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

Randomized Algorithm

Here we will study randomized algorithm for tow basic problems. Later we will discuss other randomized algorithms too in the next chapters. We will also try to derandomize an algorithm in the next chapter.

1.1 Estimated Binary Search Tree Height

In this section we will calculate the expected height of a tree obtained by constructing a binary tree by picking elements uniformly at random from a given array. For this we have the following simple Intersection Algorithm

Algorithm 1: Simple Intersection Algorithm

Input: Array A of n elements of $[n]$ in any order.
Output: Construct a binary tree from A

```
1 begin
2     S ⟵ A
3     T ⟵ ∅
4     while S ≠ ∅ do
5         u ⟵ Extract(S)
6         Insert each element at the appropriate leaf of T
7     return T
```

Question 1.1

What is the expected height of the tree obtained by this Simple Intersection Algorithm assuming sequence of keys is uniformly random permutation of $[n]$.

Suppose X_n be the random variable for the height of the tree obtained by the algorithm running on any permutation of $[n]$. Let R_n be the random variable for the root of the tree obtained by the algorithm. Now consider the random variable Y_n defined as $Y_n = 2^{X_n}$. Then if we know $R_n = i$ we have

$$X_n = 1 + \max\{\text{Height of left subtree, Height of right subtree}\} = 1 + \max\{X_{i-1} + X_{n-i}\} \implies Y_n = 2\max\{Y_{n-1}, Y_{n-i}\}$$

Now for the case of $n = 1$ $Y_1 = 1$ since there is only one element and for the convenience we define $Y_0 = 0$. Now consider the following indicator random variable $Z_{n,i}$ where

$$Z_{n,i} = \begin{cases} 1 & \text{if } i \text{ is first element} \\ 0 & \text{otherwise} \end{cases}$$

So basically $Z_{n,i} = \mathbb{1}\{R_n = i\}$. Now if i is the first element then i the root of the tree obtained by the algorithm. Therefore

we have

$$\begin{aligned} Y_n &= \sum_{i=1}^{n} Z_{n,i}\,(1+\max\{Y_{i-1}, Y_{n-i}\}) \\ &\leq 2\sum_{i=1}^{n} Z_{n,i}(Y_{i-1}+Y_{n-i}) && \text{[Using Lemma 1.1.1]} \end{aligned}$$

Lemma 1.1.1 Soft Max

For any $a, b \in \mathbb{R}$,

$$\max\{a,b\} \leq \log(2^a + 2^b)$$

Therefore we have

$$\begin{aligned} \mathbb{E}[Y_n] &\leq 2\sum_{i=1}^{n} \mathbb{E}\Big[Z_{n,i}(Y_{i-1}+Y_{n-i})\Big] \\ &= 2\sum_{i=1}^{n} \mathbb{E}[Z_{n,i}]\mathbb{E}[Y_{i-1}+Y_{n-i}] \\ &= \frac{2}{n}\sum_{i=1}^{n}(\mathbb{E}[Y_{i-1}]+\mathbb{E}[Y_{n-i}]) = \frac{4}{n}\sum_{i=0}^{n-1}\mathbb{E}[Y_i] \end{aligned}$$

Now to compute $\mathbb{E}[Y_n]$ we use the following lemma

Lemma 1.1.2

$\mathbb{E}[Y_n] \leq \frac{1}{4}\binom{n+3}{3}$

Proof: We will prove this using induction on n. The base case is true for $n = 0$. Suppose this is true for $0, \ldots, n-1$.

$$\mathbb{E}[Y_n] \leq \frac{4}{n}\sum_{i=0}^{n-1}\mathbb{E}[Y_i] \leq \frac{1}{n}\sum_{i=0}^{n-1}\binom{i+3}{3} = \frac{1}{n}\binom{n+3}{4} = \frac{1}{n}\frac{(n+3)!}{4!(n-1)!} = \frac{1}{4}\binom{n+3}{3}$$

Hence by mathematical induction this is true for all n. ■

Hence by the lemma we have $\mathbb{E}[Y_n] \leq \frac{1}{4}\binom{n+3}{3} = O(n^3)$. Now by Jensen Inequality we have

$$\mathbb{E}[Y_n] = \mathbb{E}[2^{X_n}] \geq 2^{\mathbb{E}[X_n]}$$

Therefore $\mathbb{E}[X_n] \leq O(\log n)$. Therefore the expected height of a binary search tree is $O(\log n)$.

1.2 Solving 2-SAT

In this section we will discuss a randomized algorithm for deciding if a n-variate 2-SAT boolean formula is satisfiable or not.

2-SAT
Input: 2-SAT formula φ consisting of n variables.
Question: Given n-variate 2-SAT boolean formula determine if φ is satisfiable.

Here we give a simple randomized algorithm for solving the 2-SAT problem:

Algorithm 2: 2-SAT Randomized Algorithm

Input: n variate 2-SAT formula φ
Output: Decide if φ is satisfiable or not

```
begin
    ∀ i ∈ [n], Set x_i = 0
    while ∃ clause C that is not satisfied do
        Let x_i and x_j be variables in C
        Pick from {x_i, x_j} with equal probability and flip the assignment for that variable.
    return x
```

Now if the algorithm terminates it terminates with a satisfying assignment. For now assume that φ is satisfiable. We will deal with the case that φ is not satisfiable later.

Now since there are n variables there can be at most $O(n^2)$ many clauses can be in the formula. Therefore for each step of the while loop to occur it can at most take $O(n^2)$ time to find a clause which is not satisfied.

Let S represents the set of satisfying assignments for φ. Let at j^{th} iteration let A_j denote the current assignment of the variables. Let X_j be the random variable which denotes maximum number of variables of A_j that matches with some satisfying assignment of S i.e.

$$X_j = \max\{n - |x - A_j| : x \in S\}$$

At any step if $X_j = n$ then the algorithm terminates since the algorithm has found a satisfying assignment. Now starting with $X_j < n$ we consider how X_j evolves over time and how long it takes before X_j reaches n.

Now at each step we pick a clause which is unsatisfied. So we know A_j and all assignments of S disagree on the value of at least one variable of this clause. If all the assignments in S disagree with A_j on both variables changing either one will increase X_j. If there are assignments in S which disagree on the value of one of the two variables then with probability $\frac{1}{2}$ we choose that variable and increase X_j by 1 and with probability $\frac{1}{2}$ we choose the other variable and decrease X_j by 1.

Therefore X_j behaves like a random walk on a line starting from 0 which denotes the worst possible case and ends once it reaches at n where at any nonzero point it goes up or down by 1 with probability $\frac{1}{2}$. This is a Markov Chain. We want to calculate how many steps does it take on average for X_j to stumble all the way up to n. Before that we first properly define our Markov Chain.

The Markov Chain consists states from 0 to n. Where from 0 it goes to 1 with probability 1 and from n it always stays at n. And for any other state i it goes to $i+1$ with probability $\frac{1}{2}$ and goes to $i-1$ with probability $\frac{1}{2}$. Now let

$$T(k) = \text{Expected time to walk from } k \text{ to } n$$

Then we have

$$T(n) = 0, \qquad T(0) = T(1) + 1, \qquad \forall\, i \in [n-1],\ T(i) = \frac{T(i-1)}{2} + \frac{T(i+1)}{2} + 1$$

Then we have n unknowns and n equations in the above system. Therefore on average at most $O(n^2)$ steps needed to find a solution.

Now at first we said we are assuming we are dealing with the case of there exists a solution.

Question 1.2

How to deal with the issue of no solution?

In this case we will run for more number of iterations before we give up since when we give up we me might just not have found the solution. So we will run the algorithm for $100n^2$ steps. And if no solution was found then we will give up.

We first of all divide the execution of the algorithm into segments of $2n^2$ steps each. We will calculate the failure case of each segment. If the 2-SAT formula has no solution then the algorithm gives correct output. Suppose it has a solution. Then by Markov's Inequality the probability of number of steps needed to find the solution is greater than the expected number of steps needed to find a solution is at most $\frac{1}{2}$. Now after total $100n^2$ steps the probability none of the segments found a solution is 2^{-50}.

Chapter 2

Derandomization

In this section we will see a derandomization technique called Conditional Expectation. With this technique we will show derandomization of some randomized algorithms in the following sections.

2.1 Conditional Expectation

Let $\mathscr{A}$ be a randomized algorithm which is successful with probability at least $\frac{2}{3}$. Suppose $\mathscr{A}$ uses m random bits and suppose the random bits are $R_1, \ldots, R_m$. Then we have

$$\mathop{\mathbb{P}}_{R_1,\ldots,R_m}[\mathscr{A}(x, R_1, \ldots, R_m) = \text{Correct}] \geq \frac{2}{3}$$

We want to derandomize $\mathscr{A}$.

Now think of $\mathscr{A}$ as a binary tree which, given x, branches on the sampled value of each random bit R_i where it goes to left child if the random bit takes value 0 and goes to right child if the random bit takes value 1. Every path in this tree from root to leaf corresponds to different possible random strings and the leaf nodes corresponds to the output of the algorithm with the corresponding random string. Since $\mathscr{A}$ succeeds with probability at least $\frac{2}{3}$ means that at least $\frac{2}{3}$ of the leaves are good outputs for the input x.

Idea. *To derandomize $\mathscr{A}$ we need to find a deterministic algorithm that traverses from the root to a leaf which at any branch at level i chooses a direction which leads to a good output.*

Now suppose $r_1, \ldots, r_m \in \{0, 1\}$ denote the values taken by the random variables $R_1, \ldots, R_m$. Now let $P(r_1, \ldots, r_i)$ denote the fraction of the leaves of the subtree below the node obtained by following the path $r_1, \ldots, r_i$. Formally,

$$P(r_1, \ldots, r_i) = \mathbb{P}[\mathscr{A}(x, R_1, \ldots, R_m) \mid R_1 = r_1, \ldots, R_i = r_i] = \frac{1}{2}P(r_1, \ldots, r_i, 0) + \frac{1}{2}P(r_1, \ldots, r_i, 1)$$

From the last equality it is clear that there is a choice r_{i+1} such that $P(r_1, \ldots, r_{i+1}) \geq P(r_1, \ldots, r_i)$. Therefore to find a good path in the tree it suffices at each branch to pick such an $r \in \{0, 1\}$. Then we would have

$$P(r_1, \ldots, r_m) \geq P(r_1, \ldots, r_{m-1}) \geq \cdots \geq P(r_1) \geq \mathbb{P}[\mathscr{A}(x, R_1, \ldots, R_m) = \text{Correct}] \geq \frac{2}{3}$$

Since $P(r_1, \ldots, r_m)$ is either 0 or 1 it must be 1.

2.2 Max-SAT

Max-SAT

Input: SAT formula φ with n variables and m clauses and non negative weights w_c on clauses.

Question: Given a SAT formula φ with n variables and m clauses and non negative weights w_c on clauses find an assignment that maximizes weight of satisfied clauses.

We will first show a randomized algorithm for this problem. Then we will use conditional expectation to derandomize the algorithm.

2.2.1 Randomized Algorithm

First lets see what is the expected weight of satisfied clauses. Let Y_c be the indicator random variable if clause C is satisfied. Suppose there are k variables in C. Then we have $\mathbb{E}[Y_c] = 1 - \frac{1}{2^k} \geq \frac{1}{2}$. Therefore expected weight of satisfied clauses is

$$\mathbb{E}\left[\sum_C w_c Y_c\right] = \sum_C w_c \mathbb{E}[Y_c] \geq \frac{1}{2}\sum_C w_c$$

Let OPT be the optimal MAX-SAT solution for the given formula. Then we have $\sum_C w_c \geq \text{OPT}$. Therefore

$$\mathbb{E}\left[\sum_C w_c Y_c\right] \geq \frac{1}{2}\text{OPT}$$

Hence we have the following randomized algorithm:

Algorithm 3: 2-APPROXIMATE MAX-SAT

Input: SAT formula φ with n variables and m clauses and non negative weights w_c on clauses.
Output: Find an assignment that maximizes weight of satisfied clauses.

```
1 begin
2     for i ∈ [n] do
3         x_i ⟵ Pick a value from {0, 1} uniformly at random
4     return x
```

By the above discussion we have an assignment with an expected weight of satisfied clauses at least half the maximum.

2.2.2 Derandomization

Now we want to derandomize the algorithm using conditional expectation. Let $X_1, \ldots, X_n$ denote the random variable for each variables and $x_1, \ldots, x_n \in \{0, 1\}$ denote the value the random variables took. A key step will be evaluate the conditional probabilities:

$$\mathbb{E}\left[\sum_C w_c Y_c \mid X_1 = x_1, \ldots, X_i = x_i\right] = \sum_C w_c \mathbb{P}[Y_c = 1 \mid X_1 = x_1, \ldots, X_i = x_i] \quad \forall\, i \in [n]$$

Hence we have to find the value of $\mathbb{P}[Y_c = 1 \mid X_1 = x_1, \ldots, X_i = x_i], \forall\, i \in [n]$. Now if the clause C is already satisfied by the setting $x_1, \ldots, x_i$ then $Y_C = 1$. Else if C has r variables from $x_{i+1}, \ldots, x_n$ then

$$\mathbb{P}[Y_c = 1 \mid X_1 = x_1, \ldots, X_i = x_i] = 1 - \frac{1}{2^r}$$

. Now if at height i, we find $\mathbb{E}\left[\sum_C w_c Y_c \mid X_1 = x_1, \ldots, X_i = 0\right]$ and $\mathbb{E}\left[\sum_C w_c Y_c \mid X_1 = x_1, \ldots, X_i = 1\right]$ and which ever gives the higher value we will set the assignment for X_i to be that one. Thus we can derandomize the algorithm.

2.3 Set Balancing

SET-BALANCE
Input: $A \in \{0, 1\}^{n \times n}$ matrix with A_i is the i^{th} row of A and $A_{i,j}$ is the $(i, j)^{th}$ entry
Question: Given $n \times n$, 0-1 matrix A find $b \in \{1, -1\}^n$ to minimize $\|Ab\|_\infty = \max_{i \in [n]} |A_i b|$.

In the following sections we will not optimize on $\|Ab\|_\infty$. Instead we will give bound on how large min $\|Ab\|_\infty$ can be for any A.

Algorithm 4: Set-Balancing

Input: $A \in \{0,1\}^{n \times n}$ matrix
Output: Find an $b \in \{1,-1\}^n$ to minimize $\|Ab\|_\infty$

```
1 begin
2     for i ∈ [n] do
3         x_i ⟵ Pick a value from {1, −1} uniformly at random
4     return x
```

2.3.1 Randomized Algorithm

Clearly for each row $i \in [n]$ we have

$$\mathbb{E}[A_i b] = \mathbb{E}\left[\sum_j A_{i,j} b_j\right] = \sum_j \mathbb{E}[A_{i,j} b_j] = 0$$

But that does not mean $\mathbb{E}[|A_i b|] = 0$. To get a bound on $\mathbb{E}[|A_i b|]$ we will use Hoeffding's Inequality

Theorem 2.3.1 Hoeffding's Inequality

Let $Y_1, \ldots, Y_n$ be independent random variables with bounded supposer $[l_i, u_i]$ for Y_i and let $Y = \sum_{i=1}^{n} Y_i$. Then for any $\delta > 0$

$$\mathbb{P}[|Y - \mathbb{E}[Y]| > \delta] \leq 2e^{-\frac{2\delta^2}{\sum_i (u_i - l_i)^2}}$$

In our case we have $Y_{i,j} = A_{i,j} b_j$ and $Y_i = \sum_j A_{i,j} b_j$. Then each $Y_{i,j} \in \{-1, 0, 1\}$, $\mathbb{E}[Y_{i,j}] = 0$ and $\mathbb{E}[Y_i] = 0$. Therefore

$$\mathbb{P}[|Y_i| > \delta] \leq 2e^{-\frac{2\delta^2}{4n}}$$

Now we choose $\delta = 2\sqrt{n \ln n}$

$$\mathbb{P}[|A_i b| \geq 2\sqrt{n \ln n}] \leq \frac{2}{n^2}$$

Therefore $\mathbb{P}[\|Ab\|_\infty \geq 2\sqrt{n \ln n}] \leq \frac{2}{n}$ by union bound. Hence choosing each entry b uniformly at random from ± 1 we can obtain $\|Ab\|_\infty \leq 2\sqrt{n \ln n}$ with high probability.

2.3.2 Derandomization

Again we will use conditional expectation to derandomize the algorithm. Let a node at height j corresponds to a setting of $b_1, \ldots, b_j$ and we will calculate $\mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_j]$. Now consider a leaf corresponding to some choice of $b_1, \ldots, b_n$ such that the value of the leaf is < 1. But there is no randomness at the leaf. Then $\mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_n] = 0$. Hence for this choice of $b_1, \ldots, b_n$ it must have $\|Ab\|_\infty \leq 2\sqrt{n \ln n}$. Now

$$\mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_j] = \mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_j, 0] + \mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_j, 1]$$

One of them have

$$\mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_j, b_{j+1}] \leq \mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n} \mid b_1, \ldots, b_j]$$

So we choose that one. Also note that at the root $\mathbb{P}[\|Ab\|_\infty > 2\sqrt{n \ln n}] < \frac{2}{n}$. Then for choosing such a path for the corresponding choice of b we will have $\|Ab\|_\infty \leq M = 2\sqrt{n \ln n}$. But this depends on being able to calculate $\mathbb{P}[\|Ab\|_\infty > M \mid b_1, \ldots, b_j]$ which we don't know how to do in polynomial time. Instead we will use pessimistic estimator which.

2.3.3 Using Pessimistic Estimator to Derandomize

Instead of $\mathbb{P}[\|Ab\|_\infty > M \mid b_1, \ldots, b_j]$ we will use $\sum_{i\in[n]} \mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j]$. Naturally we have

$$\sum_{i\in[n]} \mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j] \geq \mathbb{P}[\|Ab\|_\infty > M \mid b_1, \ldots, b_j]$$

Now we know how to calculate $\mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j]$. For any $i \in [n]$ we have

$$\mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j] = \sum_{k=M+1}^{n} \mathbb{P}[A_i b = k \mid b_1, \ldots, b_j] + \mathbb{P}[A_i b = -k \mid b_1, \ldots, b_j]$$

Let $S_i = \{j' > j : A_{i,j'} = 1\}$ and $l = \sum_{j'\leq j} A_{i,j'}$. Then

$$\mathbb{P}[A_i b = k \mid b_1, \ldots, b_j] = \mathbb{P}\left[\sum_{j'\in S_i} b_{j'} = k - l\right]$$

Let in S_i n_i coordinates of b are 1 and rest of the coordinates of b in S_i are -1. Then

$$\sum_{j'\in S_i} b_{j'} = 2n_i - |S_i| = k - l \implies n_i = \frac{1}{2}(k - l + |S_i|)$$

Therefore we have

$$\mathbb{P}[A_i b = k \mid b_1, \ldots, b_j] = \frac{1}{2^{|S_i|}} \binom{|S_i|}{\frac{1}{2}(k - l + |S_i|)}$$

Thus we can calculate $\mathbb{P}[A_i b = k \mid b_1, \ldots, b_j]$ for all $n \geq |k| > M$. Therefore we can calculate $\mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j]$ and henceforth $\sum_{i\in[n]} \mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j]$. With this pessimistic estimator we calculate at height j both $\sum_{i\in[n]} \mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j, b_{j+1} = 0]$ and $\sum_{i\in[n]} \mathbb{P}[|A_i b| > M \mid b_1, \ldots, b_j, b_{j+1} = 1]$ and the one which have value less than 1 we will follow that path and eventually we will get an assignment of b for which $\|Ab\|_\infty \leq 2\sqrt{n \ln n}$.

Chapter 3

Global Min Cut

Global Min Cut
Input: Undirected graph $G = (V, E)$
Question: Find cut $(S, V \setminus S)$ that minimizes $|\delta(S)|$ where $\delta(S) = \{e = (u, v) \mid u \in S, v \notin S\}$.

3.1 Naive Algorithm

In previous chapter we have seen the algorithm to find $s - t$ min cut given any $s, t \in V$ in $O(n^2\sqrt{m})$ time. So naively we can run over all possible vertex pairs (s, t) and output the global min cut in $O(n^4\sqrt{m})$ time.

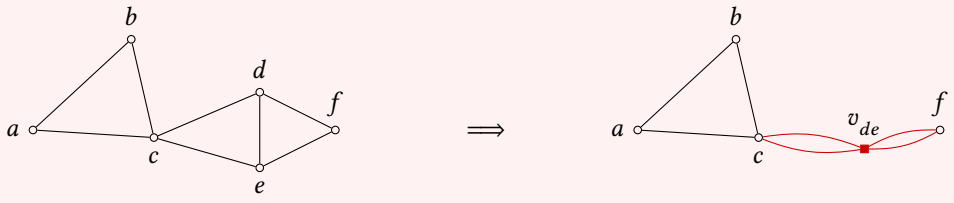
Or we can fix a vertex $s \in V$ and then for all $t \in V$ we can find the $s - t$ min cut and output the minimum. This takes $O(n^3\sqrt{m})$ time.

3.2 Karger's GMC Algorithm

Instead of naively solving the problem like above we will use randomization and will construct an algorithm which will output a global min-cut with high probability using edge contraction.

Definition 3.2.1: Edge Constraction

Given a graph $G = (V, E)$, $e = (u, v)$ edge contraction gives a multigraph (graph with multiple edges between two vertices but no self-loops) $G \setminus e = (V', E')$ where $V' = V \setminus \{u, v\} \cup \{v_e\}$ and for all $e' \in E$ if $e \cap e' = \emptyset$ then $e' \in E$ and otherwise $e' = (w, u)$ then $(w, v_e) \in E'$. The vertex v_e is called the supernode.



Observation. *For any edge $e \in G$:*

- *Any cut in $G \setminus e$ is also a cut in G of same size.*
- *Size of min cut in $G \setminus e$ is at least the size of min cut in G.*
- *Any cut in G that does not separate vertices of e is also cut in $G \setminus e$.*

Then we have the following lemma:

Lemma 3.2.1

Say k is the size of global min cut in $G' = (V', E')$ [G possible a multigraph] i.e. $\exists\ S \subseteq V'$ such that $|\delta(S)| = k$. Then $\min\{\deg(v) \mid v \in V'\} \geq k$ and $|E'| \geq \frac{k}{2}|V'|$.

Proof: If any vertex $v \in V'$ has degree less than k then we can take the cut $(\{v\}, V' \setminus \{v\})$ then $|\delta(v)| < k$, but that contradicts the fact that size of global min cut is k. Hence, contradiction ↯ Therefore $\forall\ v \in V'$, $\deg(v) \geq k$. Therefore, $|E'| = \frac{1}{2}\sum_{v \in V'} \deg(v) \geq \frac{k}{2} \cdot |V'|$. ■

So we at each round we will pick an edge from the graph uniformly at random and then contract that edge and in the next round we will pick an edge from the contracted graph. We will do $n-2$ such iterations since after that we are left with 2 supernodes $(X, V \setminus X)$.

Algorithm 5: Karger's GMC Algorithm

Input: Undirected graph $G = (V, E)$
Output: Find a cut $(S, V \setminus S)$ such that $|\delta(S)|$ is minimum

```
1 begin
2     H ⟵ G;
3     for i = 1, ..., n − 2 do
4         e ⟵Picked uniformly at random from E;
5         H ⟵ H \ e;
6     return E(H)
```

Question 3.1

What is the probability that the above algorithm returns a global min cut?

Let $(S, V \setminus S)$ is the global min cut with $|\delta(S)| = k$. Now probability that the algorithm returns $(S, V \setminus S)$ is equal to the probability that none of the edges in $\delta(S)$ is picked. So let $e_1, \ldots, e_{n-2}$ are the edges that are picked in the $n-2$ iterations of the algorithm. We need to calculate $\mathbb{P}[e_i \notin \delta(S),\ \forall\ i \in [n-2]]$

Lemma 3.2.2

$\mathbb{P}[e_1 \notin \delta(S)] \geq 1 - \frac{2}{n}$

Proof: We have $|\delta(S)| = k$. Hence, we have $|E| \geq \frac{n \cdot k}{2}$. Since e_1 is picked uniformly at random we have

$$\mathbb{P}[e_1 \notin \delta(S)] \geq 1 - \frac{k}{\frac{nk}{2}} = 1 - \frac{2}{n}$$

Hence we have the lemma. ■

Lemma 3.2.3

$\mathbb{P}[e_i \notin \delta(S) \mid e_1, \ldots, e_{i-1} \notin \delta(S)] \geq 1 - \frac{2}{n-i+1}$

Proof: Let $e_1, \ldots, e_{i-1} \notin \delta(S)$. Hence S is still a min cut in $G \setminus \{e_1, \ldots, e_{i-1}\}$. Then number of edges after contracting $e_1, \ldots, e_{i-1}$ is at least $\frac{k(n-i+1)}{2}$. Therefore

$$\mathbb{P}[e_i \notin \delta(S) \mid e_1, \ldots, e_{i-1} \notin \delta(S)]1 - \frac{k}{\frac{k(n-i+1)}{2}} = 1 - \frac{2}{n-i+1}$$

Therefore we have the lemma. ■

Hence we have

$$\begin{aligned}\mathbb{P}[\text{Success}] &\geq \mathbb{P}[e_i \notin \delta(S),\ \forall\ i \in [n-2]] \\ &= \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) \\ &= \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} = O\left(\frac{1}{n^2}\right)\end{aligned}$$

So we run the above algorithm $2n^2 \log n$ times then take the cut which gives minimum size. Then we have

$$\begin{aligned}\mathbb{P}[\text{Succeeds}] &= 1 - \mathbb{P}[\text{All } 4n^2 \log n \text{ runs fails}] \\ &\geq 1 - \left(1 - \frac{2}{n^2}\right)^{4n^2 \log n} \\ &\geq 1 - \exp\left[-\frac{2}{n^2} 2n^2 \log n\right] \\ &= 1 - \frac{1}{n^4}\end{aligned}$$

Hence, this gives a much higher probability of success. So our final algorithm is

Algorithm 6: Multiple run of Karger's GMC Algorithm

Input: Undirected graph $G = (V, E)$
Output: Find a cut $(S, V \setminus S)$ such that $|\delta(S)|$ is minimum

```
1  begin
2      S ⟵ ∅;
3      cutEdgeSize ⟵ |E|;
4      for i ∈ [2n² log n] do
5          H ⟵ G;
6          for j = 1, ..., n − 2 do
7              e ⟵Picked uniformly at random from E;
8              H ⟵ H \ e;
9          if |E(H)| < cutEdgeSize then
10             Let H = (X, V \ X);
11             S ⟵ X;
12             cutEdgeSize ⟵ |E(H)|;
13     return S
```

3.3 Karger-Stein Algorithm

In Karger's algorithm the probability of getting a min cut is low because in later stages the probability of picking an edge from a min-cut is high because

$$\mathbb{P}[e_i \in \delta(S) \mid e_1, \ldots, e_{i-1} \notin \delta(S)] \leq \frac{2}{n-i+1} \implies \mathbb{P}[e_1, \ldots, e_i \notin \delta(S)] \geq \frac{\binom{n-i}{2}}{\binom{n}{2}}$$

If the above probability is at least $\frac{1}{2}$ then $2(n-i)^2 \geq n^2 \implies n - i \geq \frac{n}{\sqrt{2}}$. Hence, i can't be too high.

So instead of running the entire algorithm $\tilde{O}(n^2)$ times we can just run the later stages multiple times. So after $i \leq n - \frac{n}{\sqrt{2}} - 1$ iterations of Karger's GMC algorithm we have

$$\mathbb{P}[e_1, \ldots, e_i \notin \delta(S)] \geq \frac{(n-i)(n-i-1)}{n(n-1)} \geq \frac{n^2}{2n(n-1)} \geq \frac{1}{2}$$

from Lemma 3.2.3. We also have the following lemma:

Lemma 3.3.1

For any $1 \leq i < j \leq n-2$ we have

$$\mathbb{P}[e_i, e_{i+1}, \ldots, e_j \notin \delta(S) \mid e_1, \ldots, e_{i-1} \notin \delta(S)] \geq \frac{(n-j)(n-j-1)}{(n-i+1)(n-i)}$$

Now fix an $i \leq n-2$. Let $l = n-i+1$. Then For $j \leq n - \frac{l}{\sqrt{2}} - 1$ we have

$$\mathbb{P}[e_i, \ldots, e_{i+j-1} \notin \delta(S) \mid e_1, \ldots, e_{i-1} \notin \delta(S)] \geq \frac{l^2}{2l(l-1)} \geq \frac{1}{2}$$

So we have the following algorithm:

Algorithm 7: KS-Algorithm

Input: Undirected graph $G = (V, E)$
Output: Find a cut $(S, V \setminus S)$ such that $|\delta(S)|$ is minimum

```
begin
    if |V| = 2 then
        return Any vertex of V
    Run Karger's GMC Algorithm on H for n − n/√2 − 1 iterations.;
    Let H be the resulting multigraph.;
    S_1 ⟵ KS-Algorithm(H);
    S_2 ⟵ KS-Algorithm(H);
    return arg min{|S_i| : i ∈ [2]}
```

Let $p(n)$ the probability of success for KS-Algorithm for a graph with n vertices. Then probability of not picking an edge until $\frac{n}{\sqrt{2}} + 1$ nodes remain is $\geq \frac{1}{2}$ as we have calculated above. Now the resulting graph has $\frac{n}{\sqrt{2}} + 1$ nodes. Hence, probability that KS-ALGORITHM(H) returns the min-cut is at least $\frac{1}{2}p\left(\frac{n}{\sqrt{2}} + 1\right)$. Therefore,

$$\mathbb{P}[\text{At least one of the run KS-ALGORITHM(H) returns the min cut}] \geq 1 - \left(1 - \frac{1}{2}p\left(\frac{n}{\sqrt{2}} + 1\right)\right)^2$$

Therefore we have

$$p(n) \geq 1 - \left(1 - \frac{1}{2}p\left(\frac{n}{\sqrt{2}} + 1\right)\right)^2$$

Solving this recursion relation we have $p(n) \geq \frac{1}{\log n}$. Hence, to succeed with high probability we need to run $2\log^2 n$ times.

Now For each run of the KS-Algorithm we have the recursion relation

$$T(n) \geq 2T\left(\frac{n}{\sqrt{2}} + 1\right) + O(n^2)$$

Solving the recursion relation we have $T(n) = O(n^2 \log n)$. Therefore, the time complexity of the total running time is $O(n^2 \log^3 n)$.

Chapter 4

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.

4.1 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

4.1.1 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 8: 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}
3      E′ ⟵ E
4      for v ∈ L do
5          E′ ⟵ E′ ∪ {(s, v)}
6      for v ∈ R do
7          E′ ⟵ E′ ∪ {(v, t)}
8      for e ∈ E′ do
9          c_e ⟵ 1
10     f ⟵ EDMONDS-KARP(G′ = (V, E′), {c_e : e ∈ E′})
11     return {e : f(e) > 0, e ∈ E}
```

Lemma 4.1.1

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.

4.1.2 Using Augmenting Paths

Definition 4.1.1: 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

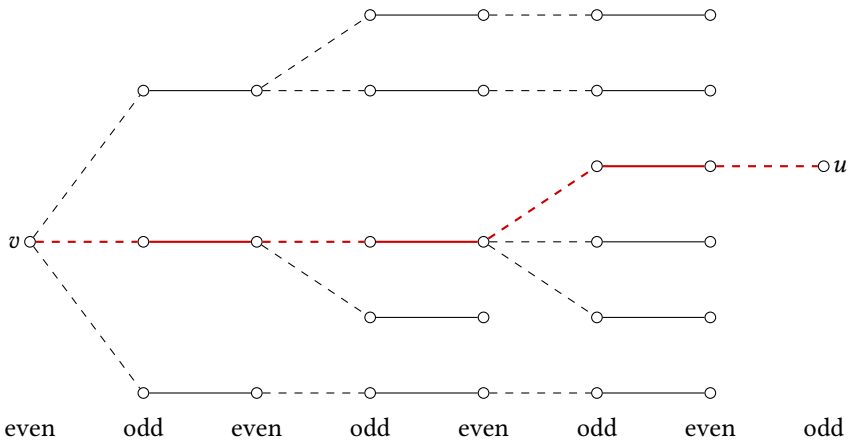
Algorithm 9: FIND-AUGMENTING-PATH(G, v)

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
```

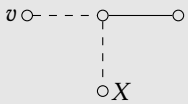
Algorithm 10: AUTree(Q)

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

Observation 4.2. *All even vertices except v are matched in T.*

Observation 4.3. *There are no edges between two odd levels or even levels.*

Lemma 4.1.2

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 4.2 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 4.1.3

$N(L_T) = R_T$

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 4.1.4

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 4.4. *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 4.1.4 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 4.1.3, $N(L_{T_{j+1}}) = R_{T_{j+1}}$ 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 4.1.5 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 11: BP-Maximum-Matching(G)

Input: $G = (L \cup R, E)$ 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}
```

4.1.3 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 4.1.6 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 4.1.2: 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 4.1.7

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 4.1.3: 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 4.1.8

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 12: 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 4.1.9

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 4.1.10 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 13: 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 4.1.9.

Lemma 4.1.10

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.

4.2 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 4.1.2. 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 4.2.1

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.

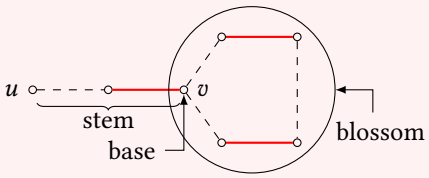
Now we will show that if M is not a maximum matching then G has an augmenting path. So suppose M is not a maximum matching. Let M' is a maximum matching. Then consider the graph $G' = (V, E')$ where $E' = M \triangle M'$. Now every vertex in V has degree $\in \{0, 1, 2\}$ in G'. Hence, the connected 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. ■

4.2.1 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 4.2.1: 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 4.5. *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.

4.2.2 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 4.1

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 4.2.2

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.

Chapter 5

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.