CSS.201.1 Algorithms

Instructor: Umang Bhaskar

TIFR 2024, Aug-Nov

Scribe: Soham Chatterjee

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

Contents

Chapter 1

Matching

In **??** we saw how to find a maximal matching in a graph using matroids. Here we will try to find maximum matching.

MAXIMUM MATCHING
Input: Graph $G = (V, E)$
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.

1.1 Bipartite Matching

So in this section we will study the following problem:

BIPARTITE MAXIMUM MATCHING
Input: Graph $G = (L \cup R, E)$
Question: Find a maximum matching $M \subseteq E$ of G

1.1.1 Using Max Flow

One approach to find a maximum matching is by using 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. Then the max-flow for this directed graph is the maximum matching of the bipartite graph. So we have the algorithm:

Algorithm 1: BP-MAX-MATCHING-FLOW

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

begin
  $V \longleftarrow A \cup B \cup \{s, t\}$
  $E' \longleftarrow E$
  for $v \in L$ **do**
    $E' \longleftarrow E' \cup \{(s, v)\}$
  for $v \in R$ **do**
    $E' \longleftarrow E' \cup \{(v, t)\}$
  for $e \in E'$ **do**
    $c_e \longleftarrow 1$
  $f \longleftarrow$ EDMONDS-KARP$(G' = (V, E'), \{c_e : e \in E'\})$
  return $\{e : f(e) > 0, e \in E\}$

Lemma 1.1.1

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

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 there is a flow f of value k. Since each edge has capacity 1 then either an edge has flow 1 or it has 0 flow. Since value of flow is k there are exactly k edges outgoing from s with positive flow. Let the edges are (s, u_i) for $i \in [k]$. Now from each u_i there is exactly one edge going out which has positive flow. Now if $\exists\ i \neq j \in [k]$ such that $\exists\ v \in R$, $f(u_i, v) = (u_j, v) = 1$ then $f(v, t) = 2$ but $c_{v,t} = 1$. So this is not possible. Therefore, the edges going out from each u_i goes to distinct vertices. These edges now form a matching of size k. ■

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: M-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 augmenting path.

Given a matching M and if there exists an augmenting path p then we can obtain a larger matching M' just by taking the edges in p not in M. Now suppose we are given a bipartite graph $G = (L \cup R, E)$. Let M is a matching in G. Suppose M is a maximum matching. If there exists an augmenting path p then we can obtain a larger matching just by taking the edges in p not in M. This contradicts with M is maximum matching. Hence, there are no augmenting paths.

Now we will show that given G and M which is not maximum then we can find an augmenting path with an algorithm. Since M is not maximum there is a vertex v which is not matched

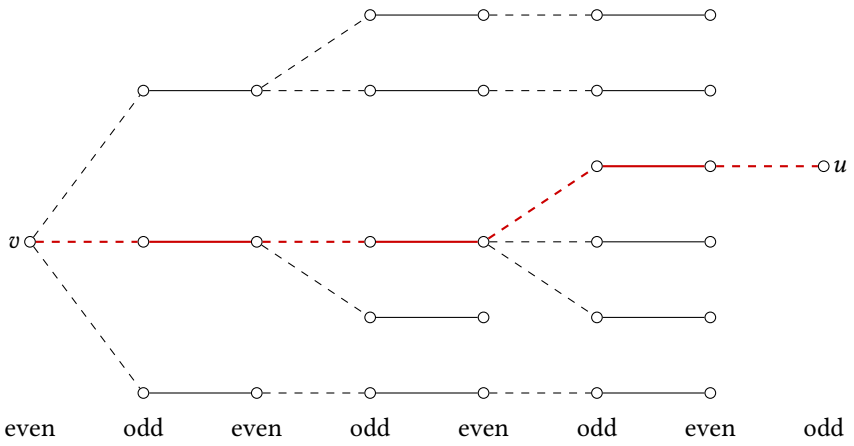
Algorithm 2: Find-Augmenting-Path(G, v)

Input: $G = (L \cup R, E)$ bipartite graph, matching M (not maximum) and an exposed vertex v
Output: Find an augmenting path starting from v

```
1 begin
2     v.mark ⟵ even
3     for u ∈ L ∪ R \ {v} do
4         u.mark ⟵ Null
5     Queue Q                          // For BFS
6     Enqueue(Q, v)
7     while Q not empty do
8         AUTree(Q)
9     return Fail
```

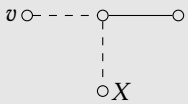
Algorithm 3: AUTree(Q)

```
1  u ⟵ Dequeue(Q)
2  if u.mark == even then
3      for (u, w) ∈ E \ M do
4          if w.mark == Null then
5              Enqueue(Q, w)
6              w.mark ⟵ odd
7              w.p ⟵ u
8  if u.mark == odd then
9      if ∃ (u, w) ∈ M and w.mark == Null then
10         w.mark ⟵ even
11         w.p ⟵ u
12         Enqueue(Q, w)
13     else
14         Print "v ⇝ u augmenting path found"
```



The above algorithm in each iteration checks if the new vertex has mark NULL before adding to the queue. Because of this we are not adding same vertex more than one into the queue and if we follow the parent and child pointers, this forms a tree. We call this tree to be an M-alternating tree. Denote the tree by T.

Note:-



The algorithm may not visit all the vertices in $L \cup R$ in the tree. For example in case of the graph at left the algorithm will not find the vertex

Since the algorithm runs a BFS if there was an edge between two vertices at levels separated by 2 we would have explored that vertex earlier. So our first observation is:

Observation 1.1. *In the tree T there are no edges between vertices at levels separated by 2.*

Observation 1.2. *All even vertices except v are matched in T.*

Observation 1.3. *There are no edges between two odd levels or even levels.*

Lemma 1.1.2

If leaf u is odd there is a $v \rightsquigarrow u$ augmenting path.

Proof: If the odd vertex u is unmatched then clearly there is a $v \rightsquigarrow u$ augmenting path. So let's assume u is matched. Say $(u, w) \in M$. If w is not in T then u can not be a leaf as the algorithm will take the edge $(u, w) \in M$ for next iteration.

So suppose w is in T. Then $w.mark = even$ since otherwise we would have taken then (w, u) edge in T before. But by Observation 1.2 all the even vertices except v are matched in the tree already. So u can not be matched with w ■

Now from the tree T we partition the vertices of T into the even marked vertices and odd marked vertices. So let $L_T = L \cap T$ and $R_T = R \cap T$. Therefore, L_T is the set of even marked vertices and R_T is the set of odd marked vertices.

Lemma 1.1.3

$N(L_T) = R_T$

Proof: Vertices in L_T are even vertices from which we explore all the edges not in M. Also, all the even vertices except v are matched. So except v for all the vertices in L_T their parent is the matched vertex. Hence, for all even vertices except v all the neighbors are in R_T. Since v is exposed v has no matched neighbor. So all the neighbors of v are also in R_T. Therefore, $N(L_T) = R_T$. ■

Lemma 1.1.4

Suppose we start the algorithm from an exposed vertex v. Suppose there is no augmenting path from v and let the tree formed by the algorithm is T. Then $|L_T| = |R_T| + 1$.

Proof: Since there is no augmenting path the graph all the leaves of T are even vertices. Otherwise, the leaves are odd vertices and then all of them have to be matched. If not then there will exists an augmenting path. Therefore, all the leaves of T are even vertices. Now since the vertices in L_T are even vertices and all even vertices except v are matched to unique odd vertex in R_T we have $|L_T| = |R_T| + 1$. ■

Now suppose M is a matching. Let $L' = \{v_1, \ldots, v_k\} \subseteq L$ are unmatched vertices. Therefore, $|M| = |L| - k$. Then consider the following algorithm:

- Let T_1 be M-alternating tree from v_1 by Find-Augmenting-Path(G, v_1). L_{T_1}, R_{T_1} are vertices of T_1.
- Let T_2 be M-alternating tree from v_2 by Find-Augmenting-Path$(G \setminus T_1, v_2)$. L_{T_2}, R_{T_2} are vertices of T_2.
- Let T_3 be M-alternating tree from v_3 by Find-Augmenting-Path$(G \setminus (T_1 \cup T_2), v_3)$. L_{T_3}, R_{T_3} are vertices of T_3. $\cdots$
- Let T_k be M-alternating tree from v_k by Find-Augmenting-Path$\left(G \setminus \left(\bigcup_{i=1}^{k-1} T_i\right), v_k\right)$. L_{T_k}, R_{T_k} are vertices of T_k.

Observation 1.4. *v_i is not in T_j for any $j < i$ because otherwise we would have found an augmenting path in T_j.*

Now L_{T_i} for all $i \in [k]$ are disjoint and R_{T_i} for all $i \in [k]$ are disjoint. If G had no augmenting path from v_i for all $i \in [k]$ then there are no augmenting paths in $G \setminus \left(\bigcup_{i=1}^{j} T_i\right)$ for all $j \in [k-1]$ from v_{j+1}. Therefore, by Lemma 1.1.4 we have $|L_{T_i}| = |R_{T_i}| + 1 \; \forall \; i \in [k]$. Hence, we have

$$\sum_{i=1}^{k} |L_{T_i}| = \sum_{i=1}^{K} (|R_{T_i}| + 1) \implies \left|\bigcup_{i=1}^{k} L_{T_i}\right| = \left|\bigcup_{i=1}^{k} R_{T_i}\right| + k$$

Now by Lemma 1.1.3, $N(L_{T_{j+1}}) = R_{T_{j+1}}$ for all $j \in [k-1]$ in $G \setminus \left(\bigcup_{i=1}^{j} T_i\right)$. Hence,

$$N(L_{T_j}) \subseteq \bigcup_{i=1}^{j} R_{T_i} \implies N\left(\bigcup_{i=1}^{k} L_{T_i}\right) = \bigcup_{i=1}^{k} R_{T_i}$$

But $\left|\bigcup_{i=1}^{k} L_{T_i}\right| = \left|\bigcup_{i=1}^{k} R_{T_i}\right| + k$. Therefore, any matching of $\bigcup_{i=1}^{k} L_{T_i}$ must leave at least k vertices unmatched. Now all the vertices in $L \setminus \left(\bigcup_{i=1}^{k} L_{T_i}\right)$ with $R \setminus \left(\bigcup_{i=1}^{k} R_{T_i}\right)$ and vice versa. Therefore, any matching of L must leave at least k vertices unmatched. Since M is a matching such that exactly k vertices are unmatched. M is a maximum matching. Therefore, if there is no augmenting path in G then M is a maximum matching.

We also showed before that if M is a maximum matching then there is no augmenting path in G. Therefore, we have the following theorem:

Theorem 1.1.5 Berge's Theorem

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

Therefore, if we start with any matching and each time we find an augmenting path we update the matching by taking the odd edges in the augmenting path and obtain a larger matching. After continuously doing this once when there is no augmenting path we can conclude that we obtained a maximum matching.

Since every time the size of the maximal matching is increased by at least 1. The total number of iterations the algorithm takes to output the maximal matching is $O(n)$ where n is the number of vertices in G. In each iteration it calls the Find-Augmenting-Path algorithm which takes the time same as time taken in BFS. Hence, Find-Augmenting-Path takes $O(m+n)$ time. Therefore, the BP-Maximum-Matching algorithm takes $O(n(n+m))$ time.

Algorithm 4: BP-MAXIMUM-MATCHING(G)

```
Input: G = (L ∪ R, E) bipartite graph
Output: Find a maximum matching
begin
    M ⟵ ∅
    while True do
        v ⟵ unmatched vertex
        p ⟵ FIND-AUGMENTING-PATH
        if p == FAIL then
            return M
        for e ∈ p do
            if e ∈ M then
                M ⟵ M \ {e}
            else
                M ⟵ M ∪ {e}
```

1.1.3 Using Matrix Scaling

Here we will show a new algorithm for deciding if a bipartite graph has a perfect matching using matrix scaling. The paper which we will follow is [LSW98]

BIPARTITE PERFECT MATCHING
Input: Graph $G = (L \cup R, E)$
Question: Decide if G has a perfect matching or not.

Suppose $G = (L \cup R, E)$ a bipartite graph. If bipartite adjacency matrix of the graph G is A then the permanent of the matrix A,

$$per(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} x_{i,\sigma(i)}$$

counts the number of perfect matchings in G. So we want to check if for a given bipartite graph $(L \cup R, E)$, $per(A) > 0$ or not where A is the bipartite adjacency matrix. Now there is a necessary and sufficient condition for existence of perfect matching in a bipartite graph which is called Hall's condition.

Theorem 1.1.6 Hall's Condition

A bipartite graph $G = (L \cup R, E)$ has an L-perfect matching if and only if $\forall\, S \subseteq L$, $|S| \leq |N(S)|$ where $N(S) = \{v \in R \colon \exists\, u \in L,\ (u, v) \in E\}$

Proof: Now if G has an L-perfect matching then for every $S \subseteq L$, S is matched with some $T \subseteq R$ such that $|S| = |T|$. Therefore, $T \subseteq N(S) \implies |S| = |T| \leq |N(S)|$.

Now we will prove the opposite direction. Suppose for all $S \subseteq L$ we have $|S| \leq |N(S)|$. Assume there is no L-perfect matching in G. Let M be a maximum L-matching in G. Let $u \in L$ is unmatched. Now consider the following sets:

$$X = \{x \in L \colon \exists\, M\text{-alternating path from } u \text{ to } x\}, \qquad Y = \{y \in R \colon \exists\, M\text{-alternating path from } u \text{ to } y\}$$

Now notice that $N(X) \subseteq Y$. Since in a M-alternating path from u whenever the odd edges are not matching edges and the even edges are matching edges. So in the odd edges we can pick any neighbor except the one it is matched with and the immediate even edge before that connects that vertex with the vertex in R it is matched with. Hence, we have $N(X) \subseteq Y$.

Now it suffices to prove that $|X| > |Y|$. Now let $y \in Y$. Suppose $u \rightsquigarrow x' \to y$ be the M-alternating path. If y is not matched then we could increase the matching by taking the odd edges of the path and thus obtain a matching with larger size than M. But M is maximum matching. Hence, y is matched. Therefore, we can extend the path by taking the matching edge incident on y and go the vertex $x'' \in L$ i.e. the new M-alternating path becomes $u \rightsquigarrow x' \to y \to x''$ to have an M-alternating path $u \rightsquigarrow x''$. So $|X| > |Y|$.

Therefore, we obtained a set of vertices $X \subseteq Y$ such that $|X| > |Y| \geq N(X)|$. This contradicts the assumption. Hence, contradiction. Therefore, G has an L-perfect matching. ∎

We will use hall's condition on the adjacency matrix to check if $per(A)$ is positive or not. Now multiplying a row or a column of a matrix by some constant c also multiplies the permanent of the matrix by c as well. In fact if $d_1, d_2 \in \mathbb{R}_+^n$ and $D_1 = diag(d_1)$ and $D_2 = diag(D_2)$ then $per(D_1AD_2) = \left(\prod_{i=1}^{n} d_{1_i}\right)\left(\prod_{i=1}^{n} d_{2_i}\right) per(A)$. So we can scale our original matrix A to obtain a different matrix B and from B we can approximate $per(A)$ by approximating $per(B)$. A natural strategy is to seek an efficient algorithm for scaling A to a doubly stochastic B.

Definition 1.1.2: Doubly Stochastic Matrix

A matrix $M \in \mathbb{R}^{m\times m}$ is doubly stochastic if entries are non-negative and each row and column sum to 1.

First we will show that Hall's Condition holds for doubly stochastic matrix. First let's see what it means for a matrix to satisfy hall's condition. A matrix with all entries non-negative holds Hall's Condition if for all $S \subseteq [n]$ if $T = \{i \in [n] : \exists\ j \in S,\ A(i,j) \neq 0\}$ then $|T| \geq |S|$. This also corresponds to the bipartite adjacency matrix satisfying the hall's condition since for any set of rows S the number of columns for which in the S rows at least one entry is non-zero should be greater than or equal to $|S|$.

Lemma 1.1.7

Hall's Condition holds for doubly stochastic matrix.

Proof: Let M be the doubly stochastic matrix. Let $S \subseteq [n]$. So consider the $|S| \times n$ matrix which only consists of the rows in S. Call this matrix M_S^r. Now suppose T be the set of columns in M_S^r which has nonzero entries. Now consider the $n \times |T|$ matrix which only consists of the columns in T. Call this matrix M_T^c. Now since M is doubly stochastic we know sum of entries of M_S^r is $|S|$ and sum of entries of M_T^c is $|T|$. Our goal is to show $|S| \leq |T|$. Now since T is the only set of columns which have nonzero columns in M_S^r the elements which contributes to the sum of entries in M_S^r are in the T columns in M_S^r. Since these elements are also present in M_T^c we have $|T| \geq |S|$. ∎

Hence, for doubly stochastic matrices the permanent is positive. Now not all matrices are doubly stochastic. And in fact matrices with permanent zero will not be doubly stochastic, so no amount of scaling will make it doubly stochastic. So we will settle for approximately doubly stochastic matrix. In order to make a matrix doubly stochastic first for each row we will divide the row with their row some. Now it becomes row stochastic. Then if it's not approximately doubly stochastic for each column we will divide the column entries with their column sum. But first what ϵ-approximate doubly stochastic matrix means.

Definition 1.1.3: ϵ−Approximate Doubly Stochastic Matrix

A matrix is ϵ−approximate doubly stochastic if for each column, the column sum is in $(1-\epsilon, 1+\epsilon)$ and for each row, the row sum is in $(1-\epsilon, 1+\epsilon)$

Now we will show that even for ϵ-approximate doubly stochastic matrix the hall's condition holds.

Lemma 1.1.8

Halls's Condition holds for ϵ-approximate doubly stochastic matrix for $\epsilon < \frac{1}{10n}$

Proof: Let M is ϵ−approximate doubly stochastic matrix. Let $S \subseteq [n]$. So consider the $|S| \times n$ matrix which only consists of the rows in S. Call this matrix M_S^r. Now suppose T be the set of columns in M_S^r which has nonzero entries. Now consider the $n \times |T|$ matrix which only consists of the columns in T. Call this matrix M_T^c. Now the sum of entries in M_S^r is $\geq |S|(1-\epsilon)$ and sum of entries in M_T^c is $\leq |T|(1-\epsilon)$. Now since T is the only set of columns which have nonzero columns in M_S^r the elements which contributes to the sum of entries in M_S^r are in the T columns in M_S^r. Since these elements are also present in M_T^c we have $||T|(1+\epsilon) \geq |S|(1-\epsilon)$. Therefore we have

$$|T| \geq |S|\frac{1-\epsilon}{1+\epsilon} = |S|\left(1 - \frac{2\epsilon}{1+\epsilon}\right) \geq |S|(1-2\epsilon) > |S|\left(1 - \frac{1}{5n}\right) \geq |S|\left(1 - \frac{1}{|S|}\right) > |S| - 1$$

Since T is an integer we have $|T| \geq |S|$. Hence the Hall's condition holds. ∎

Therefore, permanent of ϵ-approximate doubly stochastic matrix is also positive. Hence, our algorithm for bipartite perfect matching is:

Algorithm 5: BP-MATRIX-SCALING

Input: Bipartite adjacency matrix A of $G = (L \cup R, E)$
Output: Decide if G has a perfect matching.

```
begin
    while True do
        A ⟵ Scale every row of A to make it row stochastic.
        if All column-sums are in (1 − ϵ, 1 + ϵ) then
            return Yes
        A ⟵ Scale every column of A to make it column stochastic.
        if All row-sums are in (1 − ϵ, 1 + ϵ) then
            return Yes
```

In both if conditions we are checking if the matrix is $\epsilon-$approximate doubly stochastic matrix. The moment it becomes a $\epsilon-$approximate doubly stochastic matrix we are done.

Now if G doesn't have a perfect matching then we will never reach a ϵ-approximate doubly stochastic matrix since otherwise Hall's condition will hold, and then we will have that the permanent is positive. So if G doesn't have a perfect matching the algorithm will run in an infinite loop. We only need to check if G has a perfect matching the algorithm returns Yes.

We will now define a potential function $\Phi\colon \mathbb{Z}_0 \to \mathbb{R}_+$. Let $\sigma \in S_n$ such that $a_{i,\sigma(i)} \neq 0$ for all $i \in [n]$. Now if an entry of the matrix is nonzero then it is always nonzero since all the entries are non-negative. Now since the scalings are symmetric we will define the potential function for i^{th} scaling (row/column) is $\Phi(i) = \prod_{i=1}^{n} a_{i,\sigma(i)}$. So we have $\Phi(0) = 1$ since at first all the entries of the matrix are from $\{0, 1\}$. Also, we know $\Phi(t) \leq 1$ for all t since every time we are scaling the matrix. Now $\Phi(1) \geq \frac{1}{n^n}$ since every row-sum can be at most n so it will be divided by n and therefore $a_{i,\sigma(i)} \geq \frac{1}{n}$ for all $i \in [n]$. Now to show the while loop stops if G has a perfect matching it suffices to show that $\Phi(t)$ increases by a multiplicative factor. So we have the following lemma.

Lemma 1.1.9

For all t, $\Phi(t+1) \geq \Phi(t)(1+\alpha)$ for some $\alpha \in (0, 1)$.

Proof: Let A' denote the matrix at the t^{th} scaling where the $(t-1)^{th}$ scaling was column-scaling. Let A'' denote the matrix after row-scaling. Now since we went to the next iteration not all column-sums are in $(1-\epsilon, 1+\epsilon)$ after scaling the rows. Now the row sums of A'' are 1. Therefore we have

$$\frac{\Phi(t)}{\Phi(t+1)} = \prod_{i=1}^{n} \textit{Col-sum}_i(A'') \leq \left(\frac{\sum_{i=1}^{n} \textit{Col-sum}_i(A'')}{n}\right)^n = \left(\frac{\sum_{i=1}^{n} \textit{Row-sum}_i(A'')}{n}\right)^n = 1 \implies \Phi(t) \leq \Phi(t+1)$$

Similarly we can say the same if $(t-1)^{th}$ scaling was row-scaling. Since not all column-sums are in $(1-\epsilon, 1+\epsilon)$ we have $\sum_{i=1}^{n} (\textit{Col-sum}_i(A'') - 1)^2 \geq \epsilon^2$. Therefore using Lemma 1.1.10 we have

$$\frac{\Phi(t)}{\Phi(t+1)} \leq 1 - \frac{\epsilon^2}{2} \implies \Phi(t+1) \geq \left(1 + \frac{\epsilon^2}{2}\right)\Phi(t)$$

Therefore we have the lemma. ∎

We have $\epsilon < \frac{1}{10n}$. Therefore, if $t \geq 200n^4$ then we have

$$1 \geq \Phi(t) \geq \frac{1}{n^n}\left(1 + \frac{1}{200n^2}\right)^t \geq \frac{1}{n^n} e^{n^2} > 1$$

Hence the while loop will iterate for at most $200n^4$ iterations. Hence, this algorithm takes $O(n^4)$ time. Hence, if G has a perfect matching the algorithm runs for at most $O(n^4)$ iterations. And if G doesn't have a perfect matching then the loop never stops. So we have the new modified algorithm to prevent infinite looping:

Algorithm 6: BP-Matrix-Scaling

Input: Bipartite adjacency matrix A of $G = (L \cup R, E)$
Output: Decide if G has a perfect matching.

```
begin
    ϵ ⟵ 1/(20n)
    for i ∈ [200n^4] do
        A ⟵ Scale every row of A to make it row stochastic.
        if All column-sums are in (1 − ϵ, 1 + ϵ) then
            return Yes
        A ⟵ Scale every column of A to make it column stochastic.
        if All row-sums are in (1 − ϵ, 1 + ϵ) then
            return Yes
```

We will prove the helping lemma needed to prove Lemma 1.1.9.

Lemma 1.1.10

Suppose $x_1, \ldots, x_n \geq 0$ and $\sum_{i=1}^{n} x_i = n$ and $\sum_{i=1}^{n} (1 - x_i)^2 \geq \delta$. Then $\prod_{i=1}^{n} x_i \leq 1 - \frac{\delta}{2} + o(\delta)$.

Proof: Denote $\rho_i = x_i - 1$. So $\sum_{i=1}^{n} \rho_i = 0$ and $\sum_{i=1}^{n} \rho^2 \geq \delta$. Now

$$\log(1+\rho_i) = \sum_{j=1}^{\infty} (-1)^{j-1} \frac{\rho_i^j}{j} \implies \log(1+\rho_i) \leq \rho_i - \frac{\rho_i^2}{3} + \frac{\rho_i^3}{3} \implies 1 + \rho_i \leq e^{\rho_i - \frac{\rho_i^2}{3} + \frac{\rho_i^3}{3}}$$

Therefore we have

$$\prod_{i=1}^{n} x_i \leq \exp\left[\sum_{i=1}^{n} \rho_i - \sum_{i=1}^{n} \frac{\rho_i^2}{3} + \sum_{i=1}^{n} \frac{\rho_i^3}{3}\right] \leq \exp\left[0 - \frac{\delta}{2} + \frac{\left(\sum_{i=1}^{n} \rho_i^2\right)^{\frac{3}{2}}}{3}\right] = \exp\left[-\frac{\delta}{2} + \frac{\delta^{\frac{3}{2}}}{3}\right] \leq 1 - \frac{\delta}{2} + o(\delta)$$

Therefore we have the lemma. ∎

There is also a survey, [Ide16] on use of matrix scaling in different results.

1.2 Matching in General Graphs

Here we give a similar algorithm[1] 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. First we will show an extension of Berge's lemma to general graphs.

Theorem 1.2.1

For any graph $G = (V, E)$, $M \subseteq E$ is a maximal matching if and only if there is no augmenting paths in G.

[1] I learned this algorithm in both Algorithm course by Umang and Combinatorial Optimization course by Kavitha. So I am mixing their notes here.

Proof: Suppose M is a maximal matching. Then if G has an augmenting path p. Then we can just take the odd edges in p and then replace the edges in $M \cap p$ with those edges i.e. $M \triangle p$ and this is a larger matching than M. But this contradicts the maximum property of M. Hence, G has no augmenting paths.

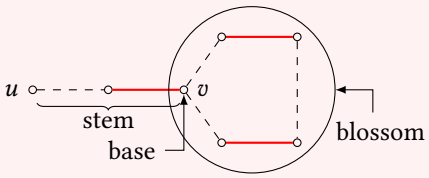
Now we will show that if M is not a maximum matching then G has an augmenting path. So suppose M is not a maximum matching. Let M' is a maximum matching. Then consider the graph $G' = (V, E')$ where $E' = M \triangle M'$. Now every vertex in V has degree $\in \{0, 1, 2\}$ in G'. Hence, the connected components of G' are isolated vertices, paths and cycles. In a path or cycle the edges of M and M' not in both appear alternatively. Therefore, the cycles are even cycles. Since $|M'| > |M|$ there exists a path p such that number $|p \cap M'| > |p \cap M|$. Therefore, the starting and ending edge of p are in M'. Hence, p is an augmenting path in G. ■

1.2.1 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 1.2.1: 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 1.5. *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.

1.2.2 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 1.1

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 $M/B = M \setminus \{\text{Matching edges in } B\}$

Chapter 2

Bibliography

[Ide16] Martin Idel. A review of matrix scaling and sinkhorn's normal form for matrices and positive maps. September 2016.

[LSW98] Nathan Linial, Alex Samorodnitsky, and Avi Wigderson. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, STOC '98, pages 644–652. ACM Press, 1998.