**Siddharth Joshi** **CS 289A Midterm Problem Set** May 10, 2019

## 1: Isomorphic Graphs

*Problem Statement:*
Alice and Bob each have an undirected graph on n vertices. How much communication do they need to find out deterministically if their graphs are isomorphic?

*Analysis:*
The problem of Graph Isomorphism naturally seems very similar to the Equality function: a function we are very familiar with. Moreover, due to the inherent difficulty in accurately deciphering the pattern in the characteristic matrix for Graph Isomorphism, the approach taken was to reduce Equality of $n^2$ bits to the the problem of Isomorphism of Simple Undirected Graphs on n vertices.

*Definitions:*
Let $\mathbf{G_n}$ = the set of simple undirected graphs on n vertics
Let $IS_n : \mathbf{G_n} \times \mathbf{G_n} \to \{0, 1\}$ be the graph isomorphism function i.e. 1 if the Graph x is an isomorphism of Graph y and 0 otherwise

*Claim.*
$D(IS_n) \geq n^2$

*Proof.* Reduction from equality of $n^2$ bit strings to isomorphism of simple undirected graphs on n vertices. (Note in the reduction can only use local work i.e. local work is free as Alice and Bob are computationally unbounded entities)
Show that the number of unique simple undirected graphs under isomorphism on n vertices are $\theta 2^{(n^2)}$ and therefore each can be represented using $\theta(n^2)$ bits. Therefore the $n^2$ bits of equality can be interpreted as graphs $\in \mathbf{G_n}$ and isomorphism $\leftrightarrow$ equality.

## 2: Integer Multiplication

*Problem Statement:*
How much deterministic communication does it take to compute the n least significant bit of the product of two natural numbers, of which Alice has one and Bob the other?

*Analysis:*
Product of n bits strings (do not need to consider entire number)

*Definitions:*
Let $Product_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be the the function on n bits strings indicating the nth least significant bit of the product of the integers represented by the strings

*Claim.*
logn bound using powers of 2 being distinct s

*Claim.*
$D(Product_n) \geq n + 1$ (maybe plus 1 look at the inequality)

*The kernel has always dimension* $< 2^{n-1}$ *vectors therefore by rank nullity*

proof for the above statement:

Argue that vectors of the desired form span the nullspace

Show that there can exist only $< 2^{n-1}$ such linearly independent vectors and hence nullity $< 2^{n-1}$

Use rank nullity and hence rank $> 2^{n-1}$

Thus conclude.

*rank* $> 2^{n-1} \implies$ *log rank* $>$ *n - 1* $\implies$ *ceiling log rank* $\geq$ *n and hence D(f)* $\geq$ *1*

## 3: Boolean Formulas

*Problem Statement:*
A Boolean formula in variables z1, . . . , zn is a fully parenthesized expression with operands $z1, \neg z1, ..., zn, \neg zn$ and operators $\wedge$ and $\vee$. Let $\phi(z1, ..., zn)$ be a Boolean formula in which every variable occurs exactly once. Prove that computing $\phi(x \oplus y$ deterministically on input $x, y \in 0, 1^n$ requires $\phi(n)$ bits of communication.

*Analysis:*
Describe the boolean function and the requirements it imposes

*Definitions:*
Let $Boolean_n : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$

*Claim.*
$D(Boolean_n) = n + 1$

*Proof.* For each bit there are two strings that differ in value of just that bit and have different outputs thus must be separated by any protocol

Thus there exist n such distinct partitioning cuts (note that the partitions must necessarily be distinct as they are not this implies that the 2 strings corresponding to bit i differ in bit j as well - contradiction)

Any valid protocol must induce these n partitions (as for any given bit, it is impossible for Alice or Bob to know precisely which side of the partition the final answer lies in with just their input for all possible input strings)

To induce these n distinct partitions any protocol must send atleast n bits and hence the deterministic commmunication complexity of any such boolean formula is exactly n + 1 bits.

## 4: Jazzy Inner Product

*Problem Statement:*
Define $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ by $f(x,y) = 1 \iff \sum x_i \cdot y_i = 0 \bmod (18181)$. What is the nondeterministic communication complexity of f?

*Analysis:*
compare to known problem

*Definitions:*
Let $JIP_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$

*Claim.*
$N(JIP_n) = \Omega(n)$

*Proof.* Maximal rectangles -¿ cross of the boolean vectors in subspace A of dimension n - k and subspace B of dimension k

Number of boolean vectors in A = $2^{n-k}$ and number of boolean vectors in B = $2^k$ therefore total number of elements in any rectangle $\leq 2^{n-k} \cdot 2^k = 2^n$

Identical proof here on forth

## 5: Relative Primality

*Problem Statement:*
Alice and Bob's inputs are integers a and b, respectively, where $a, b \in [1, 2^n]$. Prove that $\Theta(n/logn)$ bits are necessary and sufficient to verify nondeterministically that a and b are relatively prime.

*Analysis:*
...

*Definitions:*
Let $RP_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$

*Claim.*
$N(RP_n) = \Omega(n/log(n))$

*Proof.* Use prime number theorem

Not really sure how to progress from there, going to write program to see matrices, but need to come up with protocol in order to prove upper bound

## 6: Orthogonal Subspaces

*Problem Statement:*
On input linear subspaces $A, B \subseteq F_2^n$, prove that $\Theta(n^2)$ bits of nondeterministic communication are necessary and sufficient to check if A and B are orthogonal.

*Analysis:*
$N^2$ bits for the representation of the Subspace

*Definitions:*
Let $OS_n : \{0,1\}^{n^2} \times \{0,1\}^{n^2} \rightarrow \{0,1\}$

*Claim.*
$N(OS_n) = \Omega(n^2)$

*Proof.* Large fooling set

The fooling set contains all vector spaces and their orthogonal complement.

By definition of orthogonal complement, it is the subspace containing all vectors orthogonal to a susbspace and hence any other all crosspoints will equal 0 (thus a valid fooling set)

Number of subspaces of $\mathbb{F}_2{}^n = 1 + \binom{2^n}{n-1} + ... + \binom{2^n}{1} + 1$

Largest term is by far second term as terms reduce in magnitude now as they $n << 2^{n-1}$

Therefore fooling set $= \Omega\binom{2^n}{n-1} = \Omega(2^{n^2})$

Therefore $N(OS_n) = \Omega(log(2^{n^2})) = \Omega(n^2)$

And by the trivial protocol of sending the entire input it is easy to see that this bound is tight

## 7: Communication v/s Randomness

*Problem Statement:*
Prove that any randomized protocol for EQn with probability of correctness 2/3 and communication cost c must use more than $log_2(n/c)$ bits of randomness.

*Analysis:*
say something

*Definitions:*
Let $EQ_n : \{0,1\}^{n^2} \times \{0,1\}^{n^2} \to \{0,1\}$

Let P be a particular randomized protocol for $EQ_n$ with cost c with probability of correctness = 2/3

Let $B_P$ = number of random bits used by protocol P

*Claim.*
$B_P \geq log(n/c)$

*Proof.*

## 8: Better Than Random

*Problem Statement:*
Prove that every $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ has a randomized protocol with constant cost and error at most $1/2 - \Theta(2^{-n/2})$

*Analysis:*
Thoughts

*Definitions:*
Let $EQ_n : \{0,1\}^{n^2} \times \{0,1\}^{n^2} \to \{0,1\}$

Let P be a particular randomized protocol for $EQ_n$ with cost c with probability of correctness = 2/3

Let $B_P$ = number of random bits used by protocol P

*Claim.*
$B_P \geq log(n/c)$

*Proof.*

## Appendix: Python Script used for Empirical Data for Rank in Various Fields

```
from sage.all import *
    num_of_bits = int(raw_input("How many bits?\n"))
    m_f = []
    for x in range(0, 2**num_of_bits):
        l1 = []
        for y in range(0, 2**num_of_bits):
            z = x * y
            binary = ("{0:b}".format(z))[::-1]
            if (len(binary) < num_of_bits):
                l1.append(0)
            else:
                l1.append(int(binary[num_of_bits - 1]))
        m_f.append(l1)
    field_num = 2**(num_of_bits - 1)
    print(2*field_num)
    matrix_f = (matrix(GF(2), m_f))
    print(matrix_f.rank())
    matrix_f = (matrix(GF(2**num_of_bits), m_f))
    print(matrix_f.rank())
    matrix_f = (matrix(GF(2**(num_of_bits+1)), m_f))
    print(matrix_f.rank())
    matrix_f = (matrix(m_f))
    print(matrix_f.rank())
    kernel_f = (kernel(matrix_f)).matrix()
    print(kernel_f)
    hypothesis = True
    row_num = 0
    for row in kernel_f:
        pos = 0
        sum = 0
        abs_sum = 0
        digits = 0
        for element in row:
            sum += ((pos*element))
            abs_sum += abs((pos * element))
            pos += 1
            if element != 0:
                digits += 1
        if sum != 0:
            hypothesis = False
            break
        print(str(row_num) + ":" + str(digits) + " " + str(abs_sum))
        row_num += 1
    print(hypothesis)
```