CSS.309.1 Combinatorial Optimization

Instructor: Kavitha Telikepalli

TIFR 2025, Jan-May

Scribe: Soham Chatterjee

soham.chatterjee@tifr.res.in
Website: sohamch08.github.io

Contents

Chapter 1

Bipartite Matching

Definition 1.1: Bipartite Graph

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. $E \subseteq L \times R$.

In This chapter we will look into two main problems in Bipartite graphs: Maximum Matching and Minimum cost Perfect Matching.

1.1 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

1.1.1 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

```
begin
    V ⟵ A ∪ B ∪ {s, t}, E' ⟵ E
    for v ∈ L do
        E' ⟵ E' ∪ {(s, v)}
    for v ∈ R do
        E' ⟵ E' ∪ {(v, t)}
    for e ∈ E' do
        c_e ⟵ 1
    f ⟵ Ford-Fulkerson(G' = (V, E'), {c_e : e ∈ E'})
    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.

1.1.2 Using Augmenting Paths

Definition 1.1.1: 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 1.1.3 Berge's Lemma

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
```

1.1.2.1 Construction of Hungarian Forest

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

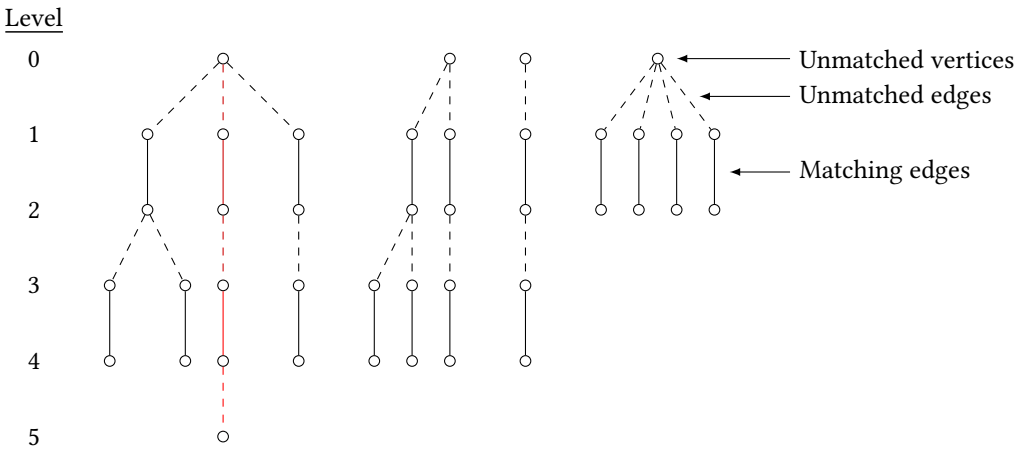


Figure 1.1: Hungarian Forest

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

Observation 1.2. *All even vertices except the vertices in level 0 are matched.*

1.1.2.2 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 1.1.2: Vertex Cover

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

Lemma 1.1.4

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 1.1.5 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 $C = \mathcal{O}_L \cup \mathcal{O}_R \cup \mathcal{U}_L$. Now $|C| = |\mathcal{O}_L| + |\mathcal{O}_R| + |\mathcal{U}_L|$. 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 $|\mathcal{O}_L| + |\mathcal{O}_R| + |\mathcal{U}_L| = |M|$. 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 1.1.6

$\mathcal{O}_L \cup \mathcal{O}_R \cup \mathcal{U}_L$ 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 $\mathcal{O}_L, \mathcal{O}_R, \mathcal{U}_L$. So $\mathcal{O}_L \sqcup \mathcal{O}_R \sqcup \mathcal{U}_L$ 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 $|V| = n$.

1.1.3 Hall's Theorem

We will use König-Egerváry Theorem to prove Hall's Marriage Theorem, which establishes a necessary and sufficient condition for the existence of a perfect matching in a bipartite graph.

Theorem 1.1.7 Hall's Theorem, 1935

A bipartite graph $G = (L \cup R, E)$ has an L-perfect matching if and only if for all $S \subseteq L$: $|N(S)| \geq |S|$

Proof: Suppose G has a L-perfect matching. Let M be the L-perfect matching. Then let $S \subseteq L$. Then $N(S) \supseteq T$ where $T := \{v \in R \colon (u, v) \in M, u \in S\}$. Then $|T| = |S|$. Therefore we have $|N(S)| \geq |T| = |S|$.

Now suppose for all $S \subseteq L$, $|S| \leq |N(S)|$. Suppose G doesn't have a perfect matching. Let M be a maximum L-matching in G. So $|M| < |L|$. Let C be the minimum vertex cover. By König-Egerváry Theorem $|C| = |M|$. Therefore $|C| < |L|$. Hence take $S = \{u \in L \colon u \notin C\}$ and take $T = R \cap C$. Therefore we have $N(S) \subseteq T$. Hence we have

$$|C| = |L \cap C| + |T| = |L| - |S| + |T| \implies |S| = |L| - |C| + |T| > |T| \geq |N(S)|$$

Hence we got a set $S \subseteq L$ for which $|S| > |N(S)|$. Hence contradiction ↯. Therefore G has a L-perfect matching. ■

1.2 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.

1.2.1 Constructing an LP

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 where $c_{i,j} = c(i, j)$ for the edge $(i, j) \in E$.

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 $0 \leq x_{i,j} \leq 1$.

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

Observation 1.3. *The first two constraints of the LP suggests that the matrix X is doubly stochastic.*

The feasible region of the LP-relaxation contains all possible doubly stochastic matrix with each entry being non-negative. The feasible region is a bounded polyhedron or polytope. We will call this polytope P. We are optimizing a linear constraint over a polytope the optimum will be attained at one of the "corners" or *extreme points*.

Definition 1.2.1: Extreme Point

$u \in Q$ is an extreme point of a set Q if u cannot be written as $\lambda x + (1-\lambda)y$ where $x, y \in Q$, $x \neq y$ and $\lambda \in (0,1)$.

1.2.2 Finding Extreme Point of the LP

We aim to show in the following theorem that the extreme points of the polytope P correspond to perfect matchings in a bipartite graph. Every perfect matching corresponds to a permutation matrix. Therefore specifically, we will prove that any doubly stochastic matrix is a convex combination of permutation matrices. This implies that every extreme point of P is a $0-1$ vector, corresponding to a permutation matrix.

Theorem 1.2.1 Birkhoff-Von Neumann Theorem, 1946

Every doubly stochastic matrix can be written as a convex combination of permutation matrices.

Proof: Suppose there exists values $u_i \; \forall \; i \in L$ and $v_j \; \forall \; j \in R$ such that

$$u_i + v_j \leq c_{i,j} \; \forall \; i \in L, j \in R$$

Then for any perfect matching M we have

$$\sum_{(i,j)\in M} c_{i,j} \geq \sum_{i \in L} u_i + \sum_{j \in R} v_j$$

Therefore $\sum\limits_{i\in L} u_i + \sum\limits_{j\in R} v_j$ gives a lower bound on the cost of minimum cost perfect matching for the bipartite graph. So to get the best lower bound we would like to maximize this quantity. So we have the following LP:

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in L} u_i + \sum_{j\in R} v_j \\ \text{subject to} \quad & u_i + v_j \leq c_{i,j} \quad \forall \; i \in L, j \in R \end{aligned}$$

Let D be the polyhedron generated by the constraint $u_i + v_j \leq c_{i,j} \; \forall \; i \in L, j \in R$. Now consider any $X \in P$. Then we have

$$\sum_{i\in L}\sum_{j\in R} c_{i,j}x_{i,j} \geq \sum_{i\in L}\sum_{j\in R}(u_i+v_j)x_{i,j} = \left(\sum_{i\in L}\sum_{j\in R} x_{i,j}\right) + \left(\sum_{j\in R} v_j \sum_{i\in L} x_{i,j}\right) = \sum_{i\in L} u_i + \sum_{j\in R} v_j$$

Therefore we have

$$\min_{\text{perfect matching } M} \sum_{(i,j)\in M} c_{i,j} \geq \min_{x\in P} \sum_{i\in L}\sum_{j\in R} c_{i,j}x_{i,j} \geq \max_{(u,v)\in D} \sum_{i\in L} u_i + \sum_{j\in R} v_j$$

Thus we get a primal-dual relation between the two LP problems.

Now our goal is to come up with a perfect matching M and $(u,v) \in D$ such that $\sum\limits_{(i,j)\in M} c_{i,j} = \sum\limits_{i\in L} u_i + \sum\limits_{j\in R} v_j$. Then M will be optimal solution to the LP-relaxation. Given solution u^*, v^* of the dual LP a perfect matching M would satisfy equality if it contains only the edges (i,j) such that $c_{i,j} = u_i + v_j$ by *complementary slackness*. But for a given (u,v) we may not be able to find a perfect matching among the edges with $c_{i,j} = u_i + v_j$.

We will describe an algorithm now which performs a series of iterations to obtain an appropriate u and v. It maintains a feasible solution for the dual problem and finds an "almost" primal feasible solution x satisfying complementary slackness. Satisfying complementary slackness we are working in the subgraph $G' = (V, E')$ where $E' = \{(i,j) \colon c_{i,j} = u_i + v_j\}$.

In the algorithm when M is not a perfect matching then it updates u, v to get a new feasible solution which has value $\sum\limits_{i\in L} u_i + \sum\limits_{j\in R} v_j$ greater than before which we will prove now and then runs the while loop again.

Algorithm 3: MIN-COST-BPM

Input: Complete bipartite graph $G = (L \sqcup R, E)$ with $|L| = n$ and cost function $c : E \to \mathbb{R}$
Output: Find minimum cost perfect matching

begin
  Initialize $u_i \longleftarrow 0 \; \forall \; i \in L$
  $v_j \longleftarrow \min_{i \in L} c_{i,j} \; \forall \; j \in R$
  willMatchingInc $\longleftarrow$ *True*
  while *True* **do**
    $E' \longleftarrow \{(i,j) : c_{i,j} = u_i + v_j\}$
    if *willMatchingInc* == *True* **then**
      $M \longleftarrow$ FIND-MAXIMUM-MATCHING$(G' = (V, E'))$
      if *M is perfect matching* **then**
        return *M*
    $(\hat{i}, \hat{j}) \longleftarrow \arg\min_{i \in \mathcal{E}_L} \min_{j \in \mathcal{E}_R \cup \mathcal{U}_R} (c_{i,j} - u_i - v_j)$
    if $(\hat{i}, \hat{j}) \in \mathcal{E}_L \times \mathcal{U}_R$ **then**
      willMatchingInc $\longleftarrow$ *False*
      Relabel vertices to obtain $O_L, O_R, \mathcal{E}_L, \mathcal{E}_R, \mathcal{U}_L, \mathcal{U}_R$
    $\delta \longleftarrow c_{(\hat{i},\hat{j})} - u_{\hat{i}} - v_{\hat{j}}$
    $u_i \longleftarrow u_i + \delta, \forall \; i \in \mathcal{E}_L$
    $v_j \longleftarrow v_j - \delta, \forall \; i \in O_R$

In the FIND-MAXIMUM-MATCHING algorithm if M is not a perfect matching then $C = O_L \cup O_R \cup \mathcal{U}_L$ is the minimum vertex cover of $G = (V, E')$. In order to update u, v to get u', v' we will use the information of the minimum vertex cover.

By the algorithm we obtain new u', v' where

$$u'_i = \begin{cases} u_i + \delta & \forall \; i \in \mathcal{E}_L \\ u_i & \forall \; i \in O_L \cup \mathcal{U}_L \end{cases} \qquad v'_j = \begin{cases} v_i - \delta & \forall \; i \in O_R \\ v_i & \forall \; i \in \mathcal{E}_R \cup \mathcal{U}_R \end{cases}$$

Claim 1.2.2

$(\hat{u}, \hat{v})$ is also a feasible solution of the LP

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in L} u_i + \sum_{j \in R} v_j \\ \text{subject to} \quad & u_i + v_j \leq c_{i,j} \quad \forall \; i \in L, j \in R \end{aligned}$$

with

$$\sum_{i \in L} \hat{u}_i + \sum_{j \in R} \hat{v}_j > \sum_{i \in L} u_i + \sum_{j \in R} v_j$$

Proof: The edges in $G' = (V, E')$ are in $O_L \times \mathcal{E}_R$, $\mathcal{E}_L \times O_R$ and $\mathcal{U}_L \times \mathcal{U}_R$. Therefore $\mathcal{E}_L \cup \mathcal{E}_R \cup \mathcal{U}_R$ is an independent set. Let after updating the vectors u, v we denote it by $\hat{u}, \hat{v}$. For edges in $(i,j) \in E \cap (O_L \times \mathcal{E}_R \cup \mathcal{U}_L \times \mathcal{U}_R)$ we have $\hat{u}_i = u_i$ and $\hat{v}_j = v_j$. Therefore

$$\hat{u}_i + \hat{v}_j = u_i + v_j \leq c_{i,j}$$

For edges in $(i,j) \in \mathcal{E}_L \times O_R$ we have $\hat{u}_i = u_i + \delta$ and $\hat{v}_j = v_j - \delta$. Therefore

$$\hat{u}_i + \hat{v}_j = u_i + \delta + v_j - \delta = u_i + v_j \leq c_{i,j}$$

Therefore $(\hat{u}, \hat{v})$ is a feasible solution of the LP.
Now we have

$$\sum_{i \in L} \hat{u}_i + \sum_{j \in R} \hat{v}_j - \sum_{i \in L} u_i - \sum_{j \in R} v_j = \delta(|\mathcal{E}_L| - |O_R|) = \delta(|L| - |C|) = \delta(n - |C|)$$

Since every vertex of O_R is matched and matched with vertices of $\mathcal{E}_L$ and $\mathcal{E}_L$ also contains the starting unmatched vertices of L we have $|\mathcal{E}_L| > |O_R|$. Hence the value strictly increases. ■

Claim 1.2.3

In each iteration of while loop size of the edge set E' increases by 1.

Proof: In any iteration of the while loop in the graph $G' = (V, E')$ none of the the edges in $\mathcal{E}_L \times (\mathcal{E}_R \cup \mathcal{U}_R)$ are present. Now in line 15 we are taking δ to be the minimum of $c_{i,j} - u_i - v_j$ over all edges of $\mathcal{E}_L \times (\mathcal{E}_R \cup \mathcal{U}_R)$. Let that edge is (i', j'). Then $c_{i',j'} - u_i - v_j = \delta$. Then in line 16 the algorithm updates $\hat{u}_{i'} = u_{i'} + \delta$ but $\hat{v}_{j'}$ is unchanged since $j' \in \mathcal{E}_R \cup \mathcal{U}_R$. Then in the next iteration of the while loop we have for the edge (i', j')

$$c_{i',j'} - \hat{u}_{i'} - \hat{v}_{j'} = c_{i',j'} - u_{i'} - \delta - v_{j'} = 0$$

Therefore the edge (i', j') is in the edge set constructed in line 6 in the next iteration of the while loop. The edges in $(\mathcal{E}_L \times O_R) \cup (O_L \times \mathcal{E}_R) \cup (\mathcal{U}_L \times \mathcal{U}_R)$ continue to remain tight after our update of u_i, v_j values. Since each time one edge is becoming tight the edge set is increasing by 1 in each iteration of the while loop. ■

Since each time one edge is becoming tight the edge set is increasing by one and henceforth the algorithm terminates. Therefore in after $O(n^2)$ iterations all the edges has been added and therefore the matching found is perfect.

Now if an edge of $\mathcal{E}_L \times \mathcal{U}_R$ becomes tight then the maximum matching continues to remain maximum matching. So only the labels of vertices are changed and new odd, even and unreachable sets are obtained. Therefore we don't need to run the algorithm for maximum matching again. So it takes $O(n^2)$ time for such iterations.

When an edge of $\mathcal{E}_L \times \mathcal{E}_R$ becomes tight then there exists an augmenting path and the size of the maximum matching increases. In this case only we run the Find-Maximum-Matching algorithm on the tight edges. In each such iterations the size of the maximum matching increases. Therefore there can be $O(n)$ iterations of this kind. Therefore in these iterations it takes $O(n^3)$.

Therefore the total time taken by the algorithm is $O(n^2) \times O(n^2) + O(n) \times O(n^3) = O(n^4)$. Now since carefully choosing the cost function we can make any extreme point of the polytope of LP be the optimum solution of the linear program this proves that all the extreme points are integral. ■

Therefore we have proved both that we can find the minimum cost perfect matching in a bipartite graph in $O(n^4)$ times and every doubly stochastic matrix can be written as a convex combination of permutation matrix.

Chapter 2

Matching in General Graphs

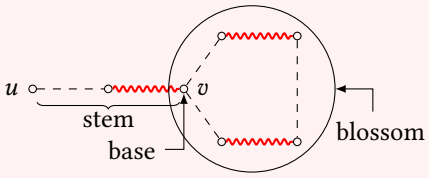
Here we give a similar algorithm for finding maximum matching in general graph as in the case of bipartite graphs in subsection 1.1.2. We will give a similar characterization for the maximum matching in general graphs. We have already showed Berge's Lemma that a matching M is maximum if and only if there are no M-augmenting paths in G. So we will search for augmenting paths in the graph G with respect to a matching M.

2.1 Flowers and Blossoms

By the above theorem like in the case of bipartite graphs we will search for augmenting paths in G for matching M and if we can find an augmenting path p we will update the matching by taking $M' = M \triangle p$ and obtain a larger matching. But unlike bipartite graphs we can not run the same algorithm for finding augmenting paths as there can be edges between two odd layers or two even layers. So in the M-alternating tree there can be odd cycles, but these odd cycles have all vertices except one vertex are matched using edges of the cycle. So we look for these special structures in the M-alternating tree called *blossom* and *flower*.

Definition 2.1.1: Flower and Blossom

For a matching M a *flower* consists of an even M-alternating path P from an exposed vertex u to vertex v, called the *stem* and an odd cycle containing v in which the edges alternate between in and out of the matching except for the two edges incident to v. This odd cycle is called the *blossom*.



Observation 2.1. *For a flower since the stem is an even augmenting path the base of the blossom is even as well ass all the other vertices of the blossom are even.*

Since blossoms are in the way of getting augmenting paths we want to remove the blossoms from the graph.

2.2 Shrinking Blossoms

In order to remove the blossoms from the graph we will shrink the blossoms into a single vertex every time we encounter a blossom while constructing the augmenting tree.

Question 2.1

How to shrink a blossom into a single vertex?

Let B be a blossom in G. Then the new graph is $G?B = (V', E')$ where

$$V' = (V \setminus B) \cup \{v_b\}, \qquad E' = \Big(E \setminus \{(u,v) : u \in B \text{ or } v \in B\}\Big) \cup \{(u, v_b) : u \notin B, v \in B, (u,v) \in E\}$$

So if M is a matching in G then we can also a get a matching M/B in G/B from M after shrinking B into a single vertex where $M/B = M \setminus \{\text{Matching edges in } B\}$.

Theorem 2.2.1

Let B be a blossom wrt M. M is a maximum matching in G if and only if M/B is a maximum matching in G/B.

Proof: ($\Longrightarrow$) : Suppose M/B is not maximum matching in G/B. Let N is a matching in G/B larger than M/B. Now if N has no edge incident to v_b then N is also a matching in B. Let N' is the matching in G. If there is an edge (u, v_b) incident on v_b in N then we can expand the blossom B and get the matching where the base of B is matched with u and other vertices of B are matched inside B. So we have the matching $N' = (N \setminus \{e\}) \cup \{(u, w)\}$ where w in B connected to u in G. Since $|N'| = |N| > |M/B|$ and B has $\frac{|B|-1}{2}$ matching edges we have $|N| + \frac{|B|-1}{2} > |M/B| + \frac{|B|-1}{2} = |M|$. But M is maximum matching in G. Hence, contradiction. Therefore, M/B is a maximum matching in G/B.

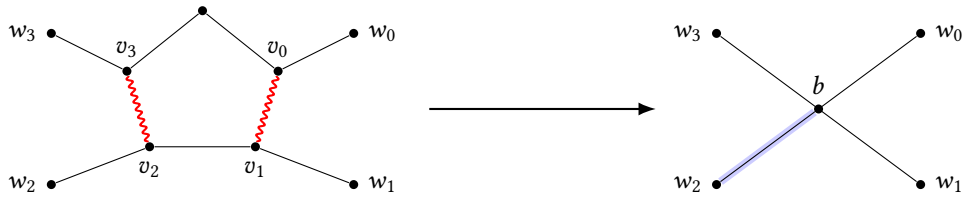
($\Longleftarrow$) : Suppose M is not a maximum matching in G. Now WLOG we can assume the blossom has an empty stem. Otherwise, if Q is the stem of B we can consider the matching $M' = M \oplus Q$ and $|M'| = |M|$. We now we will work with a matching which gives B empty stem, and we will call the matching M. This will make the base of the blossom B an exposed vertex. Now since M is not a maximum matching in G there exists an augmenting path $P : u \rightsquigarrow v$. Now if P has no vertex of B then P/B is also an augmenting path in G/B, but we assumed that M/B is a maximum matching in G/B. Hence, P must have a vertex of B. Let w be the first vertex of P in B. Then vertex v_b in G/B is unmatched. We remove the part $w \rightsquigarrow v$ from P. Let $P' = u \rightsquigarrow w$. Now if w is the base of B then P' is an augmenting path, and it is also an augmenting path of G/B which is not possible. So w is not the base of the B.

The last edge of P' is matched then it is also an edge of B. Then w is not the first vertex of P since the other end of the last of P' is before w. If the last edge of P' is not matched then P' is already an odd length alternating path from an exposed vertex. Inside B we can find an even length alternating path from w to the base of B where the edge incident on w is matched edge. Let that path is $\hat{P}$. Now consider the path $\tilde{P} = P' + \hat{P}$. It is an augmenting path from u to the base of B. Now since v_b is unmatched in G/B, $\tilde{P}/B$ is also an augmenting path in G/B. But this contradicts the assumption that M/B is a maximum matching in G/B. Then P can not exist. Hence, M is a maximum matching in G. ■

Question 2.2

If we find a maximum matching $M*$ in G/B and then let $N = M^* \cup (\text{Matching edges in } B)$, is N a maximum matching in G?

The answer is no. Consider the following example



$M = \{(u_0, u_1),\ (u_2, u_3)\}$

$N = \{(w_2, b)\}$

$\Downarrow$

$N' = \{(w_2, u_2),\ (u_3, u_4),\ (u_0, u_1)\}$

N' is not maximum matching. Since $\{(v_i, w_i) \mid i \in \{0, 1, 2, 3\}\}$ is maximum matching.

Note:-

The above is not contradicting the theorem since the blossom with respect to N' is not the blossom with respect to M.

2.3 Algorithm for Maximum Matching

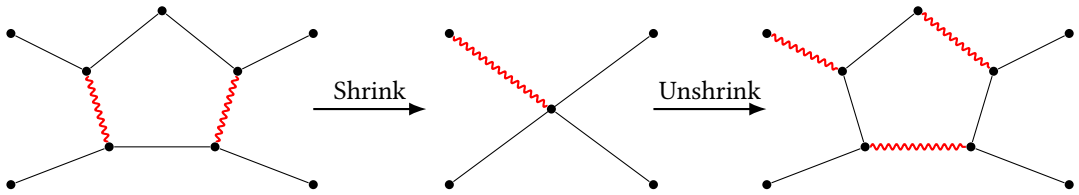
Suppose we start with a matching M in G. We mark all the exposed vertices to be even and keep all the other vertices unmarked at this point. Hence, initially all the vertices are marked even. Now we use the same algorithm for finding augmenting paths as in the case of bipartite graphs but with slight modifications. In the case of bipartite graphs at each iteration we created the M-alternating tree we went one level at a time. But in the case of general graphs we'll go two levels at a time. So at each iteration we start with a vertex which is labeled even. Let u be a vertex labeled. Now for each neighbor v of u:

Case 1: v is unmarked. This implies v is matched. Then mark v as odd and $M(v)$ i.e. the vertex matched with v as even and continue the algorithm.

Case 2: v is marked, and it is in the same tree as u. Then we have a blossom B with respect to M. We shrink the blossom B and therefore, we have the matching M/B in the graph G/B. Mark the shrunk vertex v_b even and continue the algorithm in the graph G/B with the matching M/B.

If we get an even cycle then we just ignore i.e. ignore any neighbor marked odd.

Case 3: v is marked even, and it is in a different tree from u. Let r_u and r_v are the root exposed vertices in the tree of u or v respectively. Then consider the path $P : r_u \rightsquigarrow u \rightarrow v \rightsquigarrow r_v$. This is an augmenting path from r_u to r_v. Hence, the algorithm found an augmenting path with respect to M. Now unshrink the blossoms in P to get the alternating path in G. Let that path is $\hat{P}$. So the algorithm updates the matching M to $M \oplus \hat{P}$, and then we will start the algorithm again with the new matching $M \oplus \hat{P}$.



Time Complexity: The algorithm performs at most n augmentations. Between two augmentations, it will shrink a blossom at most $\frac{n}{2}$ times as each shrinking reduces the number of vertices by at least 2. The time to construct the alternating tree is at most $O(n+m)$. Hence, the total time taken by the algorithm is $O(n^2(n+m)) = O(n^2m)$.

2.4 Tutte-Berge Theorem

Theorem 2.4.1

In any graph $G = (V, E)$ for any matching M in G and any $S \subseteq V$,

$$|M| \leq \frac{|V| + |S| - Odd(G - S)}{2}$$

where $Odd(G - S)$ is the number of odd components in $G - S$.

Proof: If $|S| > Odd(G - S)$ then we already have this since $|M| \leq \frac{|V|}{2}$. So let $|S| \leq Odd(G - S)$. Now each odd size component has at least one vertex unmatched in that component. So if that vertex is matched it is matched with a vertex in S. So M leaves at least $Odd(G - S) - |S|$ vertices unmatched. Hence, at most all the rest $|V| - (Odd(G - S) - |S|)$ vertices are matched in G. Therefore, $|M| \leq \frac{|V|+|S|-Odd(G-S)}{2}$. ■

Now the algorithm stops if none of the cases 1, 2, or 3 happens. Let $G' = (V', E')$ is the final graph after shrinking all the blossoms algorithm encountered in its runtime. And let M' is the matching in G' after the algorithm stops. Now we will show that M' is a maximum matching in G'.

Lemma 2.4.2

When none of the 3 cases holds the matching M' is maximum in G'.

Proof: We will show that M' attains equality in Theorem 2.4.1 for some subset S of vertices. Since the algorithm stops the M'-alternating tree in G' has no blossoms. So the M'-alternating tree is a forest. The algorithm marked the vertices of G' as even or odd. Take S be the set of all odd marked vertices. Hence, all the components of $G' - S$ are odd components where each component contains single vertex which is labeled even. So $\text{Odd}(G - S) = |V'| - |S|$. Therefore, $\frac{|V'|+|S|-Odd(G-S)}{2} = \frac{|V'|+S|-(|V|-|S|)}{2} = |S|$. Since all the odd vertices are matched with even vertices in G' we have $|S| = |M'|$. Hence, $|M'| = \frac{|V|+|S|-Odd(G-S)}{2}$. Therefore, M' is a maximum matching in G'. ■

Let the algorithm performs k blossom shrinking. Let $B_1, \ldots, B_k$ are the blossoms. And let M_i be the corresponding matching. $i = 0$ corresponds to the original graph. Let $G_i = (V_i, E_i)$ be the graph after i^{th} blossom shrinking. So $G_0 = G$ and G_k is the final graph after the algorithm stops. The above lemma shows that M_k is a maximum matching in G_k. We will show that if we unshrink the blossoms one at a time in the reverse order of shrinking then we will get a maximum matching.

Lemma 2.4.3

If M_k is a maximum matching in G_k. Then M_{k-1} is a maximum matching in G_{k-1}.

Proof: G_k is obtained from G_{k-1} by shrinking the blossom B_k. So $G_k = G_{k-1}/B_k$, $M_k = M_{k-1}/B_k$. So $|V_{k-1}| = |V_k| + |B_k| - 1$ and $|M_{k-1}| = |M_k| + \frac{1}{2}(|B_k| - 1)$. Let S be the set of odd vertices in G_k. Now while unshrinking the blossom B_k we add an even number of vertices $(|B_k| - 1)$ to one of the connected components to one of the connected components of $G_k - S$ and all these vertices are marked even. So the set of odd vertices of G_{k-1} are the same as set of odd vertices in G_k. Hence, $Odd(G_k - S) = Odd(G_{k-1} - S)$. Therefore,

$$\frac{|V_{k-1}| + |S| - Odd(G_{k-1} - S)}{2} = \frac{|V_k| + |B_k| - 1 + |S| - Odd(G_k - S)}{2} = |M_k| + \frac{|B_k| - 1}{2} = |M_{k-1}|$$

Therefore M_{k-1} is a maximum matching in G_{k-1}. ■

Using the same S we can show that if M_{i+1} is a maximum matching in G_{i+1} then M_i is a maximum matching in G_i. Hence, we can conclude that if M_k is a maximum matching in G_k then M_0 is a maximum matching in G. Therefore, after unshrinking all the blossoms in the reverse order of shrinking we get a maximum matching in G. Therefore, the above algorithm returns a maximum matching of G.

Also, we have shown that the maximum matching attains equality in Theorem 2.4.1 for the set of odd vertices S. Hence, we have the following theorem.

Theorem 2.4.4 Tutte-Berge Theorem

For any graph $G = (V, E)$,

$$\max_{M \text{ matching in } G} |M| = \min_{S \subseteq V} \frac{|V| + |S| - Odd(G - S)}{2}$$

where $Odd(G - S)$ is the number of odd components in $G - S$.

Now from the Tutte-Berge Theorem we conclude that a graph has a perfect matching if and only if for every $S \subseteq V$, the number of odd components in $G - S$ is at most $|S|$. Hence, we have the following corollary.

Corollary 2.4.5 Tutte's Matching Theorem

For any graph $G = (V, E)$, G has a perfect matching if and only if for every $S \subseteq V$, $Odd(G - S) \leq |S|$.

Chapter 3

Linear Programming

Chapter 4

Matroids