Parallel Algorithm and Complexity: 2023

Scribe: Soham Chatterjee

sohamchatterjee999@gmail.com
Website: **sohamch08.github.io**

Contents

1 Addition of Two Numbers in Binary

Problem: ADD_{2n}
Input: Two n bit numbers $a = a_{n-1} \cdots a_1 a_0$ and $b = b_{n-1} \cdots b_1 b_0$
Output: $s = s_n \cdots s_1 s_0$ where $s \stackrel{\text{def}}{=} a + b$

1.1 Sequential (Ripple Carry)

For sum of any position i the two bits a_i, b_i and the carry generated by the previous position c_{i-1} is added. For the initial position we can set $c_0 = 0$. If we add two bits at most 2 bits is created. The right bit is called the sum bit and the left bit is the carry bit. $a_i + b_i + c_{i-1} = c_i s_i$. Then

$$s_i = a_i \oplus b_i \oplus c_{i-1} \text{ and } c_i = (a_i \wedge b_i) \vee (b_i \wedge c_{i-1}) \vee (c_{i-1} \wedge a_i)$$

Time Complexity: This algorithm takes $O(n)$ time complexity

1.2 Parallel (Carry Look Ahead Adder)

There is a carry that ripples into position i if and only if there is some position $j < i$ to the right where this carry is generated, and all positions in between propagate this carry. A carry is generated at position i if and only if both input bits a_i and b_i are on, and a carry is eliminated at position i, if and only if both input bits a_i and b_i are off. This leads to the following definitions:

For $0 \leq i < n$, let

$g_i = a_i \wedge b_i$	position i generates a carry
$p_i = a_i \vee b_i$	position i propagates a carry that ripples int o it

So we can set for $1 \leq i \leq n$

$$c_i = \bigvee_{j=0}^{i-1} \left(g_j \wedge \bigwedge_{k=j+1}^{i-1} \right)$$

Now the sumbits are calculated as before $s_i = a_i \oplus b_i \oplus c_{i-1}$ for $0 \leq i \leq n-1$ and $s_n = c_n$
Time Complexity: This algorithm takes $O(1)$ time complexity

Definition 1.1 (AC^0). *The class of circuits consists of the gates $(\vee_n, \wedge_n, \neg_1)$ (The subscript n or 1 denotes the fanin) of polynomial size and depth $O(1) = O(\log^0 n)$*

Theorem 1.1. $ADD_{2n} = IterADD_{2,n} \in AC^0$

2 Iterated Addition

Problem: $IterADD_{k,m}$
Input: k many m-bit numbers $a_1, \ldots, a_k$
Output: The sum of the input numbers

Definition 2.1 (Length Respecting). *Let $f : \{0,1\}^* \to \{0,1\}^*$. f is length respecting if for all $x, y \in \{0,1\}^*$ $|f(x)| = |f(y)|$*

Definition 2.2 (Constant Depth Reduction). *let $f, g : \{0,1\}^* \to \{0,1\}^*$ be length respecting. Then f is constant depth reducible to g or $f \leq_{cd} g$ if there is an unbounded fanin constant depth circuit computing f from the bits of g.*

2.1 Iterated Addition of Logarithmically many n-bit numbers

Theorem 2.1. $IterADD_{\log n,n} \leq_{cd} IterADD_{\log\log n,O(n)}$

***Proof*:** We will denote $\log n = l$. We are given l many n-bit numbers $a_1, \ldots, a_l$, where $bin(a_i) = a_{i,n-1} \ldots a_{i,1}a_{i,0}$. We add all the l many bits at ith position of all numbers. we know if we add m bits then we have at most $\log m$ many bits. So adding the l many bits will take $\log l = \log\log n$ many bits. $s_k = \sum_{i=1}^{l} a_{i,k}$. Hence $bin(s_k) = s_{k,\log l-1} \ldots s_{k,1}s_{k,0}$. Hence $\sum_{i=1}^{l} a_{i,k} = \sum_{j=0}^{\log l-1} s_{k,j}2^j$

$$\sum_{i=1}^{l} a_i = \sum_{i=1}^{l}\sum_{k=0}^{n-1} a_{i,k}2^k = \sum_{k=0}^{n-1}\sum_{i=1}^{l} a_{k,k}2^k = \sum_{k=0}^{n-1}\sum_{j=0}^{\log\log n-1} s_{k,j}2^j \cdot 2^k = \sum_{j=0}^{\log\log n-1}\sum_{k=0}^{n-1} s_{k,j}2^{j+k}$$

So this is converted to addition of $\log\log n$ many numbers of at most $n + \log\log n = O(n)$ many bits. ■

Recursing like this we have $IterADD_{\log\log n,n} \leq_{cd} IterAdd_{2,O(n)}$. Hence

Theorem 2.2. $IterADD_{\log n,n} \leq_{cd} IterAdd_{2,O(n)}$ *and therefore* $IterADD_{\log n,n} \in AC^0$

Remark: Apart from this $O(\log^* n)$ method to prove $IterADD_{\log n,n} \in AC^0$ there is also another method in Vinay Kumar's Lecture Notes

2.2 Iterated Addition of n many n-bit numbers

We know $IterAdd_{n,n} \leq_{cd} IterAdd_{n,1}$ but we dont know anything about $IterAdd_{n,n} \leq_{cd} IterAdd_{\log n,n}$. If that happens it will put $IterAdd_{n,n}$ to AC^0.
Remark: $IterAdd_{n,1}$ is also known as $BCOUNT_n$.

Theorem 2.3. $IterAdd_{n,n} \leq_{cd} IterAdd_{n,1}$

***Proof*:** Let we given n many n-bit numbers $a_1, \ldots, a_n$, where $bin(a_i) = a_{i,n-1} \ldots a_{i,1}a_{i,0}$. First we compute $s_k = \sum_{i=1}^{n} a_{i,k}$ using $BCOUNT_n$ for all $0 \leq k \leq n$. Now it becomes addition of $\log n$ many $O(n)$ bit numbers which we already know is in AC^0 by Theorem 9.2. Hence $IterAdd_{n,n} \leq_{cd} BCOUNT_n$ ■

3 $IterAdd_{n,n} \equiv BCOUNT_n \equiv Threshold_{n,m} \equiv Majority_n \equiv MULT_n \equiv SORT_{n,n}$

Problem: $MULT$
Input: 2 n-bit numbers $a = a_0, \ldots, a_{n-1}$, $b = b_0, \ldots, b_{n-1}$
Output: $c = a \cdot b$

Theorem 3.1. $MULT_{n,n} \leq IterAdd_{n,n}$

Proof: Given a, b where $bin(a) = a_{n-1} \cdots a_1 a_0$ and $bin(b) = b_{n-1} \cdots b_1 b_0$ then obviously

$$a \cdot b = \sum_{i=0}^{n-1} a \cdot b_i \cdot 2^i$$

Define for all $0 \leq i \leq n-1$

$$c_i = \begin{cases} 0^{n-i-1} a_{n-1} \cdots a_1 a_0 0^u & \text{when } b_i = 1 \\ 0^{2n-1} & \text{otherwise} \end{cases}$$

i.e. $c_i = a \cdot 2^i$ if $b_i = 1$. Each c_i is of $2n-1 = O(n)$ many bits long. Hence we have $a \cdot b = \sum_{i=0}^{n-1} c_i$. Hence now we can use the $IterAdd_{n,n}$ gate to add the n many $O(n)$ many bits to find the multiplication of a and b. Therefore $MULT_n \leq IterAdd_{n,n}$. ■

Problem: $Majority_n$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if at least half of the bits are 1

Theorem 3.2. $Majority \leq MULT$

Proof: Given $a_0, \ldots, a_{n-1}$. Take the number a such that $bin(a) = a_{n-1} \cdots a_1 a_0$. Denote $l \coloneqq \log n$. Define

$$A = \sum_{i=0}^{n-1} a_i \cdot 2^{li} \text{ and } B = \sum_{i=0}^{n-1} 2^{li}$$

where both A and B consists of n blocks of length l. We took l length block because summation of n bits takes at most l bits. Let $C = A \cdot B$ We represent C in binary as l length blocks where $C = \sum_{i=0}^{2n-1} c_i \cdot 2^{li}$. Each c_i is a l length block. Then the middle block c_{n-1} have exactly the computation of the sum of the a_i. Therefore $c_{n-1} = \sum_{i=0}^{n-1} a_i$.

A and B are constructed in constant depth and fed into $MULT$ gates yielding C. Now we have to compare c_{n-1} with $\frac{n}{2}$ which can be done in constant depth. ■

Problem: $ExactThreshold_{n,m}$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if $\sum_{i=0}^{n-1} = m$

We have another similar problem but we have greater than instead of equality.

Problem: $Threshold_{n,m}$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if $\sum_{i=0}^{n-1} \geq m$

Theorem 3.3. $BCOUNT \leq ExactThreshold \leq Threshold \leq Majority$

Proof: $BCOUNT \leq ExactThreshold$: Let $\sum_{i=0}^{n-1} a_i = \sum_{i=0}^{l=\log n} s_i \cdot 2^i$. Let for all $0 \leq j \leq l$, R_j denote the set of all numbers $r \in \{0, \ldots, n\}$ whose j-th bit is 1 in its binary representation . Then we can say

$$s_j = \bigvee_{r \in R_j} \left[\!\!\left[\sum_{i=0}^{n-1} a_i = r \right]\!\!\right]$$

Now R_j don't depend on the input but only on the input n so it can be hardwired this into the circuit. Thus we have a circuit for $BCOUNT$ which uses $ExactThreshold$.

$ExactThreshold \leq Threshold$: We know for any r and a variable x certainly

$$[\![x = r]\!] = [\![x \geq r]\!] \wedge [\![x < r + 1]\!]$$

With this we have a constant depth circuit for $ExactThreshold$ using the $Threshold$ gates.

$Threshold \leq Majority$: We are given $a_0, \ldots, a_{n-1}$. Let we want to find $\sum_{i=0}^{n-1} \geq m$ then we have this following relations

$$\sum_{i=0}^{n-1} a_i \geq m \iff \begin{cases} Maj_{2n-2m}\left(a_0, \ldots, a_n, \underbrace{1, \ldots, 1}_{n-2m}\right) & \text{wher } m < \frac{n}{2} \\ Maj_{2m}\left(a_0, \ldots, a_n, \underbrace{0, \ldots, 0}_{n-2m}\right) & \text{wher } m \geq \frac{n}{2} \end{cases}$$

This Maj_{2n-2m} and Maj_{2m} can be constructed in constant depth. ■

Remark: Hence using the theorems above we have the final relation

$$Majority \leq MULT \leq IterAdd_{n,n} \leq BCOUNT \leq ExactThreshold \leq Threshold \leq Majority$$

which gives the following corollary

Corollary 3.4. $IterAdd_{n,n} \equiv BCOUNT \equiv Threshold \equiv Majority \equiv MULT$

Definition 3.1 (TC^0)**.** *Constant depth polynomial size unbounded fanin circuit family using the gates $\wedge, \vee, \neg, Maj$. Alternating Definition: Constant depth polynomial size unbounded fanin circuit family using the gates $\neg, Maj$.*

Theorem 3.5. *Both the definitions of TC^0 are equivalent.*

Theorem 3.6. $IterAdd_{n,n}, BCOUNT, MULT \in TC^0$

***Proof*:** By Corollary 10.4 we have the result. ■

Problem: $SORT$
Input: n numbers with n bits each
Output: The sequence of the input numbers in non-decreasing order.

Definition 3.2. *We define $un_n(k) \triangleq 1^k 0^{n-k}$ where $k \in \{0, \ldots, n\}$*

Problem: $UCOUNT$
Input: $a_0, \ldots, a_{n-1} \in \{0, 1\}$
Output: $un_n\left(\sum_{i=0}^{n-1} a_i\right)$

Lemma 3.7. $BCOUNT \leq UCOUNT$

Proof: Given $a_0, \ldots, a_{n-1}$ suppose $b_1 \ldots b_n = un_n \left(\sum_{i=0}^{n-1} a_i \right)$. Also let $b_0 = 1$ and $b_{n+1} = 1$. Then in the number $b_0 b_1 \ldots b_n b_{n+1}$ there is at least one 1 from left and at least one 0 from right. Now take

$$d_j = b_j \wedge \neg(b_{j+1})$$

for all $0 \leq j \leq n$. Then if $d_j = 1$ that means $b_j = 1$ and $b_{j+1} = 0$ hence b_j is the last bit which is 1 afterwards every bit is 0. Hence there are in total j many 1's except b_0. Therefore $\sum_{i=0}^{n-1} a_i = j$. So we take all $r \in 0, \ldots, n$ such that the jth bit of r is on then define

$$c_j = \bigwedge_{i \in R_j} d_i$$

Hence if $c_j = 1$ then we can say $\sum_{i=0}^{n-1} a_i$ is such a number whose jth bit is 1. Thus we take $BCOUNT(a_0, \ldots, a_{n-1}) = c_{\log n - 1} \cdots c_1 c_0$ ■

Theorem 3.8. $UCOUNT \equiv BCOUNT$

Theorem 3.9. $UCOUNT \leq SORT \leq UCOUNT$

Proof: $UCOUNT \leq SORT$: Given $a_0, \ldots, a_{n-1}$ define $A_i = a_i \underbrace{0 \cdots, 0}_{n-1}$ for all $0 \leq i \leq n-1$. Sorting these numbers the sequence of most significant bits in the ordered sequence is the $un_n \left(\sum_{i=0}^{n-1} a_i \right)$ reversed.

$SORT \leq UCOUNT$: Given $a_i = a_{i,n-1} \cdots, a_{i,1} a_{0,0}$ for all $1 \leq i \leq n$. Define

$$c_{i,j} \triangleq [\![(a_i < a_j) \vee ((a_i = a_j) \wedge i \leq j)]\!]$$

$\forall\, 0 \leq i, j \leq n-1$. Now we define for all $1 \leq j \leq n$

$$c_j \triangleq UCOUNT(c_{0,j} \ldots c_{n-1,j})$$

Hence first of all the number of 1's in $c_{0,j} \ldots c_{n-1,j}$ is the number of a_i's with value strictly less than a_j or if equal then index is less than or equal to j. Hence it represents the position of a_j when the numbers are sorted. Therefore c_j is the n-bit unary representation of the position of a_j in the ordered output sequence. If the ordered sequence is $a'_1, \ldots, a'_n$ then $a'_{c_j} = a_j$. Let $a'_i = a'_{i,n-1} \cdots a'_{i,0} a'_{i,1} a'_{i,0}$ for $1 \leq i \leq n$. Then

$$a'_{i,k} = 1 \iff [\![a'_i = a_j]\!] \wedge [\![a_{j,k} = 1]\!]$$

Hence $a'_{i,k} = \bigwedge_{1 \leq j \leq n} \left([\![c_j = 1^i 0^{n-i}]\!] \wedge a_{j,k} \right)$. ■

Corollary 3.10. $SORT \in TC^0$

4 Parallel Random-Access Machine (PRAM)

In [KR90] many Parallel Machine Models are very well written and explained

Definition 4.1 (PRAM). *A PRAM consists of an infinite sequence of processors $P_1, P_2, \ldots$. Each processor P_i has its local memory realized as an infinite sequence $R_{i,0}, R_{i,1}, R_{i,2}, \ldots$ of registers. Additionally there is a common (or global) memory given by the infinite sequence $C_0, C_1, C_2, \ldots$ of registers. Each register can hold as value a natural number, stored in binary as a bit string.*

A particular PRAM M is specified by a program and a processor bound. The program is a sequence of instructions $S_1, S_2, \ldots, S_s$ and the processor bound is a function $p : \mathbb{N} \to \mathbb{N}$

Workflow: *Initially the input is distributed over the lowest numbered global memory cells. Then all the processors $P_1, \ldots, P_{p(n)}$ start the execution of the program with the first instruction S_1. Each instruction S_m is one of the following 9 types.*

Instructions: *Suppose a fixed processor P_r, $1 \leq r \leq p(n)$. Let $c, i, j, k \in \mathbb{N}$ and $1 \leq l \leq s$. We describe the instructions and their effect in turn. If S_m is one of the first 7 types then after the execution of S_m processor P_r continues with instruction S_{m+1}*

1. *$R_i \leftarrow c$: $R_{r,i}$ gets as value the constant c.*
2. *$R_i \leftarrow \#$: $R_{r,i}$ gets as value the number of the processor i.e. r*
3. *(Numerical Operations) $R_i \leftarrow R_j + R_k$, $R_i \leftarrow R_j - R_k$: The result of adding the contents of $R_{r,j}$ and $R_{r,k}$ (Subtracting) is stored in $R_{r,i}$. (If subtraction results in a negative number the $R_{r,i}$ gets value 0)*
4. *(Bitwise Operation) $R_i \leftarrow R_j \vee R_k$, $R_i \leftarrow R_j \wedge R_k$, $R_i \leftarrow R_j \oplus R_k$: The result of performing the bitwise OR (AND, PARITY) of the contens of $R_{r,j}$ and $R_{r,k}$ is stored in $R_{r,i}$.*
5. *(Shift Operation) $R_i \leftarrow$ shl R_j, $R_i \leftarrow$ shr R_j: If the contents of $R_{r,j}$ is a where $bin(a) = a_l \cdots a_0$ then register $R_{r,i}$ will get the value b where $bin(b) = a_l \cdots a_0 0$ (for shl) and $bin(b) = a_l \cdots a_1$ (for shr)*
6. *(Indirect Addressing 1): $R_i \leftarrow (R_j)$: The contents of the global memory register whose number is given by the contents of $R_{r,i}$.*
7. *(Indirect addressing 2): $(R_i) \leftarrow R_j$: The contents of $R_{r,j}$ is copied to that global register whose number is given by the contents of $R_{r,i}$*
8. *(Conditional Jump) IF $R_i < R_j$ GOTO l: If the contents of $R_{r,i}$ is smaller in value than the contents of $R_{r,j}$ then processor P_r continues with the execution of instruction S_l; otherwise it continues with the next instruction S_{m+1}.*
9. *HALT: The computation of processor P_r stops.*

The computation of M stops when all the processors $R_1, \ldots, R_{p(n)}$ have halted

Let $f : \mathbb{N}^* \to \mathbb{N}^*$. We say that a PRAM M computes f if the following holds: Given an input $(x_1, \ldots, x_n) \in \mathbb{N}^n$ (input length: n) the x_i are first distributed in global memory cells $C_1, \ldots, C_n$ i.e. C_i is initialized to x_i for $1 \leq i \leq n$. Additionally C_0 is initialized to n. Then the computation of M is started. Let m be the contents of register C_0 after the computation stops. Then $f(x_1, \ldots, x_n)$ is the vector from $\mathbb{N}^m$ whose components are the numbers in the global register $C_1, \ldots, C_m$

Definition 4.2 (EREW-PRAM). *E:= Exclusive, R := Read, W := Write*

Question 1. *Can n, n-bit numbers be sorted in $O(\log n)$ time on an EREW-PRAM? Bir arithmetic is allowed and each with polynomially many processors. Bit operation takes $O(1)$ time. Hence is $SORT \in EREW[\log n, poly(n)]$?*

Question 2. *What about if bit arithmetic are not allowed?*

Question 3. *If comparisons are allowed can we say anything?*

Answer: AKS (Ajtai–Komlós–Szemerédi) in their paper [Pad11] showed in $O(\log n)$ time

Question 4. *If number of processors is $O(n)$ what can be said?*

There are also other parallel machine models: CRCW (C:= Concurrent), CREW, CROW (O:= Owner), OROW. Concurrent means everyone can access the memory concurrently. Owner means only the processor who is the owner of a memory can access it.

In CRCW allows simultaneous read and writes. Hence we have to resolve write conflicts. Some commonly used methods of resolving write conflicts are $COMMON$, $ARBITRARY$, $PRIORITY$ models.

Definition 4.3 (NC^1). *Class of languages accepted by circuits of depth $O(\log n)$ and size $n^{O(1)}$ and fanin 2*

Definition 4.4. *Formulas are trees i.e. circuits with fanout 1*

Theorem 4.1. *In NC^1 formulas and circuits are equivalent.*

Theorem 4.2. $NC^1 \subseteq OROW[\log n, poly(n)]$

Proof: For any circuit $C \in NC^1$ we create the OROW PRAM where each gate of C is a processor in the PRAM and each processor has the writing access of their own memory and for any edge $u \to v$ in C processor v has the reading access of the memory of u. With this OROW PRAM each processor can read memory from its children then writes the computed value in his memory location thus it computes C. Since C has size $n^{O(1)}$ the PRAM has $n^{O(1)}$ many processors and since the circuit has depth $O(\log n)$ the OROW PRAM takes $O(\log n)$ time to compute. ■

Open Question 4.3. $NC^1 \stackrel{?}{\supseteq} OROW[\log n, poly(n)]$

Definition 4.5 (AC^k, NC^k). *The class of circuits consists of the gates $(\vee, \wedge, \neg)$ (The subscript n or 1 denotes the fanin) of polynomial size and depth $O(1) = O(\log^k n)$*

Similarly for NC^k but with fanin 2

Theorem 4.4. $AC^1 = CRCW[\log n, poly(n)]$

Proof: $AC^1 \subseteq CRCW[\log n, poly(n)]$: (COMMON) For any gate all its parents have similarly reading access of the memory of the gate and all its childs has writing access. But we use COMMON model to resolve write conflicts. For an OR gate if anyone evaluates to be 0 then it ignores and if its 1 then it writes 1. Thus everybody writes 1. Every OR gate by has 0 previously in the memory. Similarly for AND gate its the opposite.

$AC^1 \supseteq CRCW[\log n, poly(n)]$: Theorem 4.5, [SV84] ■

Theorem 4.5. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a length-respecting function computed by the CRCW PRAM M in time $t(n)$. Let the processor bound of M be $p(n)$. The there is a family $\mathcal{C}$ of circuits that computes f where the depth of $\mathcal{C}$ is $O(t(n))$ and size is polynomial in $n + t(n) + p(n)$.*

Proof: See [Vol99, Theorem 2.56, Page 70] ■

5 Some Circuit Complexity Class Relations

Theorem 5.1. $AC^0 \subseteq NC^1$

Proof: For all unbounded gate in AC^1 circuit C with fanin s we can replace each gate with s many same gates fo fanin 2 and depth log s. Thus we have $AC^0 \subseteq NC^1$ ■

Theorem 5.2. $TC^0 \subseteq NC^1$

Proof: We will first show that using Redundant Algebra $ADD_{2n} \in NC^0$. In Redundant Algebra with base 4 while adding two digits the result can be at most in normal integers $-6, \ldots, 6$. We only have to check for $\pm 6, \pm 5, \pm 3$. Other case we dont have to watch out.

$$\begin{array}{lll} 6 = 12 & 5 = 11 & 3 = 1\overline{1} \\ \overline{6} = \overline{12} & \overline{5} = \overline{11} & \overline{3} = \overline{11} \end{array}$$

For ± 4 it is the normal representation in this system ie 10 and $\overline{10}$ respectively. Now whenever we add two digits in this system in the sum result can be at most 2 digits. Among them we call the right most digit the sum digit or s and the left digit the carry digit c becasue it becomes the carry for the addition. Now see in all of the numbers the sum digit $|s| \leq 2$ and carry digit $|c| \leq 1$ So whenever we add a carry generated before to the current sum no new carry is generated becasue of the carry. So We dont have to look for carry generation and propagation. The carry generated at the previous position will add to the sum digit in the current position after getting added to that it will not further propagate after getting added the final digit at the current place will be between 3 and $\overline{3}$ So we proved that addition of two n-bit numbers using Redundant algebra is in NC^0

Now we will show converting a number n from base 2 to base 4 is in NC^0. let

$$n = \sum_{k=0}^{m} a_k 2^k = \sum_{k=0}^{\lfloor \frac{m}{2} \rfloor} (a_{2k+1} \times 2 + a_{2k})\, 4^k \qquad \text{if } m \text{ is odd then } a_{m+1} = 0$$

So we just take two bits in binary multiply the left one with 2 and add to the right one and we have the digit at base 4 in that position. So we can change base in NC^0. From now on we will by default assume all the addition is done using Redundant Algebra.

Then we are done in showing $TC^0 \subseteq NC^1$. We will first show that adding n bits is in NC^1 ie $BCOUNT \in NC^1$. So we have n bits $a_n, a_{n-1}, \ldots, a_1$. We will group the bits into groups of two bits and add the two bits in each group. We can do the addition for all groups in parallel. And since we have already shown addition of two k-bit numbers is in NC^0 this takes constant depth and size since we are adding two bits. Now the number of bits are halved. We do tha same process again and again. Each time it takes $poly(n)$ size and constant depth and at each iteration the number of numbers becomes half. So this whole process takes $O(\log n)$ many iterations. So adding n bits takes $poly(n)$ size and depth $O(\log n)$. So $BCOUNT \in NC^1$

Now We will show that $Majority \in NC^1$. Let the addition of n bits we got is a and the Redundant Algebra representation of $\lceil \frac{n}{2} \rceil$ is b. Now we can calculate $-b$ with reversing the sign of every digit in b ie if $b = \sum_{i=0}^{m} a_k \times 4^k$ then $-b = \sum_{i=0}^{m} \overline{a_k} \times 4^k$. Now we add a and $-b$ which can be done in NC^0. And now we will look for the left most negative digit. If there is nonw we output 1 i.e. majority of the bits are 1 and if there exists a negative bit then output is 0. Hence we have $Majority \in TC^0$. So $TC^0 \subseteq NC^1$. ■

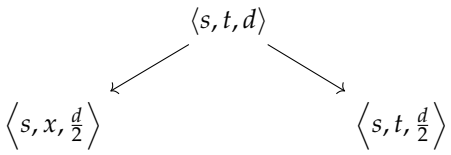
Definition 5.1 (SAC^k)**.** *The class of circuits consists of the gates* $(\vee, \wedge, \neg)$ *of polynomial size and depth* $O(1) = O(\log^k n)$ *but semi unbounded fanin i.e either* $\wedge$ *gates have unbounded fanin and* $\vee$ *gate have bounded fanin or* $\wedge$ *gates have bounded fanin and* $\vee$ *gates have unbounded fanin.*

Theorem 5.3. $AC^0 \subseteq TC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq SAC^1 \subseteq AC^1 \subseteq NC^2 \subseteq AC^2 \subseteq \cdots \subseteq NC^i \subseteq AC^i \subseteq \cdots \subseteq NC = AC \subseteq P$

Proof: $NC^1 \subseteq L$: The idea is to simply evaluate the circuit using a depth-first search, which can be defined inductively as follows: visit the output gate of the circuit, visit the gates of the left subcircuit, visit the gates of the right subcircuit. At any moment, we store the number of the current gate, the path that led us there (as a sequence of left's and right's) and any partial values that were already computed. For example, we could have L, R (0), R (1),

L, L, 347. This would mean that we are visiting gate 347 and we got there by going left, right, right, left and left. The 0 after the first R indicates that the left input of the second gate on the path evaluated to 0. When we are done visiting a gate (and computed its value), we return to the previous gate on the path. Note that we can recompute the number of that gate by following the path from the beginning. The space requirements of this algorithm are therefore determined by the maximum length of the path, which is equal to the depth of the circuit. The uniformity of the circuit is used when traveling through the circuit and evaluating its gates.

$NL \subseteq SAC^1$: We know $PATH$ is $NL - complete$. Let there is a path $s \rightsquigarrow t$ of length d. Then there exists a vertex x in between s and t such that there exists a path of length $\frac{d}{2}$ from $s \rightsquigarrow x$ and $x \rightsquigarrow t$.



Since there exists one such vertex x and in circuit we will add all this in a big $\vee$ gate for all vertices in the graph. We can represent this as a circuit

$$\langle s,t,d\rangle = \bigvee_{x\in V}\left(\left\langle s,x,\frac{d}{2}\right\rangle \wedge \left\langle s,t,\frac{d}{2}\right\rangle\right)$$

Like this we extend it to 1 length paths. In this circuit we are using $\vee$ gates with unbounded fanin and $\wedge$ gates with 2 fanin. Hence we have $NL \subseteq SAC^1$ ■

Theorem 5.4 (Formula Depth Reduction)**.** *If f has a formula F of size s then there exists an equivalent formula F' of depth $d = O(\log s)$*

***Proof*:** We will prove this by induction on s. When $s = 1$ or 2 this is trivially true.

For the induction step, assume that the inequality holds for all formulas of size strictly less than s. Given a formula F of size s, we find a node g in the formula such that the number of leaves in the subformula below g is at least $\frac{s}{2}$ but individually the left and right subformula below g have strictly less than $\frac{s}{2}$ leaves. Such a node can easily be found by starting at the root and checking if the current node satisfies the properties required. If so we have found g, else there exists a child of g such that number of leaves below it is at least $\frac{s}{2}$. We move to this child and check if this node satisfies the properties needed. Continuing in this way we find the node g. We use the following equivalence

$$F \equiv (g \wedge F|_{g=1}) \vee (g \wedge F|_{g=0})$$

Let $g = g_L B g_R$ where $B \in \{\wedge, \vee\}$. By the choice of g, we have that the size of each subformula g_L, g_R, $F|_{g=0}$, and $F| + g = 1$ is at most $\frac{s}{2}$. Inductively we balance these subformulas to obtain the formula F'. Hence the depth of the formula thus constructed is given by the following recurrence

$$d(s) = d\left(\frac{s}{2}\right) + O(1) \implies d(s) = O(\log s)$$

Therefore $depth(F') = O(\log s)$.■

6 Iterated Multiplication

Problem: $ItMult_{m,n}$
Input: m many n-bit numbers are given
Output: The product of the given numbers

Theorem 6.1 (Chinese Remainder Theorem). *Let $k \in \mathbb{N}$, $P = p_1 p_2 \cdots p_k$. Let $a_1, \ldots, a_k$ be given. Then there is a unique number $a \in \{0, \ldots, P-1\}$ such that $a \equiv a_i \pmod{p_i}$ for $1 \leq i \leq k$. More specifically, we have*

$$a = \left[\sum_{i=1}^{k} (a_i \pmod{p_i}) \right] \pmod{P}$$

where $r_i = \frac{P}{p_i}$ and $s_i = \left(\frac{P}{p_i}\right)^{-1} \pmod{p_i}$ (i.e. s_i is the multiplicative inverse of $r_i \pmod{p_i}$).

Theorem 6.2 (Prime Number Theorem). *Let $\Pi(n)$ denote the number of prime numbers not greater than n. Then*

$$\Pi(n) = \Theta\left(\frac{n}{\log n}\right)$$

Corollary 6.3. *Let p_k denote the kth prime. $p_n = O(n \log n)$*

Theorem 6.4. *Let p_i denote the ith prime. Then*

$$\prod_{i=1}^{n} p_i \leq 4^{n \log n}$$

Theorem 6.5. $ItMult_{n,n} \in TC^0$

Proof: Let $a_i = a_{i,n-1} \cdots a_{i,1} a_{i,0}$ for $1 \leq i \leq n$ be the given numbers which we want to multiply up. Hence

$$a = a_1 \cdot a_2 \cdots a_n < (2^n)^n = 2^{n^2} < p_1 \cdot p_2 \cdots p_{n^2}$$

Then take $p = p_1 \cdot p_2 \cdots p_{n^2}$. Then our goal is to compute $a \pmod{p}$. Our first goal is to compute $a \pmod{p_i}$ for all $1 \leq i \leq n^2$.

Now take $r_i = \frac{p}{p_i}$ and $s_i = \left(\frac{p}{p_i}\right)^{-1} \pmod{p_i}$. Then by Chinese Remainder Theorem we have

$$a = \sum_{i=1}^{n^2} (a \pmod{p_j}) \cdot r_i \cdot s_i$$

Now we have

$$a \pmod{p_j} = \prod_{i=1}^{n} a_i \pmod{p_j} = \prod_{i=1}^{n} (a_i \pmod{p_j})$$

Take

$$a'_{i,j} := a_i \pmod{p_j} = \left(\sum_{k=0}^{n-1} a_{i,k} \cdot 2^k \right) \pmod{p_j} = \sum_{k=0}^{n-1} a_{i,k} \cdot (2^k \pmod{p_j}) \quad \text{for } 1 \leq i \leq n,\ 1 \leq j \leq n^2$$

This gives n^3 many numbers all of which can be computed in constant depth. The values $2^k \pmod{p_j}$ can be hardwired into the circuit since they do not depend on the input but only the input length. So we can compute $a'_{i,j}$ using $ItAdd$ and $MULT$ gates (since we know they are already in TC^0) in constant depth. Using the Corollary 6.3 we have $p_j = O(n^2 \log n)$ for all $1 \leq j \leq n^2$. Therefore all the $a'_{i,j}$ consists of $O(\log n)$ bits only.

For $1 \leq j \leq n^2$ let g_j be the generator of the multiplicative group $\mathbb{Z}^*_{p_j}$ group of the finite field $\mathbb{Z}_{p_j}$. Let for $k \in \{1, \ldots, p_j - 1\}$ take $g^{m_j(k)} = k$ where $m_j(k) \in \{0, \ldots, p_j - 2\}$. And we set $m_j(0) := p_j - 1$. So now we will compute $m_j\left(a'_{i,j}\right)$. Since the numbers $a'_{i,j}$ contains $O(\log n)$ many bits, these computations can be done in constant depth. Therefore

$$m_j\left(\prod_{i=1}^{n} a_i \pmod{p_j}\right) = m_j\left(\prod_{i=1}^{n} (a_i \pmod{p_j})\right) = \left(\sum_{i=1}^{n} m_j\left(a'_{i,j}\right)\right) \pmod{p_j - 1}$$

for $1 \leq j \leq n^2$. This can be computed in constant depth using $ItAdd$ gates. The case when $a'_{i,j} = 0$ can be taken care of. These n^2 numbers have logarithmically many bits. Thus we can compute the value $\prod_{i=1}^{n} a_i \pmod{p_j}$ for $1 \leq j \leq n^2$ in constant depth. Finally we compute

$$a' := \sum_{i=1}^{n^2} \left(\prod_{i=1}^{n} a_i \pmod{p_j} \right) \cdot r_j \cdot s_j$$

in constant depth with $ItAdd$ and $MULT$ gates. The values r_j and s_j can be hardwired into the circuit since they don't depend on the input instead the length of the input.

By Chinese Remainder Theorem we have $a = a' \pmod{a'}$ that is $q := \left\lfloor \frac{a'}{p} \right\rfloor$ we have $a = a' - q \cdot p$. Since $r_j = \frac{p}{p_j}$ and $s_j \leq p_j$ we have $r_j \cdot s_j \leq p$. Therefore

$$a' = \sum_{i=1}^{n^2} \left(\prod_{i=1}^{n} a_i \pmod{p_j} \right) \cdot r_j \cdot s_j \leq \sum_{i=1}^{n^2} \left(\prod_{i=1}^{n} a_i \pmod{p_j} \right) p \leq \sum_{i=1}^{n^2} p_{n^2} p = n^2 \cdot p_{n^2} \cdot p \implies q \leq n^2 \cdot p_{n^2}$$

This the value of q is polynomial in n and can be determined by parallely testing for all $i \leq n^2 \cdot p_{n^2}$ if $i \cdot p \leq a' \leq (i+1)p$ which can be done in constant depth and q is the one value of i for which the answer of this is yes. Then the result of the iterated multiplication is $a = a' - q \cdot p$. ■

Theorem 6.6. $ItMult \in FO[Maj]$

7 Some Special Problems

Definition 7.1 ($ApproximateMajority$ or AM). *For $x \in \{0,1\}^n$ let $wt(x)$ be the number of ones in x. Then*

$$AM(x) = \begin{cases} 0 & if\, wt(x) \leq \frac{n}{4} \\ 1 & if\, wt(x) \geq \frac{3n}{4} \\ don't\ care & otherwise \end{cases}$$

Theorem 7.1 ([Ajt83, Section 4]). $Approximate\ Majority \in AC^0$

Theorem 7.2 ([MC89, Theorem 2.1, Corollary 2.2]). $Dyck_k \in DLogTime - Uniform\ TC^0$

Theorem 7.3 ([Bus87]). *Boolean Formula Value Problem* $\in ALogTime$

Corollary 7.4. $NC^1 = ALogTime$

Theorem 7.5 ([GL18]). *Tree Balancing* $\in DLogTime - uniform\ TC^0$

8 Addition of Two Numbers in Binary

Problem: ADD_{2n}
Input: Two n bit numbers $a = a_{n-1} \cdots a_1 a_0$ and $b = b_{n-1} \cdots b_1 b_0$
Output: $s = s_n \cdots s_1 s_0$ where $s \stackrel{\text{def}}{=} a + b$

8.1 Sequential (Ripple Carry)

For sum of any position i the two bits a_i, b_i and the carry generated by the previous position c_{i-1} is added. For the initial position we can set $c_0 = 0$. If we add two bits at most 2 bits is created. The right bit is called the sum bit and the left bit is the carry bit. $a_i + b_i + c_{i-1} = c_i s_i$. Then

$$s_i = a_i \oplus b_i \oplus c_{i-1} \text{ and } c_i = (a_i \wedge b_i) \vee (b_i \wedge c_{i-1}) \vee (c_{i-1} \wedge a_i)$$

Time Complexity: This algorithm takes $O(n)$ time complexity

8.2 Parallel (Carry Look Ahead Adder)

There is a carry that ripples into position i if and only if there is some position $j < i$ to the right where this carry is generated, and all positions in between propagate this carry. A carry is generated at position i if and only if both input bits a_i and b_i are on, and a carry is eliminated at position i, if and only if both input bits a_i and b_i are off. This leads to the following definitions:

For $0 \leq i < n$, let

$g_i = a_i \wedge b_i$	position i generates a carry
$p_i = a_i \vee b_i$	position i propagates a carry that ripples int o it

So we can set for $1 \leq i \leq n$

$$c_i = \bigvee_{j=0}^{i-1} \left(g_j \wedge \bigwedge_{k=j+1}^{i-1} \right)$$

Now the sumbits are calculated as before $s_i = a_i \oplus b_i \oplus c_{i-1}$ for $0 \leq i \leq n-1$ and $s_n = c_n$
Time Complexity: This algorithm takes $O(1)$ time complexity

Definition 8.1 (AC^0). *The class of circuits consists of the gates $(\vee_n, \wedge_n, \neg_1)$ (The subscript n or 1 denotes the fanin) of polynomial size and depth $O(1) = O(\log^0 n)$*

Theorem 8.1. $ADD_{2n} = IterADD_{2,n} \in AC^0$

9 Iterated Addition

Problem: $IterADD_{k,m}$
Input: k many m-bit numbers $a_1, \ldots, a_k$
Output: The sum of the input numbers

Definition 9.1 (Length Respecting). *Let $f : \{0,1\}^* \to \{0,1\}^*$. f is length respecting if for all $x, y \in \{0,1\}^*$ $|f(x)| = |f(y)|$*

Definition 9.2 (Constant Depth Reduction). *let $f, g : \{0,1\}^* \to \{0,1\}^*$ be length respecting. Then f is constant depth reducible to g or $f \leq_{cd} g$ if there is an unbounded fanin constant depth circuit computing f from the bits of g.*

9.1 Iterated Addition of Logarithmically many n-bit numbers

Theorem 9.1. $IterADD_{\log n, n} \leq_{cd} IterADD_{\log \log n, O(n)}$

Proof: We will denote $\log n = l$. We are given l many n-bit numbers $a_1, \ldots, a_l$, where $bin(a_i) = a_{i,n-1} \ldots a_{i,1} a_{i,0}$. We add all the l many bits at ith position of all numbers. we know if we add m bits then we have at most $\log m$ many bits. So adding the l many bits will take $\log l = \log\log n$ many bits. $s_k = \sum_{i=1}^{l} a_{i,k}$. Hence $bin(s_k) = s_{k,\log l-1} \ldots s_{k,1} s_{k,0}$. Hence $\sum_{i=1}^{l} a_{i,k} = \sum_{j=0}^{\log l-1} s_{k,j} 2^j$

$$\sum_{i=1}^{l} a_i = \sum_{i=1}^{l} \sum_{k=0}^{n-1} a_{i,k} 2^k = \sum_{k=0}^{n-1} \sum_{i=1}^{l} a_{k,k} 2^k = \sum_{k=0}^{n-1} \sum_{j=0}^{\log\log n-1} s_{k,j} 2^j \cdot 2^k = \sum_{j=0}^{\log\log n-1} \sum_{k=0}^{n-1} s_{k,j} 2^{j+k}$$

So this is converted to addition of $\log\log n$ many numbers of at most $n + \log\log n = O(n)$ many bits. ■

Recursing like this we have $IterADD_{\log\log n,n} \leq_{cd} IterAdd_{2,O(n)}$. Hence

Theorem 9.2. *$IterADD_{\log\log n,n} \leq_{cd} IterAdd_{2,O(n)}$ and therefore $IterADD_{\log\log n,n} \in AC^0$*

Remark: Apart from this $O(\log^* n)$ method to prove $IterADD_{\log\log n,n} \in AC^0$ there is also another method in Vinay Kumar's Lecture Notes

9.2 Iterated Addition of n many n-bit numbers

We know $IterAdd_{n,n} \leq_{cd} IterAdd_{n,1}$ but we dont know anything about $IterAdd_{n,n} \leq_{cd} IterAdd_{\log n,n}$. If that happens it will put $IterAdd_{n,n}$ to AC^0.
Remark: $IterAdd_{n,1}$ is also known as $BCOUNT_n$.

Theorem 9.3. *$IterAdd_{n,n} \leq_{cd} IterAdd_{n,1}$*

Proof: Let we given n many n-bit numbers $a_1, \ldots, a_n$, where $bin(a_i) = a_{i,n-1} \ldots a_{i,1} a_{i,0}$. First we compute $s_k = \sum_{i=1}^{n} a_{i,k}$ using $BCOUNT_n$ for all $0 \leq k \leq n$. Now it becomes addition of $\log n$ many $O(n)$ bit numbers which we already know is in AC^0 by Theorem 9.2. Hence $IterAdd_{n,n} \leq_{cd} BCOUNT_n$ ■

10 $IterAdd_{n,n} \equiv BCOUNT_n \equiv Threshold_{n,m} \equiv Majority_n \equiv MULT_n \equiv SORT_{n,n}$

Problem: $MULT$
Input: 2 n-bit numbers $a = a_0, \ldots, a_{n-1}$, $b = b_0, \ldots, b_{n-1}$
Output: $c = a \cdot b$

Theorem 10.1. *$MULT_{n,n} \leq IterAdd_{n,n}$*

Proof: Given a, b where $bin(a) = a_{n-1} \cdots a_1 a_0$ and $bin(b) = b_{n-1} \cdots b_1 b_0$ then obviously

$$a \cdot b = \sum_{i=0}^{n-1} a \cdot b_i \cdot 2^i$$

Define for all $0 \leq i \leq n-1$

$$c_i = \begin{cases} 0^{n-i-1} a_{n-1} \cdots a_1 a_0 0^u & \text{when } b_i = 1 \\ 0^{2n-1} & \text{otherwise} \end{cases}$$

i.e. $c_i = a \cdot 2^i$ if $b_i = 1$. Each c_i is of $2n-1 = O(n)$ many bits long. Hence we have $a \cdot b = \sum_{i=0}^{n-1} c_i$. Hence now we can use the $IterAdd_{n,n}$ gate to add the n many $O(n)$ many bits to find the multiplication of a and b. Therefore $MULT_n \leq IterAdd_{n,n}$. ■

Problem: $Majority_n$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if at least half of the bits are 1

Theorem 10.2. $Majority \leq MULT$

Proof: Given $a_0, \ldots, a_{n-1}$. Take the number a such that $bin(a) = a_{n-1} \cdots a_1 a_0$. Denote $l := \log n$. Define

$$A = \sum_{i=0}^{n-1} a_i \cdot 2^{li} \text{ and } B = \sum_{i=0}^{n-1} 2^{li}$$

where both A and B consists of n blocks of length l. We took l length block because summation of n bits takes at most l bits. Let $C = A \cdot B$ We represent C in binary as l length blocks where $C = \sum_{i=0}^{2n-1} c_i \cdot 2^{li}$. Each c_i is a l length block. Then the middle block c_{n-1} have exactly the computation of the sum of the a_i. Therefore $c_{n-1} = \sum_{i=0}^{n-1} a_i$.

A and B are constructed in constant depth and fed into $MULT$ gates yielding C. Now we have to compare c_{n-1} with $\frac{n}{2}$ which can be done in constant depth. ■

Problem: $ExactThreshold_{n,m}$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if $\sum_{i=0}^{n-1} = m$

We have another similar problem but we have greater than instead of equality.

Problem: $Threshold_{n,m}$
Input: n bits $a_{n-1}, \ldots, a_0$
Output: Find if $\sum_{i=0}^{n-1} \geq m$

Theorem 10.3. $BCOUNT \leq ExactThreshold \leq Threshold \leq Majority$

Proof: $BCOUNT \leq ExactThreshold$: Let $\sum_{i=0}^{n-1} a_i = \sum_{i=0}^{l=\log n} s_i \cdot 2^i$. Let for all $0 \leq j \leq l$, R_j denote the set of all numbers $r \in \{0, \ldots, n\}$ whose j-th bit is 1 in its binary representation . Then we can say

$$s_j = \bigvee_{r \in R_j} \left[\!\!\left[\sum_{i=0}^{n-1} a_i = r \right]\!\!\right]$$

Now R_j don't depend on the input but only on the input n so it can be hardwired this into the circuit. Thus we have a circuit for $BCOUNT$ which uses $ExactThreshold$.

$ExactThreshold \leq Threshold$: We know for any r and a variable x certainly

$$[\![x = r]\!] = [\![x \geq r]\!] \wedge [\![x < r + 1]\!]$$

With this we have a constant depth circuit for $ExactThreshold$ using the $Threshold$ gates.

$Threshold \leq Majority$: We are given $a_0, \ldots, a_{n-1}$. Let we want to find $\sum_{i=0}^{n-1} \geq m$ then we have this following relations

$$\sum_{i=0}^{n-1} a_i \geq m \iff \begin{cases} Maj_{2n-2m}\left(a_0, \ldots, a_n, \underbrace{1, \ldots, 1}_{n-2m}\right) & \text{wher } m < \frac{n}{2} \\ Maj_{2m}\left(a_0, \ldots, a_n, \underbrace{0, \ldots, 0}_{n-2m}\right) & \text{wher } m \geq \frac{n}{2} \end{cases}$$

This Maj_{2n-2m} and Maj_{2m} can be constructed in constant depth. ■

Remark: Hence using the theorems above we have the final relation

$$Majority \leq MULT \leq IterAdd_{n,n} \leq BCOUNT \leq ExactThreshold \leq Threshold \leq Majority$$

which gives the following corollary

Corollary 10.4. $IterAdd_{n,n} \equiv BCOUNT \equiv Threshold \equiv Majority \equiv MULT$

Definition 10.1 (TC^0)**.** *Constant depth polynomial size unbounded fanin circuit family using the gates $\wedge, \vee, \neg, Maj$. Alternating Definition: Constant depth polynomial size unbounded fanin circuit family using the gates $\neg, Maj$.*

Theorem 10.5. *Both the definitions of TC^0 are equivalent.*

Theorem 10.6. $IterAdd_{n,n}, BCOUNT, MULT \in TC^0$

Proof: By Corollary 10.4 we have the result. ■

11 Division

12 Matrix Inversion

13 Maximal Independent Set (MIS)

Theorem 13.1. $MIS \in P$

13.1 Matching and Independent Set of Line Graph

Definition 13.1 (Line Graph)**.** *The line graph of the graph G, written $L(G)$ is the graph whose vertices are the edges of G, with $(e_1, e_2) \in E(L(G))$ when $e_1 \cap e_2 \neq \phi$*

Theorem 13.2. *Given a graph G a set of edges $S \subseteq E$ is a matching if and only if it is a independent set in the line graph $L(G)$*

Proof: ($\Rightarrow$): Let S be a matching of G. Therefore for all $e_1, e_2 \in S$ we have $e_1 \cap e_2 - \phi$. Hence e_1, e_2 are not adjacent in $L(G)$. Hence S is an independent set of $L(G)$.

($\Leftarrow$): Let S be a independent set in $L(G)$. Then for all $e_1, e_2 \in S$, e_1 and e_2 are not adjacent. Therefore $e_1 \cap e_2 = \phi$. Hence the set S is a set of edges of G where none of them shares any endpoint. Hence S is a matching in G. ■

Fact 1. *Maximal (Maximum) Matching in G is an Maximal (Maximum) Independent Set in the line graph $L(G)$.*

13.2 Luby's Algorithm (Randomized Algorithms)

Definition 13.2 (RNC^k)**.** *The class RNC^k is the class of problems that can be solved by a randomized algorithm that runs in $O(\log^k n)$ time with a polynomial number of processors.*

Remark: Therefore RNC is the randomized counterpart of NC.

Luby's Algorithm puts MIS in RNC^2. The main steps or the ideas of the algorithm are:

- The algorithm tries to find I in each stage.
- Each stage finds an independent set I in parallel, using calls on a
- Create a set S of candidates for I as follows: For each vertex v in parallel, include $v \in S$ with probability $\frac{1}{2d(v)}$
- For each edge in E if both its endpoints are in S, discard the one of lower degree, ties are resolved arbitrarily
- The resulting set is I

Here we denote for any $v \in V$ $d(v) \coloneqq \deg(v)$.

13.3 Analysis of Luby's Algorithm

Now we will define certain things which will help us to analyze the algorithm:

- A vertex $v \in V$ is *good* if
$$\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \frac{1}{6}$$
A pair of vertices $u, v \in V$ is said to be *good* if
$$good(u,v) \iff [\![(u,v) \in E]\!] \wedge [\![good(u) \vee good(v)]\!]$$
Intuitively a vertex is good if it has lots of neighbors of low degree. This will give it a decent chance of making into $N(I)$. Therefore in case of bad vertex $bad(v) = \neg good(v)$ and for a pair of vertices $u, v \in V$ we have $bad(u,v) = \neg good(u,v)$
- We also define
$$N^-(v) = \{u \in N(v) \mid d(u) \leq d(v)\} \qquad d^-(v) = |N^-(v)|$$
$$N^+(v) = \{u \in N(v) \mid d(u) > d(v)\} \qquad d^+(v) = |N^+(v)|$$
Therefore we have $d^+(v) + d^-(v) = d(v)$.

Lemma 13.3. *For any $v \in V$*
$$bad(v) \implies d^+(v) \geq 2d^-(v) \iff d^-(v) \leq \frac{d(v)}{3} \iff d^+(v) \geq \frac{2d(v)}{3}$$

Algorithm 1: Luby's Randomized Algorithm on MIS

```
begin
    A ⟵ ∅
    while G ≠ ∅ do
        S ⟵ ∅
        I ⟵ ∅
        for v ∈ V(G) in parallel do
            add v to S with probability 1/(2d(v))
        for {u, v} ∈ E(G) in parallel do
            if u ∈ S and v ∈ S then
                if d(u) < d(v) then
                    delete u from S
                else if d(v) < d(u) then
                    delete v from S
                else if u < v then
                    delete u from S
                else
                    delete v from S
        Call the resulting set after deletions I
        A ⟵ A ∪ I
        G ⟵ G \ (I ∪ N(I))
    return A
```

Proof: We make the graph directed where if $(u, v) \in E$ previously then the direction of this edge will be

$$u \to v \iff [\![d(u) < d(v)]\!] \vee \Big([\![d(u) = d(v)]\!] \wedge [\![u < v]\!]\Big)$$

Then $d^-(v)$ in the original graph indicates the in-degree of v and $d^+(v)$ indicates the out-degree of v. Therefore $d^+(v) \geq \frac{2d(v)}{3}$ means out=degree of v is twice more than the in-degree of v. Now assume the statement is not true. Let v is *bad*. Then

$$\frac{1}{6} > \sum_{u \in N(v)} \frac{1}{2d(u)} \geq \sum_{u \in N^-(v)} \frac{1}{2d(u)} \geq \sum_{u \in N^-(v)} \frac{1}{2d(v)} \geq \frac{d^-(v)}{2d(v)} > \frac{d(v)}{3} \times \frac{1}{2d(v)} \geq \frac{1}{6}$$

Hence contradiction. ■

Lemma 13.4. *At least half of the edges in the graph are good*

Proof: We again construct the same directed graph from G as in the proof of Lemma 13.3. If $(u, v) \in E$ then the direction of this edge will be

$$u \to v \iff [\![d(u) < d(v)]\!] \vee \Big([\![d(u) = d(v)]\!] \wedge [\![u < v]\!]\Big)$$

Now for any edge $e = (u, v) \in E$

$$bad(e) \iff bad(u) \wedge bad(v)$$

Let the direction of e is $u \to v$. Since e is bad the vertex v is bad. Therefore using Lemma 13.3 out-degree of v is twice more than the in-degree of v. Hence there is at least two edges out-going from v. Let those edges are $e_1 = v \to w_1$ and $e_2 = v \to w_2$. Hence for every bad vertex there are two edges in E. Hence

$$2\,|\{e \in E \mid bad(e)\}| \leq |E|$$

Therefore we have $2\#bad(e) \leq \#good(e) + \#bad(v) \implies \#good(e) \geq \#bad(e) \implies 2\#good(v) \geq |E|$. ■

Lemma 13.5. *For any $v \in V$*

$$Pr[v \notin I \mid v \in S] \leq \frac{1}{2}$$

Proof: If $v \in S$ and $v \notin I$ only if there exists some element of $N^+(v)$ which is also in S. Then

$$\begin{aligned}
Pr[v \notin I \mid v \in S] &= Pr[\exists\, u \in N^+(v) \cap S \mid v \in S] \\
&\leq \sum_{u \in N^+(v)} Pr[u \in S \mid v \in S] \\
&= \sum_{u \in N^+(v)} Pr[u \in S] && \text{[Pairwise independence]} \\
&\leq \sum_{u \in N^+(v)} \frac{1}{2d(u)} \leq \sum_{u \in N^+(v)} \frac{1}{2d(v)} \\
&\leq \frac{d(v)}{2d(v)} && \text{[Lemma 13.3]} \\
&\leq \frac{1}{2}
\end{aligned}$$

■

Lemma 13.6. *For any $v \in V$*

$$Pr[v \in I] \geq \frac{1}{4d(u)}$$

Proof: We have

$$\begin{aligned}
Pr[v \in I] &= Pr[v \in I \mid v \in S]\, Pr[v \in S] \\
&\geq (1 - Pr[v \notin I \mid v \in S])\, \frac{1}{2d(v)} \\
&= \frac{1}{2} \times \frac{1}{2d(v)} = \frac{1}{4d(v)} && \text{[Lemma 13.5]}
\end{aligned}$$

■

Lemma 13.7. *If $v \in V$ is good then*

$$Pr[v \in N(I)] \geq \frac{1}{36}$$

Proof: We will consider two cases.
Case 1: v has a neighbor u of degree 2 or less. Then using Lemma 13.6

$$Pr[v \in N(I)] \geq Pr[u \in I] \geq \frac{1}{4d(u)} \geq \frac{1}{8}$$

Case 2: $d(u) \geq 3$ for all $u \in N(v)$. Then for all $u \in N(v)$ we have $\frac{1}{2d(u)} \leq \frac{1}{6}$. Now since v is good we have $\sum_{u \in N(v)} \frac{1}{2d(u)} \geq \frac{1}{6}$. Therefore there must exist a subset $M(v) \subseteq N(v)$ such that

$$\frac{1}{6} \leq \sum_{u \in M(v)} \frac{1}{2d(u)} \leq \frac{1}{3} \tag{1}$$

Therefore

$$\begin{aligned}
Pr[v \in N(v)] &\geq Pr[\exists\, u \in M(v) \cap I] \\
&\sum_{u \in M(v)} pr[u \in I] - \sum_{\substack{u,w \in M(v) \\ u \neq w}} Pr[u \in I \wedge w \in I] && \text{[Inclusion-Exclusion]} \\
&\sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{\substack{u,w \in M(v) \\ u \neq w}} pr[u \in S \wedge w \in S] && \text{[Lemma 13.6]} \\
&\sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{\substack{u,w \in M(v) \\ u \neq w}} Pr[u \in S] Pr[v \in S] && \text{[Pairwise independence]} \\
&\sum_{u \in M(v)} \frac{1}{4d(u)} - \sum_{u \in M(v)} \sum_{w \in M(v)} \frac{1}{2d(u)} \frac{1}{2d(w)} \\
&\sum_{u \in M(v)} \frac{1}{2d(u)} \left[\frac{1}{2} - \sum_{w \in M(v)} \frac{1}{2d(w)} \right] \\
&\geq \frac{1}{6} \left(\frac{1}{2} - \frac{1}{3} \right) = \frac{1}{36} && \text{[By (1)]}
\end{aligned}$$

■

Lemma 13.8. *If $e \in E$ is good then*

$$Pr[e \text{ is deleted}] \geq \frac{1}{36}$$

Proof: If the edge $e = (u, v) \in E$ is deleted then either u or v is good vertex. Let u is good. Since e is deleted then $Pr[u \in N(I)] \geq \frac{1}{36}$ by Lemma 13.7. Hence

$$Pr[e \text{ is deleted}] \geq Pr[good(u) \wedge u \in N(I)] + Pr[good(v) \wedge v \in N(I)] \geq \frac{1}{36}$$

■

Lemma 13.9. *Let X ve the random variable representing the number of deleted edges. Then*

$$\mathbb{E}[X] \geq \frac{|E|}{72}$$

Proof: Take the indicator random variable for each edge $e \in E$

$$X_e = \begin{cases} 1 & \text{if } e \text{ is deleted} \\ 0 & \text{otherwise} \end{cases}$$

Therefore $X = \sum_{e \in E} X_e$. Then we have

$$\begin{aligned} \mathbb{E}[X] &= \mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}[X_e] \\ &\geq \sum_{good(e)} \mathbb{E}[X_e] \geq \sum_{good(e)} Pr[e \text{ is deleted}] \\ &\geq \sum_{good(e)} \frac{1}{36} && \text{[Lemma 13.8]} \\ &\geq \frac{|E|}{2} \times \frac{1}{36} = \frac{|E|}{72} && \text{[Lemma 13.4]} \end{aligned}$$

■

Theorem 13.10. *Luby's Algorithm puts $MIS \in RNC^2$*

13.4 Derandomization

Theorem 13.11. *Bertrand's Postulate For any integer $n > 1$ there always exists at least one prime p with*

$$n < p < 2n$$

From the proof of Lemma 13.5 and Lemma 13.7 we can see that we don't need the vertices to be independent. Pairwise independent is enough for the analysis. We construct pairwise independent family of functions by taking a prime p in the range n to $2n$ (such prime exists because of Theorem 13.11) and then we can assume the vertices of the graph are the elements of the finite field $\mathbb{Z}_p$. Now for each vertex u we take $a(u)$ be an integer in $\mathbb{Z}_p$ such that $\frac{1}{2d(u)} \approx \frac{a(u)}{p}$ that is $\frac{a(u)}{p}$ is as close as possible to $\frac{1}{2d(u)}$. Then we denote $A_u := \{0, 1 \ldots, a(u) - 1\}$. We can take A_u as any subset of $\mathbb{Z}_p$ of size $a(u)$. Hence probability of landing in this set is $\frac{a(u)}{p} \approx \frac{1}{2d(u)}$.

Now we choose x and y uniformly at random $\mathbb{Z}_p$ and define the function

$$f_{x,y} : v \mapsto x + vy \pmod{p}$$

Now for any $u, v, \alpha, \beta \in \mathbb{Z}_p$ where $u \neq v$ then there exists exactly one solution to the linear system

$$x + uy = \alpha \qquad x + vy = \beta$$

since the matrix $\begin{bmatrix} 1 & u \\ 1 & v \end{bmatrix}$ is invertible and

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & u \\ 1 & v \end{bmatrix}^{-1} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Therefore

$$\begin{aligned}
\Pr_{x,y\in\mathbb{Z}_p}\left[f_{x,y}(u)\in A_u \wedge f_{x,y}(v)\in A_v\right] &= \Pr_{\substack{\alpha\in A_u\\ \beta\in A_v}}\left[x+uy=\alpha \wedge x+vy=\beta\right] \\
&= \frac{1}{p^2}\left|\{(x,y)\mid x+uy\in A_u \wedge x+vy\in A_v\}\right| \\
&= \frac{1}{p^2}\sum_{\alpha\in A_u}\sum_{\beta\in A_v}\left|\{(x,y)\mid x+uy=\alpha \wedge x+vy=\beta\}\right| \\
&= \frac{1}{p^2}\sum_{\alpha\in A_u}\sum_{\beta\in A_v} 1 \\
&= \frac{1}{p^2}a(u)a(v) = \frac{a(u)}{p}\,\frac{a(v)}{p} \approx \frac{1}{2d(u)}\,\frac{1}{2d(v)}
\end{aligned}$$

So we take the family of pairwise independent functions $\mathcal{H}=\{f_{x,y}\mid x,y\in\mathbb{Z}_p\}$. We can construct this family with only $2\log p = O(\log n)$ random bits (while choosing x,y). Hence there are total $2^{O(\log n)}=n^{O(1)}$ many functions. So $|\mathcal{H}|=n^{O(1)}$.

So we in parallel consider all possible strings of $O(\log n)$ length representing all possible outcomes of the $O(\log n)$ random bits. From each of this string we construct the function in $\mathcal{H}$ and carry on the algorithm deterministically. Since we expect to delete at least a constant fraction of the edges (Lemma 13.9) one of the functions must delete at least that many edges. So we pick the function which deletes the most edges and throw the other parallel computation away and then repeat the whole process. Everything is deterministic and at least a constant fraction of the edges are removed at each stage.

For each stage we choose different the prime p and then do the process as discussed above.

Theorem 13.12. *Derandomization of Luby's Algorithm puts $MIS\in NC^2$*

14 Counting Perfect Matchings in Planar Graph

Definition 14.1 (Tutte Matrix)**.** *The Tutte Matrix of a digraph $G=V(E)$ is the matrix $T_G[X]$ where*

$$T_G[X]=\begin{cases} x_{ij} & \text{when } (i,j)\in E \\ -x_{ij} & \text{when } (i,j)\in E \\ 0 & \text{when } i=j \\ 0 & \text{otherwise}\end{cases}$$

Hence we can see that $T_G[X]$ is an skew symmetric matrix.

Definition 14.2 (Nice Cycle)**.** *The cycle C which is of even length and $G-V(C)$ has a perfect matching is called a nice cycle.*

Definition 14.3 (Oddly Oriented)**.** *An even cycle C is called oddly oriented if for either choice of direction of traversal around C the number of edges of C directed in the direction of the traversal is odd.*

In other words the number of edges of C directed in the opposite direction of the traversal is odd.

Definition 14.4 (Pfaffian Orientation)**.** *Pfaffian Orientation of a graph G is an orientation where every nice cycle of G is oddly oriented.*

Definition 14.5 (Pfaffian Graph)**.** *A graph G is Pfaffian if it has a Pfaffian orientation.*

Lemma 14.1. *Let $\sigma\in S_n$ where $\sigma=(C_1)(C_2)\cdots(C_k)$. If $\exists\, i\in[k]$ such that C_i is an odd cycle then the monomial $\prod_{i=1}^{n} T_{i,\sigma(i)}$ gets canceled out in $\det(T_G[X])$*

Proof: $\sigma = (C_1)(C_2)\cdots(C_k)$. Therefore σ represents a cycle cover $\mathcal{C}$ of G. Given that $\mathcal{C}$ has an odd cycle. Now for any cycle C define the head of the cycle to be the vertex with the smallest index. . Let the odd cycle in $\mathcal{C}$ with smallest head is C_j. Now $C)j$ corresponds to a permutation $\tau \in S_n$. Therefore C_j reversed i.e C_j^R corresponds to τ^{-1}. Then take the cycle cover $\mathcal{C}' = \{C_1, \ldots, C_{j-1}, C_j^R, C_{j+1}, \ldots, C_k\}$. $\mathcal{C}'$ corresponds to the permutation $\sigma' = (C_1)(C_2)\cdots(C_{j-1})(C_j^R)(C_{j+1})\cdots(C_k)$.

After reversing C_j for any edge $u \to v \in C_j$ the edge $v \to u \in C_j^R$. In $T_G[X]$ we have $T_{v,u} = -T_{u,v}$. Hence for all edge in C_j the sign is changed after reversing. Since the length of C_j is odd we have $sgn(C_j) = -sgn(C_j^R)$. Since all the other cycles in $\mathcal{C}'$ and $\mathcal{C}$ are same there sign is not changed. Hence $sgn(\sigma') = -sgn(\sigma)$. Since the monomial generated by σ and σ' are same only their sign is reversed they will cancel each other in the determinant. Hence $\prod_{i=1}^{n} T_{i,\sigma(i)}$ gets canceled out in $\det(T_G[C])$. ■

Theorem 14.2 (Tutte)**.** $\det(T_G[X]) \neq 0 \iff$ *G has a perfect matching*

Proof: $(\Rightarrow)$: Suppose $\det(T_G[X]) \neq 0$. Then there exists a $\sigma \in S_n$ such that the monomial $\prod_{i=1}^{n} T_{i,\sigma(i)}$ in not canceled out. Now by Lemma 14.1 the cycle cover $\mathcal{C}$ corresponding to σ has only even cycles. So for each cycle $C \in \mathcal{C}$ we get a matching for the vertices in C. [Just take the odd (even) positioned edges.] Since the cycles in $\mathcal{C}$ are edge disjoint union of all these edges gives a perfect matching in G. Hence G has a perfect matching.

$(\Leftarrow)$: We will prove the contrapositive. Let G has no perfect matching. Then suppose $\det(T_G[X]) \neq 0$. Then by previous arguments there exists a $\sigma \in S_n$ such that the monomial $\prod_{i=1}^{n} T_{i,\sigma(i)}$ in not canceled out. Now again by Lemma 14.1 the cycle cover $\mathcal{C}$ corresponding to σ has only even cycles. Each of these cycles will give out a matching for the vertices in that cycle. Hence we obtain a perfect matching for G. Contradiction ↯ Hence we have $\det(T_G[X]) = 0$. ■

If G has a perfect matching then you can take the cycle cover of all the transpositions for each edge in the perfect matching. Then the monomial for this permutation will not get canceled in the determinant.

Remark: From now on by matching we mean to say perfect matching. Also we define the set

$$S = \{\sigma \in S_n \mid \text{every cycle in the cycle cover corresponding to } \sigma \text{ is even cycle}\}$$

Also we define the set

$$\mathcal{M}_G = \{\text{set of all perfect matching of } G\}$$

We will use heavily in this section.

14.1 Pfaffian, Matchings, Determinant

Lemma 14.3. *There is a bijection φ between $\varphi : S \to \mathcal{M}_G \times \mathcal{M}_G$*

Proof: content... ■

Let π be any partition of the vertices of G. Let

$$\pi = \{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_k, j_k\}\}$$

Now we will define some quantities:

- By $wt(\pi)$ we define the quantity

$$wt(\pi) = sgn \underbrace{\begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & 2k-1 & 2k \\ i_1 & j_1 & i_2 & j_2 & \cdots & i_k & j_k \end{pmatrix}}_{:=\sigma_\pi} \prod_{l=1}^{k} T_{i_l,j_l}$$

- $Pf(T_G) := \sum_{pi} wt(\pi) = \sum_{M \in \mathcal{M}_G} wt(M)$
- Let D be any orientation of G. Then we define

$$sgn_D(M) = sgn \underbrace{\begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & 2k-1 & 2k \\ i_1 & j_1 & i_2 & j_2 & \cdots & i_k & j_k \end{pmatrix}}_{:=\sigma_M}$$

  where the edge direction of for $\{i_l, j_l\}$ is $i_l \to j_l \; \forall \; l \in [k]$ and we start from the smallest index i_1.

Clearly π is a matching of G. So we can write this as $wt(M)$ where $M \in \mathcal{M}_G$. Since T_G is a skew symmetric matrix $wt(\pi)$ or $wt(M)$ does not depend on the order of the blocks in π (Because if any 2 blocks' positions get swapped then in the permutation we have to multiply 2 transpositions to σ_π or σ_M which doesn't change the sign) or the order of the elements in a block in π (Because if their order is changed then we have to multiply a single transposition to σ_π or σ_M multiplies the sign with (-1) and from the variables let m is the index of the block where the positions of i_m and j_m is changed then $T_{j_m,i_m} = -T_{i_m,j_m}$ hence we get another (-1) which is multiplied hence the sign remains same.)

Lemma 14.4. *D is Pfaffian orientation $\iff$ $\forall$ M,1 $M_2 \in \mathcal{M}_G$, we have $sgn_D(M_1) = sgn_D(M_2)$*

Proof: $(\Rightarrow)$: Let $M_1, M_2 \in \mathcal{M}_G$. By Lemma 14.3 $\exists\, \sigma \in S$ such that $\varphi(\tau) = (M_1, M_2)$. Let $\tau = (C_1)(C_2)\cdots(C_k)$. Now for each C_i is even cycle. And by the bijection φ $G - V(C_i)$ has a perfect matching. Therefore each C_i is nice cycle. We denote $\sigma_{M_i} = \sigma_i$ for $i = 1, 2$ for simplicity. Since D is Pfaffian orientation for each $i \in [k]$ there are odd number of edges in C_i which are oddly oriented. Now because of orientation we can say that if $u \to v$ is an edge of M_1 then any one of $\tau(u) \to \tau(v)$ and $\tau(v) \to \tau(u)$ is an edge of M_2. Or we can say if for any $l \in [\frac{n}{2}]$ $\sigma_1(2l-1) \to sg_1(2l)$ is an edge of M_1 then any one of $\tau \circ \sigma_1(2l-1) \to \tau \circ \sigma(2l)$ and $\sigma_1(2l-1) \to sg_1(2l)$ is an edge of M_2. Now from $\tau \circ \sigma_1$ in order to obtain σ_2 we need to multiply some transpositions since some of the edges are in opposite direction. So 3 cases arise:

Case 1: If an opposite direction edge is in M_1 and the next edge is in correct direction which is in M_2. Let the opposite direction edge is $\sigma_1(2l) \to sg_1(2l-1)$ then the next edge which is in M_2 is $\tau \circ \sigma_1(2l-1) \to \tau \circ \sigma_1(2l)$. Hence we have to multiply the transposition $\begin{pmatrix} 2l-1 & 2l \end{pmatrix}$ to $\tau \circ \sigma_1$. Hence a negative sign gets multiplied.

Case 2: If a edge of M_1 is in correct direction but the next edge is in opposite direction which is in M_2. Let the edge is $\sigma_1(2l-1) \to sg_1(2l)$ then the next edge which is in M_2 is $\tau \circ \sigma_1(2l) \to \tau \circ \sigma(2l-1)$. Hence we have to multiply the transposition $\begin{pmatrix} 2l-1 & 2l \end{pmatrix}$ to $\tau \circ \sigma_1$. Hence a negative sign gets multiplied.

Case 3: 2 consecutive same direction edges either in opposite to the cycle direction or in following the cycle direction the first one is in M_1 and the next one is in M_2. Let the edge in M_1 $\sigma_1(2l-1) \to sg_1(2l)$ then the next edge which is in M_2 is $\tau \circ \sigma_1(2l-1) \to \tau \circ \sigma(2l)$. Hence no need to multiply transposition. So no need to change anything.

Now since there are odd number of edges which are in opposite direction for Case 3 the opposite edges appear in pairs apart from Case 3 there are still odd number of edges which are in opposite direction. Hence with Case 1 and 2 after multiplying the transpositions the multiply odd number of transpositions which changes the sign. Hence over all a (-1) gets multiplied. Since we do this for all cycles C_i we multiply (-1) for each cycle. Therefore we can write $sgn(\sigma_2) = (-1)^k sgn(\tau \circ \sigma_1)$. Now since τ has k many even cycles we have $sgn(\tau) = (-1)^k$. Therefore we have

$$sgn(\sigma_2) = (-1)^k sgn(\tau \circ \sigma_1) = (-1)^k sgn(\tau) sgn(\sigma_1) = (-1)^k \times (-1)^k sgn(\sigma_1) = sgn(\sigma_1)$$

Therefore for all $M_1, M_2 \in \mathcal{M}_G$ we have $sgn_D(M_1) = sgn_D(M_2)$

($\Leftarrow$) : We have $\forall\ M_1, M_2\ sgn_D(M_1) = sgn_D(M_2)$. Now take a nice cycle C. Then $G - V(C)$ has a matching, M'. Now since C is even we get two matching M_1 and M_2 by taking the odd edges in M_{C_1} and even edges in M_{C_2} of $V(C)$ from C. Now we create two matching of G, $M_1 = M' \cup M_{C_1}$ and $M_2 = M' \cup M_{C_2}$. So by Lemma 14.3 $\exists\ \tau \in S$ such that $\varphi(\tau) = (M_1, M_2)$. So in the cycle cover $\mathcal{C}$ obtained from τ for all the vertex pairs not in C forms a 2-cycle but because of orientation we have both the edges in same direction so one in the direction of the 2-cycle and the other is in the opposite direction of the direction of the 2-cycle.

Like in the forward direction proof we have 3 cases of occurrences of opposite direction edges. In order to obtain σ_2 form $\tau \circ \sigma_1$ we have a transposition multiplied to $\tau \circ \sigma_1$ for each 2-cycle. Let there are m many 2-cycles. So m many transpositions are multiplied to $\tau \circ \sigma_1$ and hence $(-1)^m$ is multiplied to $sgn(\tau \circ \sigma_1)$ because of the transpositions due to the 2-cycles. Let $(-1)^{m'}$ is multiplied because of the transpositions of C. Now since there are $m+1$ many cycles in τ $sgn(\tau) = (-1)^{m+1}$. So therefore from $sgn(\sigma_1) = sgn(\sigma_2)$ we have $(-1)^m \times (-1)^{m'} = (-1)^{m+1} \implies (-1)^{m'} = (-1)$ Hence there are odd number of transpositions multiplied to $\tau \circ \sigma_1$ because of C. Hence there are odd number of edges in C which are in opposite directions following the 3 cases described in the forward direction proof. Hence for every nice cycle there are odd number of edges which are oddly oriented. Hence D is a Pfaffian orientation. ■

Theorem 14.5 ([Cay49]). $\det(T_G[X]) = Pf(T_G)^2$

Proof: We have

$$\det(T_G) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)}$$

By Lemma 14.1 and since the diagonal entries are 0 the cycle cover corresponding to $\sigma \in S_n$ containing at least one odd cycle cancels out and the permutations having $\sigma(i) = i$ for any $i \in [n]$ the monomial becomes zero. So we have

$$\det(T_G) = \sum_{\sigma \in S} sgn(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)}$$

We have

$$Pf(T_G)^2 = \sum_{M_1, M_2 \in \mathcal{M}_G \times \mathcal{M}_G} wt(M_1) wt(M_2)$$

By Lemma 14.3 there exists $\sigma \in S$ such that $\varphi(\sigma) = (M_1, M_2)$. Let $\sigma = (C_1)(C_2)\cdots(C_m)$. Now

$$wt(M_1) = sgn_D(M_1) \prod_{l=1}^{k} T_{\sigma_1(2l-1),\sigma_1(2l)} \qquad wt(M_2) = sgn_D(M_2) \prod_{l=1}^{k} T_{\sigma_2(2l-1),\sigma_2(2l)}$$

Where D is some orientation on G.

Now to obtain σ_2 from $\tau \circ \sigma_1$ we have to multiply some transposition in case of any opposite direction edges in the cycles. Let the product of all these transpositions are denoted by ψ. Then we have

$$\sigma_2 = \psi \circ \tau \circ \sigma_1 \implies sgn(\sigma_1) sgn(\sigma_2) = sgn(\psi) sgn(\tau)$$

Now for each transposition $\psi' = (i, \quad \tau(i))$ in ψ for some i it means the edge $\tau(i) \to i$ present in a cycle of the cycle cover of τ which is opposite to the direction of cycle following the orientation D. Since we have $T_{\tau(i),i} = -T_{i,\tau(i)}$ we can replace the monomial $T_{\sigma(i),i}$ in $wt(M_1)wt(M_2)$ by $-T_{i,\sigma(i)}$. So we do this for all the transpositions in ψ. Hence after replacing all these variables we can say

$$\left[\prod_{l=1}^{k} T_{\sigma_1(2l-1),\sigma_1(2l)}\right] \left[\prod_{l=1}^{k} T_{\sigma_2(2l-1),\sigma_2(2l)}\right] = sgn(\psi) \prod_{i=1}^{n} T_{i,\tau(i)}$$

So we get

$$\begin{aligned} wt(M_1)wt(M_2) &= sgn(\sigma_1)sgn(\sigma_2)\left[\prod_{l=1}^{k} T_{\sigma_1(2l-1),\sigma_1(2l)}\right]\left[\prod_{l=1}^{k} T_{\sigma_2(2l-1),\sigma_2(2l)}\right] \\ &= sgn(\sigma_1)sgn(\sigma_2) \times \left[sgn(\psi)\prod_{i=1}^{n} T_{i,\tau(i)}\right] \\ &= sgn(\psi)sgn(\tau) \times \left[sgn(\psi)\prod_{i=1}^{n} T_{i,\tau(i)}\right] \\ &= sgn(\tau)\prod_{i=1}^{n} T_{i,\tau(i)} \end{aligned}$$

Therefore we get $Pf(T_G)^2 = \det(T_G)$ ■

14.2 Pfaffian Orientation of Planar Graph

Lemma 14.6. *G is a planar graph. Suppose G admits an orientation D such that every internal face F is oddly oriented in clockwise traversal. Then D is Pfaffian*

Proof: We will prove a stronger claim here.

Claim 1. *For any simple cycle C addition of the number f of edges in forward direction and the number k of vertices strictly inside C is odd.*

Proof of Claim 1: We will induct on k. Let $k = 0$. Then basically the cycle is the boundary cycle of an internal face. Then this is true by the hypothesis of the lemma.

Now suppose C encloses more than one face. We can view C is the symmetric difference of two smaller simple cycles C_1 and C_2. Let f_1, k_1 and f_2, k_2 be the number of forward direction edges and internal vertices for C_1 and C_2 respectively. So $f_1 + k_1 \equiv 1 \pmod 2$ and $f_2 + k_2 \equiv 1 \pmod 2$. Now let the path P shared by C_1 and C_2 contains m vertices. So the number of vertices of C is $k = k_1 + k_2 + b$. Since all the edges of P is either in forward direction with respect to C_1 or C_2 if we take $f_1 + f_2$ then this number has all the edges of P. So the number of edges in forward direction of C is $f = f_1 + f_2 - (b + 1)$. Hence

$$k + f = (k_1 + k_2 + b) + (f_1 + f_2 - (b + 1)) = (k_1 + f_1) + (k_2 + f_2) - 1 \equiv 1 \pmod 2$$

So by induction this is true for any simple cycle. ■

Now if we take any nice cycle C in G. Then $G - V(C)$ has a matching. Since C is even cycle the vertices in C also has a matching. So we obtain a matching of G. Hence if there are odd number of vertices strictly inside C there exists at least on edge of in the matching which connects a vertex inside C and a vertex outside C. But this edge would be cutting the cycle C at some point which is not possible since G is planar graph. Therefore the number of vertices strictly inside C is even. By the claim we can say that the number of edges in forward direction in C are odd. Since this is true for any nice cycle in G. D is Pfaffian. ■

Theorem 14.7 ([Kas04])**.** *Every Planar graph is Pfaffian*

Proof: content... ■

Theorem 14.8. *Let D be the pfaffian orientation of G. Then let B is the adjacency matrix of G following the orientation D. Then we have*

$$\det(B) = |\mathcal{M}_G|^2$$

Proof: Since G is planar such a pfaffian orientation D by Lemma 14.4 we have for all $M_1, M_2 \in \mathcal{M}_G$ $sgn_D(M_1) = sgn_D(M_2)$. Therefore $sgn(wt(M_1)wt(M_2)) = 1$ and each $T_{\sigma_j(2l-1),\sigma_j(2l)} = 1$ for all $l \in [k]$ in Theorem 14.5. Hence for all $M_1, M_2 \in \mathcal{M}_G$ we have the value of $wt(M_1)wt(M_2) = 1$. So we have by $\det(B) = |\mathcal{M}_G|^2$ ■

15 Isolation Lemma

Theorem 15.1 (Isolation Lemma)**.** *Fix a finite set $S \subseteq \mathbb{R}$ be a finite set and let $T_1, \ldots, T_k \in 2^{[n]}$. For each $i \in [n]$ independently assign a uniformly random weight from S. Let*

$$w(T_i) = \sum_{x \in T_i} w(i)$$

Then we have

$$Pr[\exists!\ T_i \text{ of minimum weight}] \geq 1 - \frac{n}{|S|}$$

Proof: Suppose E_i be the event that

$$\min\{w(T_j) \colon i \notin T_j\} = \min\{w(T_j) \colon i \in T_j\}$$

Claim 1: If none of the E_i occur, then the minimum weight is unique.
Proof: We will proof the contrapositive statement. Suppose T_i and T_j are distinct minimum-weight sets. Then there exists at least one element $x \in T_i$ such that $x \notin T_j$. Since T_i and T_j have minimum weights, the event E_x occurs. Therefore we have if none of the E_i occur, then the minimum weight set is uniques. ■

Proving this now notice that

$$Pr\left[\bigcap_{i \in [n]} \neg E_i\right] = 1 - Pr\left[\bigcap_{i \in [n]} E_i\right] \geq 1 - \sum_{i \in [n]} Pr[E_i]$$

Therefore in the following claim we will bound $Pr[E_i]$.

Claim 2: $Pr[E_i] \leq \frac{1}{|S|}$
Proof: Now fix all weights, $w(j)$ except the weight $w(i)$. Let

$$L = \min\{w(T_j - i) \mid i \notin T_j\} = \min\{w(T_j) \mid i \notin T_j\}$$
$$R = \min\{w(T_j - i) \mid i \in T_j\}$$

Since

$$E_i : \quad \min\{w(T_j) \colon i \notin T_j\} = \min\{w(T_j) \colon i \in T_j\}$$

we have that

$$E_i \text{ occurs} \iff L = R + wt(i)$$

Hence there is at most one option for $wt(i)$ to choose from S so that this equation holds. Therefore $Pr[E_i] = \underset{w \in S}{Pr}[L = R + w \wedge w = wt(i)] \leq \frac{1}{|S|}$ ■

Therefore by claim 2 we have $\forall\, i \in [n]$, $Pr[E_i] \leq \frac{1}{|S|}$. Hence

$$Pr\left[\bigcap_{i \in [n]} \neg E_i\right] \geq 1 - \sum_{i \in [n]} Pr[E_i] \geq 1 - \frac{n}{|S|}$$

■

16 Perfect Matching in Bipartite Planar Graph

16.1 Nice Cycles and Circulation

Let $G = (V, E)$ be a graph with a perfect matching.

Definition 16.1 (Nice Cycle)**.** *A cycle C in G is a nice cycle if it has even length and the subgraph $G - C$ still has a perfect matching*

In other words a nice cycle can be obtained from the symmetric difference of two perfect matchings.

Now suppose we have a weight function $w\colon E \to \mathbb{R}$ on the edges of a graph G. Let we have an even length cycle

$$C = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{2k-2}} v_{2k} \xrightarrow{e_{2k-1}} v_0$$

in G for some $k \in \mathbb{N}$.

Definition 16.2 (Circulation of Cycle)**.** *For a weight assignment w on the edges the circulation $c_w(C)$ of an even length cycle is defined the alternating sum of the edge weights of C i.e.*

$$c_w(C) = \left|\sum_{i=0}^{2k-1} (-1)^i w(e_i)\right|$$

The definition of circulations is independent of the edge we start with because we take the absolute value of the alternating sum. Below we show a property for cycles in a graph having nonzero circulations lead to a unique minimum weight perfect matching.

References

[Ajt83] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

[Bus87] S. R. Buss. The boolean formula value problem is in alogtime. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 123–131, New York, NY, USA, 1987. Association for Computing Machinery.

[Cay49] A. Cayley. Sur les déterminants gauches. (suite du mémoire t. xxxii. p. 119). 1849(38):93–96, 1849.

[GL18] Moses Ganardi and Markus Lohrey. A universal tree balancing theorem. *ACM Trans. Comput. Theory*, 11(1), oct 2018.

[Kas04] P. W. Kasteleyn. Dimer Statistics and Phase Transitions. *Journal of Mathematical Physics*, 4(2):287–293, 12 2004.

[KR90] Richard M. KARP and Vijaya RAMACHANDRAN. Chapter 17 - parallel algorithms for shared-memory machines. In JAN VAN LEEUWEN, editor, *Algorithms and Complexity*, Handbook of Theoretical Computer Science, pages 869–941. Elsevier, Amsterdam, 1990.

[MC89] David A. Mix Barrington and James Corbett. On the relative complexity of some languages in nc1. *Information Processing Letters*, 32(5):251–256, 1989.

[Pad11] David Padua, editor. *Ajtai–Komlós–Szemerédi Sorting Network*, pages 16–16. Springer US, Boston, MA, 2011.

[SV84] Larry Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, 1984.

[Vol99] Heribert Vollmer. *Relations to Other Computation Models*, pages 35–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.