i \in [k]$. This flow has value k. Now suppose the max-flow is more than k. Let the value of the max-flow is l, $l > k$. Since each edge has capacities 1 and by previous lemma each edge has integral flow there are l vertices in L which have positive flow from s. Then from each of these l vertices there is only one edge going to a vertex in R which has positive flow. Now it is not possible that from two vertices of L the flow goes to one vertex in R since for all edges joining vertices of R and t has capacity 1. Therefore from each of those vertices of L they goes to distinct l vertices of R. Therefore these l edges create a matching of G. So we have a matching which has size more than the maximum matching. Contradiction. Therefore the value of the max-flow is k.

Now suppose there is a max-flow f of value k. Since flow through each edge is integral by the similar argument as above we get a matching of size k. Now if M is a maximum matching which has size more than k, suppose $l > k$ then consider the flow f, $f(s, u_i) = f(u_i, v_i) = f(v_i, t) = 1$ for all $i \in [l]$ where $M = \{(u_i, v_i) : i \in [l]\}$ has flow of value l which is greater than the max-flow. Hence contradiction. Therefore the maximum matching is has size k. ■

Therefore from the max-flow if we take the edges from L to R which has positive flow they construct the maximum matching. So we have the following algorithm:

Algorithm 1: BP-Max-Matching-Flow

Input: $G = (L \cup R, E)$ bipartite graph
Output: Find a maximum matching

```
1  begin
2      V ⟵ A ∪ B ∪ {s, t}, E′ ⟵ E
3      for v ∈ L do
4          E′ ⟵ E′ ∪ {(s, v)}
5      for v ∈ R do
6          E′ ⟵ E′ ∪ {(v, t)}
7      for e ∈ E′ do
8          c_e ⟵ 1
9      f ⟵ Ford-Fulkerson(G′ = (V, E′), {c_e : e ∈ E′})
10     return {e : f(e) > 0, e ∈ E}
```

Therefore the algorithm successfully returns a maximum matching of the bipartite graph. But we don't know any algorithm for finding maximum matching in general graphs using max-flow. In the next algorithm we will use something called Augmenting paths to find a maximum matching which we will extend to general graphs.

1.2 Using Augmenting Paths

Definition 1.2.1: Alternating Path and Augmenting Path

In a graph $G = (V, E)$ and M be a matching in G. Then an M-alternating path is where the edges from M and $E \setminus M$ appear alternatively.

An M-alternating path between two unmatched (also called exposed) vertices is called an M-augmenting path.

Given a matching M and if there exists an M-augmenting path p then we can obtain a larger matching $M' = M \oplus p$. So if M is maximum matching then there is no augmenting path in G.

Theorem 1.2.1

A matching M is maximum if and only if there are no M-augmenting paths in G.

Proof: Suppose M is maximum. If there is an M-augmenting path p in G then $M \oplus p$ gives a matching with larger size. But that contradicts the fact that M is a maximum matching. Hence there are no M-augmenting paths in G.

For the other direction we will show that if M is not a maximum matching then there is an augmenting path. So let's assume that. Also assume that N be a maximum matching. Then $|N| > |M|$. Consider the graph $M \oplus N$. In the graph $M \oplus N$ every vertex has degree at most 2. Therefore the connected components of $M \oplus N$ are paths and cycles. Now since G is bipartite the cycles in $M \oplus N$ are of even length and the edges of M and N appears alternatively in the cycles. So for each cycle in $M \oplus N$ there are equal number of edges from M and edges from N. Now in the paths edges from M and N appears alternatively too. Therefore in an even path number of edges from M is equal to number of edges from N. And in a odd path either number of edges from N is one more than the number of edges from M or the opposite. Since we know $|N| > |M|$ there must exists at least one odd path p which has number of edges of N is one more than the number of edges from M. In that case the path starts and ends with edges from N. This path p is an M-augmenting path. Therefore there exists an M-augmenting path in G if M is not a maximum matching. ■

Now let M is not a maximum matching. Then we will find a M-augmenting path in G by constructing the *Hungarian Forest*. Our algorithm will be starting with empty set iteratively find augmenting paths and then take a symmetric difference with the matching set and continue like this till we can not find an augmenting path.

Algorithm 2: FIND-MAXIMUM-MATCHING

Input: $G = (L \cup R, E)$
Output: Find maximum matching $M \subseteq E$.

```
1 begin
2     M ⟵ ∅
3     while ∃ M-augmenting path do
4         p ⟵ M-augmenting path
5         M ⟵ M ⊕ p
6     return M
```

1.2.1 Construction of Hungarian Forest

In the algorithm we will find an M-augmenting path by constructing what is called a *Hungarian Forest.*

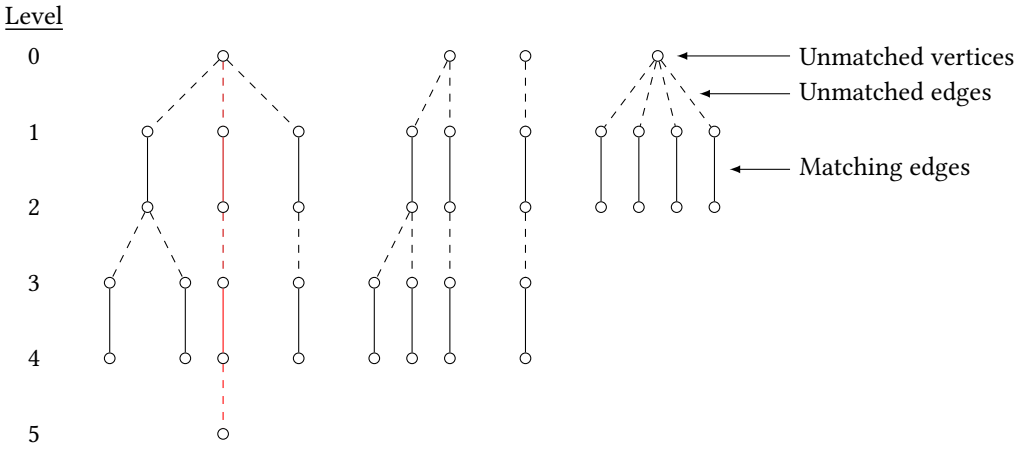

Figure 1: Hungarian Forest

We will start from each of unmatched vertices in L then we will start *Breadth-First-Search* where we will not repeat the vertices we have already visited and at odd level we will take matching edges and at even levels we will take unmatched edges. We stop when no new vertices can be found. Then we do the same for the unmatched vertices in R also. Now continuing like this starting at any unmatched vertex if we stop at odd level then we have already found a augmenting path and otherwise all the paths from unmatched vertices end at an even level. Lets call the forest F.

$\mathcal{O}_L$: Vertices of L that occur in odd levels
$\mathcal{E}_L$: Vertices of L that occur in odd levels
$\mathcal{U}_L$: Vertices of L that are unreachable
$\mathcal{O}_R$: Vertices of R that occur in odd levels
$\mathcal{E}_R$: Vertices of R that occur in even levels
$\mathcal{U}_R$: Vertices of R that are unreachable

Following the construction of Hungarian Forest we have the following observations:

Observation 1. *In the forest F there are no edges between vertices at levels separated by 2.*

Observation 2. *All even vertices except the vertices in level 0 are matched.*

1.2.2 Min Vertex Cover and Maximum Matching.

We have to show the algorithm always outputs a augmenting path. Instead of showing that we will show if the algorithm can not find an M-augmenting path then M is a maximum matching. We will show that using vertex cover.

Definition 1.2.2: Vertex Cover

$C \subseteq V$ is a vertex cover if every edge $e \in E$ has at least one end point in C

Lemma 1.2.2

For any matching M and any vertex cover C, $|M| \leq |C|$

Proof: Since for every edge $e \in E$, $e \cap C \neq \emptyset$ for all the edges in M at least one end point of each edge is in C therefore $|M| \leq |C|$. ■

Theorem 1.2.3 König-Egerváry, 1931

In a bipartite graph, the size of a maximum matching is equal to the size of minimum vertex cover.

Proof: Consider the set $C = O_L \cup O_R \cup \mathcal{U}_L$. Now $|C| = |O_L| + |O_R| + |\mathcal{U}_L|$. All the odd level vertices of $L \cup R$ are matched by the construction of Hungarian forest. And all the unreachable vertices of L are matched with unreachable vertices of R. Therefore $|O_L| + |O_R| + |\mathcal{U}_L| = |M|$. Hence if C is a vertex cover then C will be the minimum size vertex cover and M will be the maximum matching. We will show that this is a vertex cover with the following claim:

Claim 1.2.4

$O_L \cup O_R \cup \mathcal{U}_L$ is a vertex cover.

Proof: Now there is no edge in $\mathcal{E}_L \times \mathcal{E}_R$ otherwise it will make an M-augmenting path. There is also no edge in $\mathcal{E}_L \times \mathcal{U}_R$ otherwise $\mathcal{U}_R$ will not be unreachable. Hence all the other edges are incident on at least one of the three sets $O_L, O_R, \mathcal{U}_L$. So $O_L \sqcup O_R \sqcup \mathcal{U}_L$ is a vertex cover. ■

Therefore the minimum size vertex cover and the maximum matching has the same size. ■

Hence if the algorithm can not find an M-augmenting path we have shown that we obtain a minimum size vertex cover which has the same size as M which makes M to be the maximum matching. Hence the construction of Hungarian Forest always returns a M-augmenting path if M is not maximum matching.

Now in the algorithm construction of the Hungarian forest takes $O(|V| + |E|)$ time complexity. Therefore time to find an M-augmenting path in each iteration of the while loop takes $O(|V| + |E|)$ time. Now in each iteration the matching size increases by 1. So the while loop will go on for at most $O(|V|)$ iterations. Hence total time taken by the algorithm is $O(mn + n^2) = O(mn)$ where $m = |E|$ and $|V| = n$.

2 Minimum Cost Perfect Matching

Bipartite Min Cost Perfect Matching
Input: Graph $G = (L \sqcup R, E)$ with $|L| = |R|$ and cost function $c : E \to \mathbb{R}$.
Question: Find a perfect matching M with minimum cost $c(M) = \sum_{e \in M} c(e)$

WLOG we can always assume G is the complete bipartite graph. Since if its not complete then we can add those edges with their cost being ∞. So from now on we will assume G is a complete bipartite graph.

We will first write a integer program for this problem. Since the bipartite graph is complete we will take a $n \times n$

symbolic matrix X and cost function is also a $n \times n$ matrix C.

Integer Program:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j} c_{i,j} x_{i,j} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{i,j} = 1 \quad \forall\, i \in [n], \\
& \sum_{i=1}^{n} x_{i,j} = 1 \quad \forall\, j \in [n], \\
& x_{i,j} \in \{0,1\} \quad \forall\, i,j \in [n]
\end{aligned}$$

We will see the LP-relaxation of this by replacing the constraint $x_{i,j} \in \{0,1\}$ by $0 \leq x_{i,j} \leq 1$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j} c_{i,j} x_{i,j} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{i,j} = 1 \quad \forall\, i \in [n], \\
& \sum_{i=1}^{n} x_{i,j} = 1 \quad \forall\, j \in [n], \\
& x_{i,j} \geq 0 \quad \forall\, i,j \in [n]
\end{aligned}$$