Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Multiparty Communication Complexity

(Master thesis)

František Ďuriš

Study programme: 9.2.1. Informatics
Supervisor: prof. RNDr. Pavol Ďuriš, CSc.

Bratislava, 2011

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. František Ďuriš
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický

Názov : Multiparty Communication Complexity

Cieľ : Main objective of this thesis is to investigate the star model of the multiparty communication complexity. Second objective is to familiarize with results and techniques for proving lower bounds on the two party model in order to derive new results for the multiparty star model mainly concerning the so called hard and very hard functions.

Vedúci : doc. RNDr. Pavol Ďuriš, CSc.

Spôsob sprístupnenia elektronickej verzie práce:
dočasne neprístupná, po uplynutí bez obmedzenia

Dátum zadania: 13.10.2010

Dátum schválenia: 03.11.2010

prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

študent

vedúci práce

Dátum potvrdenia finálnej verzie práce, súhlas s jej odovzdaním (vrátane spôsobu sprístupnenia)

vedúci práce

I hereby proclaim, that I worked out this master thesis alone and using only quoted sources.

.....

Author would like to thank his supervisor prof. RNDr. Pavol Ďuriš, CSc. for his kind help and guidance.

Dedicated to Alice.

Why is a raven like a writing-desk?

Lewis Carroll

Abstract

In 1979 Yao proposed a simple model for studying communication complexity. In the first chapter we investigate the lower bound methods for this model - Fooling set method and Rank method. We prove the bounds on the number of functions with given communication complexity that can be exposed be these two methods. We give new proof that most of the functions are hard.

In the second chapter we focus on the multiparty extension of Yao's model, more precisely on the "number in the hand" model. Inspired by previous results concerned with communication complexity bounds for functions linked together with logic operators we fill the bounds for missing operators.

In the third chapter we propose a modification of this multiparty model and prove some results pointing at differences between the original and the modified model.

Keywords: communication complexity, bounds, hard functions

Abstrakt

V roku 1979 Yao navrhol jednoduchý model na skúmanie komunikačnej zložitosti. V prvej kapitole sa venujeme dvom metódam na dokazovanie spodných hraníc komunikačnej zložitosti - Fooling set metóde a Rank metóde. Dokážeme hranice pre počet funkcií s danou komunikačnou zložitosťou, ktorá je dokázateľná pomocou daných dvoch metód. Tiež podávame nový dôkaz, že takmer všetky funkcie sú ťažké.

V druhej kapitole sa zameriavame na viacúčastnícke rozšírenie tohto modelu, konkrétne skúmame "number in the hand" model. Inšpirovaní predošlými výsledkami týkajúcimi sa odhadov zložitosti funkcií spojených logickými operátormi, zúplňujeme tieto výsledky pre ostatné bežné logické spojky.

V tretej kapitole navrhujeme modifikáciu tohto viacúčastníckeho modelu a dokazujeme nejaké vlastnosti poukazujúce na rozdiely medzi pôvodným modelom a jeho modifikáciou.

Kľúčové slová: komunikačná zložitosť, ohraničenia, ťažké funkcie

Contents

Chapter 1

Prologue

The White Rabbit put on his spectacles. "Where shall I begin, please your Majesty?" he asked.

"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll
Alice's Adventures in Wonderland

Consider the following situation. You have a backup server for some network. Once in a week you run a backup procedure and upload important files from the network to this sever. But in order to save storage space and network resources you want to update only those files on the server that had been modified during the last week. How can you with 100% accuracy check whether you have up-to-date files on the server? Can it be done with less information transmitted than just sending the whole file to the server for it to decide?

This is a problem when (at this point intuitively understood) communication complexity arises in somewhat obvious way. But many other problems quite different in nature than the one stated above can be viewed from the viewpoint of communication complexity and eventually solved easier than through methods related to their own fields. Determining the bounds for

number of states in finite automata or time-space tradeoffs in Turing machines just to mention a few examples.

So, to be more precise now, communication complexity studies the amount of communication bits needed to be exchanged between two (or more) parties in order to evaluate some Boolean function of two (or more) variables. The basic two-party model was defined by Andrew Yao in 1979 [Yao79]. In this model, there are two parties Alice and Bob and their task is to compute the function $f(x, y)$, where x is known only to Alice and y is known only to Bob. Here we focus solely on the amount of communication bits exchanged and we are oblivious to the computational resources that Alice and Bob might need (that is we consider them to have unlimited power in this sense).

Even in its simplicity this model captures many of the fundamental issues related to the complexity of communication and, as we have mentioned before, results proven in this model can be often extended to more complicated scenarios. For example C. D. Thompson revealed the connection to VLSI (Very-Large-Scale Integration), more precisely to Area-Time tradeoffs for VLSI chips [Tho79] and P. Miltersen discovered the connection with data structures [Mil94].

The base model can be subsequently modified in many ways. We can consider probabilistic protocols (that is, each message sent is a probabilistic function of sender's input and of the communication so far, and protocol is allowed to make errors), variable partition models (where the parties are allowed to choose the partition of the $2n$ input bits for a given function f, n bits for each party), nondeterministic protocols (where messages sent are chosen nondeterministically and only existence of such execution of protocol that leads to the correct answer is required) etc ....

A more complete survey can be found in [Kus97] and [KN97].

1.1 Yao's Model

In this section we formally describe the two-party model[1]. As we have mentioned before one of the most important features of this model is its simplici-

[1]Based on the definitions from [Kus97], [KN97] and [Yao79]

ty. It considers a situation where there are only two communication parties which we will call Alice and Bob. Together they wish to evaluate a Boolean function $f(x, y) : \{0,1\}^n \times \{0,\ 1\}^n \to \{0,\ 1\}$ where the inputs x and y are two n-bit strings such that x is known only to Alice and y is known only to Bob. This model is not concerned with any computational resources needed by Alice and Bob to achieve their goal and focuses solely on the communication exchanged between them. Some examples of such functions include:

- **equality** $EQ(x, y)$ is defined to be 1 if $x = y$ (and 0 otherwise).
- **inner-product** $IP(x, y)$ is defined as $\sum_{i=1}^{n} (x_i y_i) \mod 2$, where $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$.
- **greater-than** $GT(x, y)$ is defined to be 1 if $x > y$ (and 0 otherwise), where x and y are viewed as numbers written in binary code.
- **disjointness** $DISJ(x, y)$ is defined to be 1 if there is no index i such that $x_i = y_i = 1$ (and 0 otherwise).

The functions mentioned above are again very simple yet natural so we can get a good understanding of what can be done with them.

The computation of a function is done using a communication protocol. During the execution of the protocol, the communicating parties alternate roles in sending messages where each of these messages are strings of bits. We require the messages to be self-delimiting. Protocol determines who sends a message and what is the content of such message. The protocol also specifies when the execution terminates in which case it also specifies the output.

A communication protocol P computes a function $f(x, y)$ if for every input pair $(x, y) \in \{0,\ 1\}^n \times \{0,\ 1\}^n$ the protocol terminates with the value $f(x, y)$ as its output.

It is easy to observe that every function can be computed by a trivial protocol that will send all bits that Alice possess to Bob who can now compute the function and send the result to Alice. However, we will consider n to be large thus making this protocol very expensive. There are many functions that can be computed using much less amount of communication

but, unfortunately, many more of those which cannot be computed with less communication than n.

The study of communication complexity aims at identifying the minimal number of bits that must be sent in order to compute the function f. More precisely, the complexity measure for a protocol P is the worst case number of bits exchanged by the two parties. Formally, let $s_P(x,y) = \{m_1, m_2, \ldots m_r\}$ be the communication exchanged on the input (x,y) during the execution of P, where m_i denotes the i-th message sent in the protocol. Also denote by $|m_i|$ the number of bits of m_i and let $|s_P(x,y)| = \sum_{i=1}^{r} |m_i|$. We now define the (deterministic) communication complexity of P as the worst case number of bits exchanged by the protocol. That is,

$$D(P) = \max_{(x,y)\in\{0,\ 1\}^n\times\{0,\ 1\}^n} |s_P(x,y)|$$

The (deterministic) communication complexity of a function f is the communication complexity of the best protocol that computes f. That is

$$D(f) = \min_{P:P \text{ computes } f} D(P)$$

By using the trivial protocol we described above we get

$$\forall f : \{0,\ 1\}^n \times \{0,\ 1\}^n \to \{0,\ 1\} \quad D(f) \leq n+1$$

Functions that cannot be computed with less communication then n are called hard. Moreover, hard functions whose complement functions are also hard are called very hard.

1.2 Lower Bounds

In the previous chapter we have shown a trivial upper bound for any function f. Generally, proving upper bounds of functions consists of finding some smart protocol and proving its complexity. On the other hand, to prove a lower bound of a given function, one must prove that any solution (a solution that we can assume nothing about it except the fact that it solves the problem) has at least a certain communication complexity.

While the upper bounds tell us how much time (or resources or whatever) we need to compute a given function, lower bounds tell us how good our

solution can possibly be. That is, if lower and upper bound is the same, then we have the best solution to be had. If they do not meet, then we know that there is a room for improvement.

Here we show two widely used lower bound techniques but first we need some definitions and lemmas. We will analyze so called rectangles - a combinatorial structures imposed by protocols.

Definition 1.2.1. [Kus97] A rectangle is a subset of $\{0,1\}^n \times \{0,1\}^n$ of the form $A \times B$, where each of A and B is a subset of $\{0,1\}^n$. A rectangle $R = A \times B$ is called f-monochromatic if for every $x \in A$ and $y \in B$ the value of $f(x,y)$ is the same.

To imagine this think of a matrix of size $2^n \times 2^n$ with numbered rows and columns such that each row with number x and column with number y corresponds to an input pair (x,y). A set A from the definition above is then a subset of rows and B is a subset of columns. Note that a rectangle induced by A and B does not have to have adjacent rows and columns. An example of such rectangle is on Figure 1.

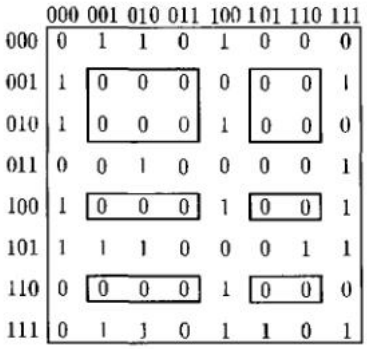


Figure 1

Lemma 1.2.2. *[Kus97] Let P be a protocol that computes a function f and $(m_1, \ldots m_r)$ be a sequence of messages. The set of inputs (x,y) for which $s_P(x,y) = (m_1, \ldots m_r)$ is an f-monochromatic rectangle.*

Definition 1.2.3. [Kus97] For a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ we define $C^P(f)$ as the minimum number of f-monochromatic rectangles that partition the space of inputs $\{0,1\}^n \times \{0,1\}^n$.

Lemma 1.2.4. *[Kus97] For every function* $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ *:*

$$D(f) \geq \log_2 C^P(f).$$

Proof. By the lemma 1.2.2, every protocol P partitions the space of inputs into f-monochromatic rectangles. The number of these rectangles is equal to the number of possible communications and that is at most $2^{D(P)}$. Since $D(f) \leq D(P)$ it follows that $C^P(f) \leq 2^{D(f)}$ and finally $D(f) \geq \log_2 C^P(f)$. □

So according to the above lemma, instead of proving lower bounds on the communication complexity we can prove the lower bounds on minimal number of f-monochromatic rectangles that partition the space of inputs.

Remark 1.2.5. [KN97] Note that although every protocol partition the input space into monochromatic rectangles, not every partition corresponds to some protocol. An example of such partition is on the Figure 2.

	y	y'	y''
x	1	0	0
x'	1	1	1
x''	0	0	1

Figure 2

Consider any protocol P for computing the function defined by Figure 2. Since the function is not constant, there must be a first player who sends a message that is not constant. Suppose that this player is Alice. Since the messages that Alice sends on x, x', x'' are not all the same, there are two possibilities: (1) her message on x and x' are different. In this case the rectangle $\{x, x'\} \times \{y\}$ is not a monochromatic rectangle induced by the protocol P; or (2) her messages on x' and x'' are different. In this case the rectangle $\{x\} \times \{y', y''\}$ is not a monochromatic rectangle induced by the protocol P. Similar analyses applies if Bob sends the first message.

1.2.1 Fooling Set Method

This method was used implicitly in [Yao79] and made more explicit in [LS81]. It is based on finding such a set of input pairs, that none of them can be in the same f-monochromatic rectangle.

Definition 1.2.6. [Kus97] A set of input pairs $\{(x_1, y_1), ..., (x_l, y_l)\}$ is called a fooling set (of size l) with respect to f if there exists $b \in \{0, 1\}$ such that

1. For all i $f(x_i, y_i) = b$
2. For all $i \neq j$ $f(x_i, y_j) \neq b$ or $f(x_j, y_i) \neq b$

Lemma 1.2.7. *[Kus97] If there exists a fooling set of size l with respect to f then*

$$D(f) \geq \log_2 l$$

For the illustration purposes we state the full proof from [Kus97].

Proof. By Lemma 1.2.4, it suffices to prove that $C^P(f) \geq l$. For this, we prove that in any partition of $\{0,1\}^n \times \{0,\ 1\}^n$ into f-monochromatic rectangles the number of rectangles is at least l. Suppose that the number of f-monochromatic rectangles is smaller than l. In this case, there exists two pairs in the fooling set (x_i, y_i) and (x_j, y_j) that belong to the same rectangle $A \times B$. This implies that $x_i, x_j \in A$ and $y_i, y_j \in B$. By the definition of fooling set $f(x_i, y_i) = f(x_j, y_j) = b$ while at least one of $f(x_i, y_j)$ and $f(x_j, y_i)$ is different than b. This implies that the rectangle $A \times B$ is not f-monochromatic. □

We can apply this method to determine the lower bound for the communication complexity of equality function $EQ(x, y)$. One can easily see that the set $\{(x, x) \mid x \in \{0, 1\}^n\}$ is a fooling set of size 2^n. Thus the EQ function is hard.

For the function $DISJ(x, y)$ the fooling set of size 2^n is $\{(A, A^c) \mid A \subseteq \{1, ..., n\}\}$. Hence the function $DISJ$ is also hard.

1.2.2 Rank Method

The second method is based on algebra and more specifically on ranks of matrices. We have previously associated the space of inputs with matrix $2^n \times 2^n$ where rows correspond to inputs x and columns to inputs y and

each (x, y) entry of this matrix is equal to $f(x, y)$. We shall call this matrix M_f (note that any zero-one matrix $2^n \times 2^n$ defines some function). The following lemma that relates the communication complexity of f to the rank of matrix M_f appeared in [MS82].

Lemma 1.2.8. *[MS82] Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a function. Then*

$$D(f) \geq \log_2 \mathit{rank}(M_f).$$

The proof is from [Kus97].

Proof. By Lemma 1.2.2, it suffices to prove that $C^P(f) \geq \text{rank}(M_f)$. Given an optimal cover of $\{0,1\}^n \times \{0,1\}^n$ with f-monochromatic rectangles, let $R_1, ..., R_t$ be those rectangles which are 1-monochromatic. It is enough to prove that $t \geq \text{rank}(M_f)$. For each of the rectangle R_i $(1 \leq i \leq t)$ define a $2^n \times 2^n$ matrix M_i whose (x, y) entry is 1 if $(x, y) \in R_i$ and 0 otherwise. It follows that

$$M_f = \sum_{i=1}^{t} M_i.$$

Since the rank is sub-additive, we get

$$\text{rank}(M_f) \leq \sum_{i=1}^{t} M_i.$$

Since R_i is a 1-monochromatic rectangle then for every i, $\text{rank}(M_i) = 1$. Hence, $\text{rank}(M_i) \leq t$, as desired. □

Remark 1.2.9. [AB09] Note that as 0,1 are elements of every field, we can compute the rank of the matrix M_f over any field. Some fields may yield lesser communication complexity and some greater. Therefore the choice of a field is crucial.

Remark 1.2.10. [KN97] What we get from the above proof is actually a lower bound on the number of 1-rectangles. By switching to a function $f' = 1 - f$ and a matrix $M_{f'} = ||1_{ij}||_{2^n \times 2^n} - M_f$ we can get similar estimate of 0-rectangles for the function f. It is easy to see that the ranks of matrices M_f

and $M_{f'}$ differ by at most 1.

$$\begin{aligned} rank(M_f) &= rank\left((1)_{2^n\times 2^n} - M_{f'}\right) = \\ &= rank\left((-1)_{2^n\times 2^n} + M_{f'}\right) \leq \\ &\leq \ rank\left((1)_{2^n\times 2^n}\right) + rank(M_{f'}) \end{aligned}$$

$$rank(M_f) - rank(M_{f'}) \leq = rank((1)_{2^n\times 2^n}) = 1$$

We used here that $rank(cM) = rank(M)$ where c is some nonzero constant in the field over which we compute the rank.

Therefore

$$D(f) \geq \log_2 (2\text{rank}(M_f) - 1).$$

Again a few examples. The matrix corresponding to the function EQ is the indentity matrix $I_{2^n\times 2^n}$ whose rank is of course 2^n.

The matrix corresponding to the inner-product function IP is a so-called Hadamard matrix whose rank is known to be $2^n - 1$.

1.3 Combinatorics of the Rank Method

We have previously defined hard functions as functions with maximal communication complexity. That is, no protocol that computes such function is better than the trivial one. We have also seen two methods for proving lower bounds on communication complexity. Now the question of how good these methods are might arise. Firstly we will focus on their ability to expose the hard functions. So how can we compare them in this sense?

One possible way might be to count the number (or give a reasonably good lower bound) of hard functions and the number of hard functions that can be detected by these two criteria. We can then consider the limit of ratio of these numbers (for fooling set method and rank method) and number of all hard functions as n (the length of the binary input strings) approaches infinity.

Let us start with enumerating those hard functions that can be exposed by the rank method. Whenever we mention a matrix in the following text we always mean a matrix over the field $\mathbb{Z}_2$.

We will start with the estimation of the number of hard functions that can be exposed with this method. That is, the number of those functions where $\text{rank}(M_f) = 2^n$ - denote this number as $R(2^n)$ (actually $\text{rank}(M_f) \geq 2^{n-2} + 1$ is sufficient but for pedagogical reasons we start with this result first).

The procedure of constructing matrices with full rank is simple. We start with the first row - here we have $2^{2^n} - 1$ possibilities as we do not want a row with zeros only (a row which is linearly dependent with anything). Otherwise anything is acceptable. As for the second row, compared to the first row we have one less possibility - which is the first row. For the third row we can use anything except zero-row, first and second row and moreover any linear combination of them (but since we work over the field $\mathbb{Z}_2$ there is only one such combination so far). It is clear that this construction covers all matrices and that it is deterministic, thus unique.

It is good to realize that in general when we consider a linear combination of k vectors $c_1\alpha_1 + c_2\alpha_2 + ... + c_k\alpha_k$, each of the coefficients c_i is either 1 or 0 therefore any such combination of k vectors can be described by a binary vector $\beta = (c_1, ..., c_k)$. Moreover, if the vectors $\alpha_1, ..., \alpha_k$ are linearly independent then any two different linear combinations give two different vectors. This allows us to effectively count the number of all linear combinations of vectors $\alpha_1, ..., \alpha_k$ - the number is 2^k.

Having said this we can write down the possibilities for each row

$$\begin{aligned}
&\text{first row :} \quad 2^{2^n} - 1 \\
&\text{second row :} \; 2^{2^n} - 1 - 1 \\
&\text{third row :} \quad 2^{2^n} - 1 - 2 - 1 \\
&\text{fourth row :} \; 2^{2^n} - 1 - 3 - 3 - 1 \quad \text{etc.}
\end{aligned}$$

Hence

$$R(2^n) = \prod_{i=1}^{2^n} \left(2^{2^n} - \sum_{j=0}^{i-1} \binom{i-1}{j}\right) = \prod_{i=1}^{2^n} \left(2^{2^n} - 2^{i-1}\right). \tag{1.1}$$

Let N_{all} be the number of all functions $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$.

It is easy to see that $N_{all} = 2^{2^{2n}}$. Now lets compare $R(2^n)$ with N_{all}:

$$\lim_{n\to\infty} \frac{\prod_{i=1}^{2^n}\left(2^{2^n} - 2^{i-1}\right)}{2^{2^{2n}}} = \lim_{n\to\infty} \frac{\left(2^{2^n}\right)^{2^n}\prod_{i=1}^{2^n}\left(1 - \frac{1}{2^i}\right)}{2^{2^{2n}}} = \lim_{n\to\infty} \prod_{i=1}^{2^n}\left(1 - \frac{1}{2^i}\right)$$

It is known that if $\{a_i\}_{i=1}^{\infty}$ is a sequence of positive numbers and the series $\sum_{i=1}^{\infty} a_i$ converges then the product $\prod_{i=1}^{\infty}\left(1 - a_i\right)$ converges to some non-zero number. Therefore $\lim_{n\to\infty}\prod_{i=1}^{2^n}\left(1 - \frac{1}{2^i}\right)$ exists and its value can be numerically determined as $0.288788\ldots$. Thus

$$\lim_{n\to\infty} \frac{\prod_{i=1}^{2^n}\left(2^{2^n} - 2^{i-1}\right)}{2^{2^{2n}}} = 0.288788\cdots > \frac{1}{4}$$

This proves that randomly chosen 0-1 matrix over the field $\mathbb{Z}_2$ has full rank with probability at least $\frac{1}{4}$ (for sufficiently large n).

We have previously mentioned that for a function f to be hard the rank of the matrix M_f does not have to be 2^n. According to the Remark 1.2.10 the rank$(M_f) \geq 2^{n-2} + 1$ is sufficient because

$$D(f) \geq \log\left(2(2^{n-2}+1) - 1\right) = \log\left(2^{n-1} + 1\right) = (n-1) + \log\left(1 + \frac{1}{2^{n-1}}\right)$$

and since $D(f)$ has to be a natural number we have $D(f) \geq n$.

Obviously, the number of functions with rank at least $2^{n-2}+1$ is greater than the number of functions with rank precisely 2^n. Thus we proved the following:

Theorem 1.3.1. *Randomly chosen Boolean function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is hard with probability at least $\frac{1}{4}$.*

According to Remark 1.2.9 the choice of a field over which we calculate the rank can be crucial. It was proved by Komlós in [Kom67] and [Kom68] that randomly chosen 0-1 matrix $n \times n$ has full rank over the field $\mathbb{Q}$ with probability approaching 1 as n tends to infinity. Thus

Theorem 1.3.2. *Let $0 < \alpha < 1$ be a real number. Then for sufficiently large n the probability that randomly chosen function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is hard is at least α.*

Moreover, for sufficiently large n the rank method can expose (in other words can be successfully used to determine tight lower bound on communication complexity of) almost all hard functions.

At the end of this section we will show that this theorem holds over the field $\mathbb{Z}_2$ as well.

Now let us continue in our analysis. How many matrices of the size $2^n \times 2^n$ and with some small rank (now again over the field $\mathbb{Z}_2$) are there? For example, a matrix with rank 1 can have at most two different kinds of rows. A non-zero one and maybe some zero-rows too. This gives us

$$R(1) = \left(2^{2^n} - 1\right) \sum_{k=1}^{2^n} \binom{2^n}{k} = \left(2^{2^n} + 1\right)^2 = 2^{2^{n+1}} - 2^{2^n+1} + 1$$

possible matrices. A small number compared to $2^{2^{2n}}$.

If $R(k)$ will denote the number of matrices of size $2^n \times 2^n$ with rank k, it is interesting to ask for which value k the numbers $R(k)$ and N_{all} are comparable. And in this direction our analysis follows - how many matrices are there with rank $\log n$ or $\sqrt{n}$ etc.?

Definition 1.3.3. We say that vectors (or rows in our case) $\alpha_1, ..., \alpha_m$ are linearly dependent if there exist index j and non-zero vector $\beta = (c_1, c_2, ..., c_{j-1})$ such that $\alpha_j = c_1\alpha_1 + c_2\alpha_2 + ... + c_{j-1}\alpha_{j-1}$. We say that vectors $\alpha_1, ..., \alpha_m$ are linearly independent if they are not linearly dependent.

Note that this definition of linearly independent vectors is equivalent to that which does not consider which vector is first, second etc.

Matrices we are interested in have k linearly independent rows and for such positioning we have $\binom{2^n}{k}$ possibilities (here we are not concerned with which vectors lie in these rows except that they are linearly independent). These rows define a submatrix of the size $k \times 2^n$ with rank k and the equation (1.1) gives us the number of all acceptable vector assignments. Hence we have

$$\binom{2^n}{k} \prod_{i=1}^{k} (2^{2^n} - 2^{i-1})$$

possibilities for choosing linearly independent vectors and their positions in matrix M_f.

Now we need to select the rest of the rows. The k independent rows create altogether at most $k + 1$ gaps between each of them or above/below the first/last row.

Let $\alpha_{i_1}, ..., \alpha_{i_k}$ be linearly independent vectors and i_j is number of the row in which the vector α_{i_j} lies. Then

$$M_f = \begin{pmatrix} \text{gap n. } 0 \\ \alpha_{i_1} \\ \text{gap n. } 1 \\ \alpha_{i_2} \\ \dots \\ \text{gap n. } k-1 \\ \alpha_{i_k} \\ \text{gap n. } k \end{pmatrix}$$

The number of all rows that lie in these gaps is clearly $2^n - k$. We will fill the l-th gap with linear combinations of vectors $\alpha_{i_1}, ..., \alpha_{i_{l-1}}$ (the set of linearly independent vectors that lie above the l-th gap). Note that the gap n. 0, if exists, must have only zero-only rows.

Now we need to show that we can construct each matrix M_f in this way and that such construction is unique.

Let M_f be any matrix with rank k. Let Λ be an empty set. Let $\{i_1, ..., i_k\}$ be a set of indices and $z = 1$. Go through all rows from top to bottom (let i be the index of current row). Vector in the row i is either

1. a linear combination of the vectors in the set Λ or zero-only vector. In this case do nothing.

2. a vector linearly independent with vectors in the set Λ. In this case add the vector in this row to Λ, set $i_z = i$ and $z = z + 1$.

After this procedure is finished the set Λ contains k linearly independent vectors (any other number would contradict the fact that the rank of the matrix M_f is k). Moreover, every vector either lies in Λ or is a linear combination of the vectors that lie higher in the matrix M_f and are in Λ. Specially, vectors that lie above the vector α_{i_1} are zero-only vectors.

Hence every matrix M_f can be constructed in our way.

Uniqueness. This also follows from the construction above. Because if we could construct some matrix in two different ways then they would have to agree at least on the number of zero-only rows on the top of the matrix

M_f. But then, they would also have to agree on the first linearly independent vector and on the index of its row. Since the procedure is deterministic, there is no way in which the computation would split and continued in two separate ways.

Now that we have showed how to construct every matrix of the size $2^n \times 2^n$ with the rank k we can calculate the number of such matrices. We have previously showed that to select k rows and k linearly independent vectors we have

$$\binom{2^n}{k} \prod_{i=1}^{k} (2^{2^n} - 2^{i-1})$$

possibilities. For the l-th gap ($l \in \{0, ..., k\}$) we have $\sum_{i=0}^{l} \binom{l}{i} = 2^l$ different vectors this gap can contain as this is the number of all linear combinations (including zero-only vector) of some $l-1$ linearly independent vectors (we do not care which these linearly independent vectors are because we have selected them before). And if the size (number of rows) of the l-th gap is i_l then it follows that

$$\sum_{\substack{(i_0,...,i_k) \\ i_0+...+i_k=2^n-k}} \prod_{j=0}^{k} (2^j)^{i_j}$$

is the number of all permissible gaps between the vectors $\alpha_1, ..., \alpha_k$.

But note that by selecting the positions of k linearly independent vectors we also defined the numbers $i_0, ..., i_k$ from the $\sum_{\substack{(i_0,...,i_k) \\ i_0+...+i_k=2^n-k}}$. And vice versa, by selecting the numbers $i_0, ..., i_k$ we defined the positions of the linearly independent vectors. Thus

Theorem 1.3.4. *The number of matrices of the size $2^n \times 2^n$ with rank k is*

$$R(k) = \left(\prod_{i=1}^{k} (2^{2^n} - 2^{i-1}) \right) \left(\sum_{\substack{(i_0,...,i_k) \\ i_0+...+i_k=2^n-k}} \prod_{j=0}^{k} (2^j)^{i_j} \right). \tag{1.2}$$

The equation (1.2) does not present us with a reasonable way to determine the number of matrices with the rank, say, n or $\log n$ etc. Therefore

we may want to try to simplify the equation even at the cost of obtaining only some bound.

Firstly, for the left part of the equation (1.2) we can write

$$\frac{1}{4}2^{k2^n} \leq \binom{2^n}{k} \prod_{i=1}^{k} (2^{2^n} - 2^{i-1}) \leq \frac{1}{2}2^{k2^n}$$

Right part of the equation (1.2) will be a bit more complicated. Firstly, we would like to get rid of the $\sum$ - we already argued above that the number of summands in this sum is $\binom{2^n}{k}$. But we can count this number in another way as well: In how many ways can we partition $2^n - k$ rows into $k + 1$ ordered groups when we permit empty groups as well?

We start with an easier task - in how many ways can we write some number n as a sum of non-zero numbers? We can think of the number n as n dots in a row. To write this number as a sum of non-zero numbers is equal to placing delimiters between the dots.

For one summand we have just one option. For two summands we have $n - 1$ positions to place a delimiter - we want only non-zero summands therefore the beginning and the end of the row of dots is not an acceptable place for a delimiter. That makes $\binom{n-1}{1}$ possibilities. In general, for $i + 1$ summands ($i \in \{1, ..., n - 1\}$) we have $\binom{n-1}{i}$ possibilities, especially for n summands ($i = n - 1$) we need to place delimiters in all $n - 1$ positions and for this we have only one option.

Thus to write a number n as a sum of non-zero numbers there are

$$\sum_{j=0}^{n-1} \binom{n-1}{j} = 2^{n-1}$$

possibilities.

Now to get back to the original task. If $2^n - k = 1$ - that is, we have one row which we want to divide between $k + 1$ groups. For this we have $\binom{k+1}{1}$ possibilities - we just need to choose into which group we put this row. If $2^n - k = 2$ then either we put 2 rows in one group - $\binom{k+1}{1}$ possibilities - or we separate these rows by placing them in two different groups - $\binom{k+1}{2}$ possibilities.

Consider $2^n - k = 3$. We have learnt that to write number 3 as 1,2 or 3 summands we have respectively $\binom{2}{0}$, $\binom{2}{1}$ and $\binom{2}{2}$ possibilities. Plus for each

number of summands we need to allocate the appropriate number of groups. Thus we have altogether

$$\binom{2}{0}\binom{k+1}{1}+\binom{2}{1}\binom{k+1}{2}+\binom{2}{2}\binom{k+1}{3}$$

possibilities.

In general, if we have 2^n-k rows which we want to divide between $k+1$ groups then we have

$$\sum_{j=0}^{2^n-k-1}\binom{2^n-k-1}{j}\binom{k+1}{j+1} \tag{1.3}$$

possibilities.

k - the rank of the matrices of size $2^n\times 2^n$ we are trying to count - ranges from 0 to 2^n-1. If $k=2^n$ then the matrices have full rank and the right part of the equation (1.2) technically does not exist - the above equation yields an empty sum. Nevertheless we shall consider the number of these nonexisting options as 1 - because in this case the equation (1.2) is

$$R(2^n)=\prod_{i=1}^{2^n}\left(2^{2^n}-2^{i-1}\right)$$

what is precisely the number of matrices with full rank we derived before.

Lets try another value. If $k=0$ then, according to what was said above, there are no linearly independent rows - hence all rows must be zero-only rows. And there is only one such matrix.

$$\sum_{j=0}^{2^n-1}\binom{2^n-1}{j}\binom{1}{j+1}=\binom{2^n-1}{0}\binom{1}{1}=1$$

Lets make a small diversion here. It is known ([GKP94]) that if l,m,n are integers and $l\geq 0$ then

$$\sum_i\binom{l}{m+i}\binom{s}{n+i}=\binom{l+s}{l-m+n} \tag{1.4}$$

The terms on the left side of this equation are nonzero for i

$$\max\{-m,-n\}\leq i\leq\min\{l-m,s-n\}.$$

Now compare this binomial identity with the equation (1.3). In this case

$$\begin{aligned} m &= 0 \\ n &= 1 \\ l &= 2^n - k - 1 \\ s &= k + 1. \end{aligned}$$

Moreover, one can easily check that the terms in the equation (1.3) are nonzero for $0 \leq j \leq \min\{2^n - k - 1, k\}$. Thus we can use the identity (1.4)

$$\sum_{j=0}^{2^n-k-1} \binom{2^n - k - 1}{j} \binom{k+1}{j+1} = \binom{2^n}{2^n - k} = \binom{2^n}{k}. \tag{1.5}$$

Lets check small cases. If $k = 0$ then identity (1.5) holds : $1 = 1$. If $k = 1$ then (1.3) holds as well - $2^n = 2^n$. For $k = 2$ we have

$$1\binom{3}{1} + \binom{2^n - 3}{1}\binom{3}{2} + \binom{2^n - 3}{2} = 3 + 2\binom{2^n - 3}{1} + \binom{2^n - 3}{1} + \binom{2^n - 3}{2} =$$

$$= 3 + 2\binom{2^n - 3}{1} + \binom{2^n - 2}{2} = 3 + \binom{2^n - 3}{1} + \binom{2^n - 2}{1} - 1 + \binom{2^n - 2}{2} =$$

$$= 2 + \binom{2^n - 3}{1} + \binom{2^n - 1}{2} = \binom{2^n - 1}{1} + \binom{2^n - 1}{2} = \binom{2^n}{2}$$

If $k = 2^n - 1$ then (1.5) holds again : $2^n = 2^n$. We once again got the result that

$$\sum_{\substack{(i_0,\ldots,i_k) \\ i_0+\ldots+i_k=2^n-k}} 1 = \binom{2^n}{k}$$

So we know the number of summands but we still know nothing about the indices i_j.

$$\sum_{\substack{(i_0,\ldots,i_k) \\ i_0+\ldots+i_k=2^n-k}} \prod_{j=0}^{k} (2^j)^{i_j} = \sum_{\substack{(i_0,\ldots,i_k) \\ i_0+\ldots+i_k=2^n-k}} 2^{\lg \prod_{j=0}^{k} 2^{ji_j}} = \sum_{\substack{(i_0,\ldots,i_k) \\ i_0+\ldots+i_k=2^n-k}} 2^{\sum_{j=0}^{k} ji_j}$$

$$2^n - k \;\leq\; \sum_{j=0}^{k} j i_j \;\leq\; (2^n - k)\, k$$

$$\binom{2^n}{k} 2^{2^n - k} \;\leq\; \sum_{\substack{(i_0,\ldots,i_k) \\ i_0+\ldots+i_k = 2^n - k}} \prod_{j=0}^{k} (2^j)^{i_j} \;\leq\; \binom{2^n}{k} 2^{(2^n-k)k}$$

Putting both estimates together we get

Theorem 1.3.5 (Alice's theorem)**.**

$$R(k) \;\geq\; \frac{1}{4} 2^{k2^n} \binom{2^n}{k} 2^{2^n - k} \tag{1.6}$$

$$R(k) \;\leq\; \frac{1}{2} 2^{k2^n} \binom{2^n}{k} 2^{(2^n-k)k} \tag{1.7}$$

Notice that the function $R(k)$ is increasing - there are more matrices with rank k than there are with rank $k-1$.

Now that we have more user friendly equations we can have a look at the number of matrices with some specific rank - in this way we can count the probability that randomly chosen function has some specific communication complexity. For example, how many matrices $2^n \times 2^n$ with the rank n are there - that is, functions with at least logarithmic communication complexity? We think the following bounds for n sufficiently big:

$$R(n) \;\geq \frac{1}{4} \binom{2^n}{n} 2^{n2^n} 2^{2^n - n} \geq 2^{n2^n}$$
$$R(n) \;\leq\; \frac{1}{2} \binom{2^n}{n} 2^{n2^n} 2^{(2^n-n)n} \leq 2^{2n2^n}$$

The bound is, in a way, asymptotically tight. Moreover, we know that function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ is hard when $\operatorname{rank}(M_f) \geq 2^{n-2} + 1$. Consider the limit

$$\lim_{n\to\infty} \frac{2^{2^{2n}} - \sum_{i=1}^{2^{n-2}} R(i)}{2^{2^{2n}}} \geq \lim_{n\to\infty} \frac{2^{2^{2n}} - 2^{n-2} R(2^{n-2})}{2^{2^{2n}}} \geq$$

We used the fact that the function $R(i)$ is increasing.

$$\geq \lim_{n\to\infty} \frac{2^{2^{2n}} - 2^{n-2}\ \frac{1}{2}2^{2^{n-2}2^n}\binom{2^n}{2^{n-2}}2^{(2^n-2^{n-2})2^{n-2}}}{2^{2^{2n}}} \geq$$

$$\geq \lim_{n\to\infty} \frac{2^{2^{2n}} - 2^{n-2}\ \frac{1}{2}2^{\frac{2^{2n}}{4}}\binom{2^n}{2^{n-2}}2^{\frac{3}{16}2^{2n}}}{2^{2^{2n}}} \geq$$

$$\geq \lim_{n\to\infty} \frac{2^{2^{2n}} - 2^{\frac{7}{16}2^{2n}}2^{O(n2^n)}}{2^{2^{2n}}} = \lim_{n\to\infty}\left(1 - \frac{2^{O(n2^n)}}{c^{2^{2n}}}\right) = 1 \quad (c>1)$$

We used the fact that $\frac{1}{2}2^{n-2}\binom{2^n}{2^{n-2}} = 2^{O(n2^n)}$.

In other words almost all functions are hard and exposable by the rank method. This is another proof of the Theorem 1.3.2 now using only the field $\mathbb{Z}_2$.

1.4 Combinatorics of the Fooling Set Method

Now we turn to the fooling set method. We would like to prove similar results as we have proved before for the rank method in order to compare them. Previously we denoted the number of matrices with rank k as $R(k)$. In this spirit we will denote the number of matrices with fooling set of size at least k as $F(k)$.

Say we have a hard function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ and a fooling set $F = \{(x_1, y_1), ..., (x_l, y_l)\}$ with respect to f (the indices $i_k, j_k,\ k = 1..l$, correspond to the coordinates of the input pair (x_{i_k}, y_{j_k}) in the matrix M_f). Since f is hard it follows from Lemma 1.2.7 that $l \geq 2^{n-1}+1$. But lets start with more simple case when $l = 2^n$ as the ideas we demonstrate here will prove very useful later.

Observe that no two elements in F can be in the same row or same column (otherwise such two elements would contradict the properties of the fooling set, namely the second condition). Since $l = 2^n$, each column and each row of the matrix M_f contains exactly one element from the set F.

Now consider any two elements from the set F, say (x_i, y_j) and (x_k, y_l) $(i \neq k,\ j \neq l)$. These two elements have by the definition of the fooling set the value $b \in \{0, 1\}$. What about the elements (x_i, y_l) and (x_k, y_j) (call this pair of elements of matrix M_f a complementary pair to the pair (x_i, y_j) and (x_k, y_l))? Again, by the definition of the fooling set, we have exactly three ways of assigning values to these two elements (that is $(1-b, 1-b), (1-b, b), (b, 1-b)$; the value (b, b) contradicts the definition of the fooling set). Note that in this case each element of matrix M_f is either an element of F or is part of some complementary pair with respect to F.

If we now realize that any function f is fully given by her matrix M_f, we can enumerate (and count) all hard functions that have fooling set of the size 2^n. Firstly we spot all elements from the fooling set F. We can think of this as a spotting of 2^n rooks on a chessboard of the size $2^n \times 2^n$ in a way that no two rooks threaten each other. Secondly we assign a single value to all rooks (the value b from the definition of the fooling set) and the values to the complementary pairs of all pairs of elements of F. By doing so we define the matrix M_f.

Thus the number of all possibilities is (choose b, spot the rooks, choose values for complementary pairs)

$$F(2^n) = 2 * (2^n)! * 3^{\binom{2^n}{2}}$$

The following example shows that it is possible to create one matrix more than once, hence what we have got is actually an upper bound

$$F(2^n) \leq 2 * (2^n)! * 3^{\binom{2^n}{2}}$$

and it is therefore not so nice result compared to the rank method, where we derived the exact number. ("Fooling set matrices" seems to be somehow harder to fully grasp.) Let $\Lambda = \{(1,1), (2,3), (3,2), (4,4)\}$ and $\Delta = \{(1,2), (2,1), (3,3), (4,4)\}$ and $b = 1$

$$M_f = \begin{pmatrix} 1_\lambda & 1_\delta & 0 & 0 \\ 1_\delta & 0 & 1_\lambda & 0 \\ 0 & 1_\lambda & 1_\delta & 0 \\ 0 & 0 & 0 & 1_{\delta,\lambda} \end{pmatrix}$$

We can derive a simple lower bound as well. Obviously, if only the elements from F had value b in the matrix M_f and the rest of the elements had value $1-b$ then each different set F corresponds to a different matrix M_f. So, the repetitions are due to the term $3^{\binom{2^n}{2}}$ - choices for complementary pairs. Let us limit these choices. Previously we allowed all of the following possibilities (the so called complementary pairs are in bold):

$$\begin{pmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ b & \dots & \mathbf{1\text{-}b} \\ \vdots & \vdots & \vdots \\ \mathbf{1\text{-}b} & \dots & b \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ b & \dots & \mathbf{b} \\ \vdots & \vdots & \vdots \\ \mathbf{1\text{-}b} & \dots & b \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ b & \dots & \mathbf{1\text{-}b} \\ \vdots & \vdots & \vdots \\ \mathbf{b} & \dots & b \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{pmatrix}$$

$$\begin{pmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ \mathbf{1\text{-}b} & \dots & b \\ \vdots & \vdots & \vdots \\ b & \dots & \mathbf{1\text{-}b} \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ \mathbf{b} & \dots & b \\ \vdots & \vdots & \vdots \\ b & \dots & \mathbf{1\text{-}b} \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ \mathbf{1\text{-}b} & \dots & b \\ \vdots & \vdots & \vdots \\ b & \dots & \mathbf{b} \\ \vdots & \vdots & \vdots \\ \dots & \dots & \dots \end{pmatrix}$$

Now we allow only the first two choices from each row (value b can be only in the top corner) - call the respective fooling set a fooling set with limited choices. We will show that in this case we removed all repetitions. Assume, on the contrary, that there is a matrix M that corresponds to two different fooling sets Λ and Δ and look at the last row of M. If Λ and Δ differ in the element from the last row then

$$M = \begin{pmatrix} \dots & \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & b_\lambda & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & b_\lambda & \dots & b_\delta & \dots \end{pmatrix}$$

But this contradicts the allowed choices because the element b_δ is in the bottom corner. Therefore Λ and Δ agree on the element from the last row.

Next consider the second last row. By the similar argument we have that Λ and Δ must agree here as well.

$$M = \begin{pmatrix} \dots & \dots & \dots & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & \dots & \dots & b_\lambda & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & b_\lambda & \dots & b_\delta & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

Finally one can prove by the induction that $\Lambda = \Delta$ what is a contradiction that they are different. Hence the lower bound on $F(2^n)$ follows

$$F(2^n) \geq 2 * (2^n)! * 2^{\binom{2^n}{2}}$$

Before we move on, consider the following limit

$$\lim_{n\to\infty} \frac{F(2^n)}{N_{all}} \leq \lim_{n\to\infty} \frac{2 * (2^n)! * 3^{\binom{2^n}{2}}}{N_{all}} = \lim_{n\to\infty} \frac{2^{O(n2^n)} 3^{\frac{2^n(2^n-1)}{2}}}{2^{2^{2n}}} =$$

$$= \lim_{n\to\infty} \frac{2^{O(n2^n)}}{\sqrt{3}^{2^n}} \left(\frac{\sqrt{3}}{2}\right)^{2^{2n}} = \lim_{n\to\infty} \frac{2^{nO(2^n)}}{c^{2^n O(2^n)}} = 0 \quad (c > 1)$$

We used here the fact that $(2^n)! = 2^{O(n2^n)}$.

This limit proves that the number of functions with full fooling set is small compared to the number of all functions. Moreover, from the last line we can see that this number decreases at least exponentially.

We argued before that no two elements from the fooling set can be in the same column or same row. This means that if we have a fooling set F of size k, we have k columns and k rows each with one element from F. If we now look at all elements of the matrix M_f that lie on the intersections of such rows and columns, we get a submatrix $A_{k\times k}$ of the matrix M_F (elements of such submatrix A may not have adjacent elements in the original matrix M_f).

Now lets get back to the case where $l \geq 2^{n-1} + 1$. We see that in this case we need to choose the submatrix $A_{(2^{n-1}+1)\times(2^{n-1}+1)}$ (which is equal to

choosing $2^{n-1}+1$ rows and columns) and assign values to the elements of this submatrix in the way we sketched above (that is, in such a way that we will create a fooling set of size $2^{n-1}+1$). The rest of the matrix M_f can be chosen arbitrarily. Therefore the upper bound (we include lots of repetitions here) for all possibilities is

$$F(2^{n-1}+1) \leq \binom{2^n}{2^{n-1}+1}^2 * 2 * (2^{n-1}+1)! * 3^{\binom{2^{n-1}+1}{2}} * 2^{2^{2n}-(2^{n-1}+1)^2}$$

We can again define a simple lower bound as well. If we consider only those matrices that have a submatrix $A_{(2^{n-1}+1)\times(2^{n-1}+1)}$ with the fooling set with limited choice of size $2^{n-1}+1$ fixed in their top left corner

$$M_f = \begin{pmatrix} A & B \\ C & D \end{pmatrix}_{2^n \times 2^n}$$

we get a lower bound on the number of matrices with fooling set of size at least $2^{n-1}+1$.

$$F(2^{n-1}+1) \geq 2 * (2^{n-1}+1)! * 2^{\binom{2^{n-1}+1}{2}} * 2^{2^{2n}-(2^{n-1}+1)^2}$$

Again we can compare the bounds for $F(2^{n-1}+1)$ with the number N_{all}.

$$\lim_{n\to\infty} \frac{F(2^{n-1}+1)}{N_{all}} \leq \lim_{n\to\infty} \frac{\binom{2^n}{2^{n-1}+1}^2 2(2^{n-1}+1)! 3^{\binom{2^{n-1}+1}{2}} 2^{2^{2n}-(2^{n-1}+1)^2}}{2^{2^{2n}}} \leq$$

$$\leq \lim_{n\to\infty} \frac{2\ 2^{2n(2^{n-1}+1)} \sqrt{3}^{2^{n-1}(2^{n-1}+1)} 2^{2^{2n}-2^{2n-2}-1}}{2^{2^{2n}}} \leq$$

$$\leq \lim_{n\to\infty} \frac{2^{n2^n+n+2^{n-2}\lg 3} \left(\sqrt[8]{3}\sqrt[4]{2}^3\right)^{2^{2n}}}{2^{2^{2n}}} \leq$$

$$\leq \lim_{n\to\infty} \frac{2^{nO(2^n)}}{c^{2^n O(2^n)}} = 0 \quad (c>1)$$

Thus the number of functions with the fooling set of the size at least $2^{n-1}+1$ (in other words the number of hard functions exposable by the fooling set method) dwindle at least exponentially with n growing.

Theorem 1.4.1. *Let $0 < \epsilon < 1$ be a real number. Then for sufficiently large n the probability that randomly chosen function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ has a fooling set of size at least $2^{n-1}+1$ is at most ϵ.*

Theorems (1.3.2) and (1.4.1) tells us that almost no hard functions have large fooling sets. Therefore, the fooling set method is weaker than rank method in this probabilistic sense. On the other hand almost all functions have a fooling set of constant size (with respect to n) - compare this with the fact that almost no functions have constant rank. This probabilistic "complementarity" of fooling set method and rank method is indeed interesting.

In the following step of our analysis we focus our attention to find a correlation between the size k of a fooling set F and some upper bound of $F(k)$ (the number of matrices with the fooling set of size at least k). We have witnessed the generalization step going from $k = 2^n$ to $k = 2^{n-1}+1$ and to go to arbitrary k is indeed simple:

Theorem 1.4.2 (Bob's theorem)**.**

$$F(k) \leq \binom{2^n}{k}^2 * 2 * k! * 3^{\binom{k}{2}} * 2^{2^{2n}-k^2}$$
$$F(k) \geq 2 * k! * 2^{\binom{k}{2}} * 2^{2^{2n}-k^2}$$

Lets use this bounds to estimate the number of functions with the fooling set with size at least n - that is, functions with at least logarithmic communication complexity that can be exposed by the fooling set method.

$$F(n) \geq 2\ n! 2^{\binom{n}{2}} 2^{2^{2n}-n^2} \geq 2^{n\lg n - n\lg e}\ 2^{\frac{1}{2}n^2-\frac{1}{2}n}\ 2^{2^{2n}-n^2} \geq 2^{2^{2n}-n^2}$$

$$F(n) \leq \binom{2^n}{n}^2 2\ n! 3^{\binom{n}{2}} 2^{2^{2n}-n^2} \leq 2^{2n^2 - n\lg n + n\lg e}\ 3^{\frac{1}{2}n^2-\frac{1}{2}n}\ 2^{2^{2n}-n^2} = 2^{2^{2n}+O(n^2)}$$

Compare this with our previous result

$$R(n) \geq 2^{n2^n}$$
$$R(n) \leq 2^{2n2^n}.$$

1.5 Other known results

Here we mention some results from [DHS96]. They are of similar nature.

We will denote the communication complexity a function f as $cc(f)$.

Definition 1.5.1. [DHS96] Let f be a Boolean function. For an arbitrary field F with identity elements 0 and 1, let $Rank_F(f)$ denote the rank of the matrix M_f over F. We define

- $Rank(f) = \max\{Rank_F(f) | F \text{ is a field with identity elements 0 and 1}\}$
- $r(f) = \lceil \log_2(Rank(f)) \rceil$

Definition 1.5.2. [DHS96] Let f be a Boolean function of $2n$ variables. For $b \in \{0,1\}$, a set

$$A(f) = \{(\alpha_1, \beta_1), ..., (\alpha_k, \beta_k)\},\ \alpha_i, \beta_i \in \{0,1\}^n,\ i = 1, ..., k$$

is called a b-fooling set for f if

1. $f(\alpha_i, \beta_i) = b$ for all $i \in \{1, ..., k\}$, and
2. $i \neq j,\ i, j \in \{1, ..., k\}$ implies that $f(\alpha_i, \beta_j) \neq b$ or $f(\alpha_j, \beta_i) \neq b$.

We define

- $Fool(f) = \max\{card(A(f)) |\ A \text{ is a b-fooling set for f and } b \in \{0,1\}\}$
- $fs(f) = \lceil \log_2(Fool(f)) \rceil$

Theorem 1.5.3. *[DHS96] (i)If $n \in \mathbb{N}$ is sufficiently large then for at least a fraction of $\frac{1}{4}$ of Boolean functions f of $2n$ variables the following holds:*

- $Fool(f) \leq 10n$ *(i.e.,* $fs(f) \leq \log_2 n + \log_2 10$*), and*
- $Rank_{\mathbb{Z}_2}(f) = 2^n$ *(i.e.,* $r(f) = cc(f) = n$*).*

(ii) Almost all Boolean functions f of $2n$ variables satisfy $Rank(f) = 2^n$ and $Fool(f) \leq 10n$.

Part (ii) of this theorem shows that the Rank method is exponentially better than the Fooling set method for almost all functions; part (i) shows that this is true for a substantial number of functions even if only rank over $\mathbb{Z}_2$ is used.

Theorem 1.5.4. *[DHS96] For all Boolean functions f and all fields F*

$$Fool(f) \leq (Rank_F(f) + 1)^2 \quad (i.e., \ fs(f) \leq 2r(f) + 2)$$

Furthermore, we consider the function $g_{2n}(x_1, ..., x_n, y_1, ..., y_n) = \sum_{i=1}^{n} x_i y_i \mod 2$, the inner product over $\mathbb{Z}_2$. The family $\{g_{2n}\}$ provides a specific example for which the Rank method is exponentially better than the Fooling Set method.

Theorem 1.5.5. *[DHS96] For every $n \in \mathbb{N}$ we have*

1. *$Rank(g_{2n}) = 2^n - 1$, and*
2. *$Fool(g_{2n}) = (n+1)^2$.*

Thus, $fs(g_{2n}) \leq \lg_2(n+1)$ and $r(g_{2n}) = n$

It was also shown that there is a function for which the Fooling Set method is better than the Rank method.

Theorem 1.5.6. *[DHS96] There is an algorithm that, for any $n = 4^m$, $m \in \mathbb{N}$, constructs a Boolean function h_{2n} of $2n$ variables such that*

1. *$Fool(h_{2n}) = 2^n$, and*
2. *$Rank(h_{2n}) = 3^{\frac{n}{2}}$.*

Thus, $0.79... \cdot n = \frac{1}{2} \log_2 3 \cdot n = r(h_{2n}) < fs(h_{2n}) = n = cc(h_{2n})$

Chapter 2

Multiparty Communication Complexity

"Curiouser and curiouser!"

Lewis Carroll
Alice's Adventures in Wonderland

In the previous chapter we dealt with two-party communication model. One natural way to extend this model is to allow more parties and distribute the input equally among them. But unlike in the two-party model, here the parties will not be able to communicate between each other. Instead, there will be a coordinator, an entity which communicates with parties and directs the computation. Also we do not require for parties to know the output of such computation, only coordinator is entitled to this knowledge.

This model was studied in [DF89], [DF92], [Ďď04] and others. The lower bound technique (a generalization of fooling set method) appeared in [ĎR98].

Another lower bound method is that of partition arguments: partition all processors into two disjoint sets and find a lower bound for the induced two-party communication complexity problem. Some results concerned with

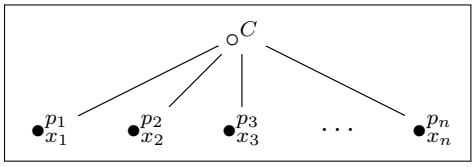
this method can be found in [DKW09].

The model we sketched above is often called "Number in the hand" model. There is also another one, called "Number on the forehead" model. In this model the input is distributed in such a way, that the party p_i sees the whole input except the part x_i. The parties then communicate through the shared "blackboard" (broadcast). This model was introduced in [CFL83] and appeared in [BNS89] where an interesting relation to time-space trade-offs and branching programs were discovered. Note that in this model the partition arguments method will not work as any two processors already know the whole input.

In this chapter we will consider the "Number in the hand" model. We shall start with definitions.

2.1 Definitions and basic properties

The definitions were adopted from [Ďď04]. The schema of the model is following:



Let ϵ be the empty string and let $w = w_1\$w_2\$...\$w_l, l \geq 1, w_i \in \{0,1\}^+$ for every i. We define : $h(\epsilon) = \epsilon$ and $h(w) = w_1w_2...w_l$. Let $r = (r_1, r_2, ..., r_t)$, $t \geq 1$, where either $r_i = r_i^1\$r_i^2\$...\$r_i^{j_i}, r_i^l \in \{0,1\}^+, j_i \geq 1$, or $r_i = \epsilon$. We define: $h(r) = h(r_1)h(r_2)...h(r_t)$. We denote the lenght of a string w (the cardinality of a set S) by $|w|$ (by $|S|$). If S is a set then by $h(S)$ we denote the set $\{h(s)|s \in S\}$.

Suppose a coordinator wishes to evaluate a function $f(x_1, x_2, ..., x_n)$. The input vector $x = (x_1, x_2, ..., x_n)$ is distributed among n parties (i.e., the processors $p_1, p_2, ..., p_n$), with x_i known only to party i, where x_i is chosen from $\{0,1\}^m$ for every i. Suppose there is a deterministic protocol P that accepts the language defined by f (when the value of f is 1). In such a case we will

say that P computes f. Generally, the computation of P consists of several phases, where one phase is as follows: The coordinator sends some messages (nonempty binary strings) to some parties (not necessary to all parties) and then, each party that got a message, sends a message back to the coordinator. The communication behaviour of P can be described by a communication vector $s = (s_1, s_2, ..., s_n)$, where either $s_i = s_i^1\$s_i^2\$...\$s_i^{j_i}, j_i \geq 2, s_i^l \in \{0,1\}^+$, or $s_i = \epsilon$; s_i is a communication sequence between the coordinator and the party i (if there is no communication then $s_i = \epsilon$). Note that j_i is an even number (each party must response after obtaining any nonempty message), and $s_i^{2l-1}[s_i^{2l}]$ is not necessary the message sent [received] by the coordinator in the phase l (since the coordinator may have sent no message to the party i in some previous phase $k < l$). We will also say "communication sequence on the link i" instead of "communication sequence between the coordinator and the party i". Also we will say "processor p_i" instead of "party i".

Formally, a deterministic protocol P is an $(n+1)$-tuple of functions $(\phi_0, \phi_1, ..., \phi_n)$, for which the following holds: Let

$$K = \{0,1,\$\}^* \times ... \times \{0,1,\$\}^* \quad \text{(n times)}$$

$$M = \{0,1\}^* \times ... \times \{0,1\}^* \quad \text{(n times)}$$

(a) ϕ_0 is a function from K to $M \cup \{$"accept", "reject"$\}$. Intuitively, behaviour of the coordinator is given by ϕ_0, where the argument of ϕ_0 is a communication vector of all previous messages, with \$ serving as a delimiter between messages. The result of ϕ_0 are either the next messages sent to the parties or the coordinator stops the communication and accepts/rejects the input.

(b) For $i = 1, 2, ..., n$, ϕ_i is a function from $\{0,1\}^m \times \{0,1,\$\}^*$ to $\{0,1\}^+$. Intuitively, behaviour of the party $i, (1 \geq i \geq n)$, is given by ϕ_i, where the first argument of ϕ_i is the local input for the party i and the second argument of ϕ_i is a sequence of all previous messages on the link i (delimited by \$). The result of ϕ_i is the next message sent by the party i to the coordinator.

A computation under P on input $x = (x_1, ..., x_n)$ with $x_i \in \{0,1\}^m$ for each i is a communication vector $s^k = (s_1^k, ..., s_n^k)$, where $k \geq 0$, (k is the number of all phases performed on x under P), such that for every

$j = 0, 1, 2, ..., k-1$ there is a communication vector $s^j = (s_1^j, ..., s_n^j)$, ($s^j$ is the communication vector after completing the j-th phase of the computation on x under P), for which (c), (d), (e) hold.

(c) $s_i^0 = \epsilon$ for $i = 1, 2, ..., n$; (the coordinator starts the communication with the empty communication vector $s^0 = (\epsilon, ..., \epsilon)$).

(d) For $j = 0, 1, 2, ..., k-1$ it holds: Let $\phi_0(s^j) = (d_1^j, ..., d_n^j)$. Then $s_i^{j+1} = s_i^j \$ d_i^j \$ \phi_i(x_i, s_i^j \$ d_i^j)$ if $d_i^j \neq \epsilon$, otherwise $s_i^{j+1} = s_i^j$ for $i = 1, 2, ..., n$.

(e) $\phi_0(s^k) \in \{$"accept", "reject"$\}$. If $\phi_0(s^k) =$"accept" [$\phi_0(s^k) =$"reject"] then s^k is an accepting [rejecting] computation vector under P, (or, s^k is an accepting [rejecting] computation under P on x)

We require that nonempty communication sequences on each link are self-delimiting, i.e., if $s_i = s_i^1 \$ s_i^2 \$ ... \$ s_i^{j_i}$ and $r_i = r_i^1 \$ r_i^2 \$ ... \$ r_i^{l_i}$ are any two different nonempty communication sequences on the link i under P, and if $s_i^1 = r_i^1, ..., s_i^q = r_i^q$ for some $q \geq 0$, then $q \leq \min\{j_i, l_i\}$ and s_i^{q+1} is not any proper prefix of r_i^{q+1}, or vice versa. (Note that one can easy show that then $h(s_i) \neq h(r_i)$ and $h(s_i)$ is not any proper prefix of $h(r_i)$, or vice versa).

In fact, we do not need the "end of transmission" symbol, "$", because of the self-delimiting property (introduced in [PS82]). We use this property, since we want to pin down exactly the communication complexity.

Let $f(x_1, ..., x_n)$ be a Boolean function with $x_i \in \{0,1\}^m$ for each i, and P be a deterministic protocol. We say that P computes f if, for each $x = (x_1, ..., x_n)$ with $x_i \in \{0,1\}^m$ for each i, the computation under P on the input x is an accepting one iff $f(x) = 1$.

Let S be the set of all accepting and rejecting communications vectors under P. By $DC(f)$ we denote the maximum over all $s \in S$ of $|h(s)|$ minimized over all deterministic protocols computing f. $DC(f)$ is called the deterministic communication complexity of f.

We also consider nondeterministic protocols. In such a case, ϕ_i's are "nondeterministic functions", i.e., they may have several values (and therefore they are not functions). Moreover, they may be "partial nondeterministic functions", i.e., they may not be defined for all possible values of arguments; in such a case, the current communication is aborted. We can apply the definitions above also for nondeterministic protocol in such a way that whenever we write $\phi_0(s)$ [$\phi_i(x, s),\ i > 0$] we mean a possible value of

$\phi_0(s)$ [of $\phi_i(x,s)$]. We require the self-delimiting property also for nondeterministic protocols.

Let $f(x_1,...,x_n)$ be a Boolean function with $x_i \in \{0,1\}^m$ for each i, and P be a nondeterministic protocol. We say that P computes f if, for each $x = (x_1,...,x_n)$ with $x_i \in \{0,1\}^m$ for each i, there is an accepting communication under P on input x iff $f(x) = 1$.

Let A be the set of all accepting communication vectors under a nondeterministic protocol P. By $C(f)$ we denote the maximum over all $s \in S$ of $|h(s)|$ minimized over all nondeterministic protocols computing f. $C(f)$ is called the nondeterministic communication complexity of f.

We have seen that the trivial upper bound for two-party communication complexity was $n+1$ if the length of input X (and of Y) was n. The similar upper bound for nondeterministic multiparty communication follows.

Theorem 2.1.1. *[Ďď04] $C(f) \leq nm$ for each Boolean function $f(x_1,...,x_n)$ with $x_i \in \{0,1\}^m$ for $i = 1,2,...,n$.*

Proof. For each f under consideration, there is a nondeterministic protocol P computing f using nm exchanged bits as follows. Given input $(x_1,...,x_n)$ the coordinator nondeterministically guesses the first bit of x_i and sends it to p_i. Then each p_i responds the rest of x_i if the guess was successful, otherwise it aborts the communication. If all guesses were successful then the coordinator knows the whole input, hence it can accept the input correctly. □

For deterministic protocols we have this upper bound.

Theorem 2.1.2. *$DC(f) \leq n(m+1)$ for each Boolean function $f(x_1,...,x_n)$ with $x_i \in \{0,1\}^m$ for $i = 1,2,...,n$.*

Proof. For each f under consideration, there is a deterministic protocol P computing f using $n(m+1)$ exchanged bits as follows. Given input $(x_1,...,x_n)$ the coordinator sends some one-bit signal message (be it 0 or 1, it does not matter) to all p_i. Then each p_i responds its input x_i. Thus coordinator knows the whole input and it can accept it correctly. □

For the two-party model, we defined the notion of hard functions. Similarly we can introduce multiparty hard functions. But now we can distinguish between deterministic and nondeterministic hard functions. On the

other hand, if a function f is hard in nondeterministic sense, then it is hard also in deterministic sense (while the other implications does not have to hold). Therefore we will consider mostly nondeterministic hard functions.

Definition 2.1.3. Let $f(x_1, ..., x_n)$ be a Boolean function with $x_i \in \{0,1\}^m$ for $i = 1, 2, ..., n$. If $C(f) \geq nm$ then we will call such function hard. Moreover, if for some hard function f the function $f' = 1 - f$ is also hard, then we call the function f (and obviously f' too) very hard.

Note that $DC(f) = DC(1 - f)$ for all functions. Therefore if function f is deterministically hard it is automatically deterministically very hard. On the other hand this does not hold in nondeterministic sense.

We have seen that almost all functions for two-party model were hard. It is not therefore surprising that similar result holds for multiparty model as well.

Theorem 2.1.4. *[Ďo04] For each integer $n \geq 3$ and for each real number α, $0 < \alpha < 1$, there is a positive integer m such that randomly chosen Boolean function $f(x_1, ..., x_n)$ with $x_i \in \{0,1\}^m$ for $i = 1, 2, ..., n$ is very hard with probability α.*

Some examples of hard and very hard functions are in place.

Theorem 2.1.5. *[ĎR98] Let $v(b)$ denote the integer represented by b. Let*

- $f_1(x_1, ..., x_n) = 1$ *iff* $x_1 = x_2 = ... = x_n$
- $f_2(x_1, ..., x_n) = 1$ *iff* $x_1 ... x_{\frac{n}{2}} = x_{\frac{n}{2}+1} ... x_n$ *(for n even)*
- $f_3(x_1, ..., x_n) = 1$ *iff* $v(x_1 ... x_{\frac{n}{2}}) \leq v(x_{\frac{n}{2}+1} ... x_n)$ *(for n even).*

where $x_i \in \{0,1\}^m$ for every i. Then $C(f_i) \geq nm$ for $i = 1, 2, 3$.

Note that that the first two functions are not very hard. Consider the following protocols for the functions $f_1' = 1 - f_1$ and $f_2' = 1 - f_2$.

In the first case the coordinator guesses a number k, $1 \leq k \leq m$, and numbers n_1, n_2, $1 \leq n_1, n_2 \leq n$ (in other words, he guesses which two inputs differ on which position). Thus the coordinator needs to send at most $\lceil \log_2 m \rceil + \lceil \log_2 m \rceil$ bits - binary encoded number k sent to processors n_1 and n_2 - while the answer he gets is just two bits long - one bit from each processor. Therefore the upper bound for this protocol is $2(\lceil \log_2 m \rceil + 1)$.

In the second case the protocol is basically the same. On the other hand it has been shown in [ĎR98] that $C(1 - f_3) \geq nm - 2$.

The function f_1 is rather interesting from the practical point of view (for example file servers etc.) and so more results were proven.

Theorem 2.1.6. *[ĎR98] Let $f_1(x_1, ..., x_n) = 1$ iff $x_1 = x_2 = ... = x_n$, where $x_i \in \{0,1\}^m$ for every i. The the following holds*

- $nm \leq C(f) \leq n(m+1)$
- $\lfloor \log_2 (m-1) \rfloor \leq C(1-f) \leq 2\lfloor \log_2 m \rfloor + 2$
- $nm \leq DC(f) = DC(1-f) \leq n(m+1)$

With this example we can see how much can nondeterminism help us (although not always). The gap between $C(1-f)$ and $DC(1-f)$ can be arbitrarily large.

Now consider somehow similar function

$$F(x_1, ..., x_n) = 1 \text{ iff } \forall_{i,j \in \{1,...,n\}} \ (i \neq j \Rightarrow x_i \neq x_j) \text{ where } x_i \in \{0,1\}^m$$

In other words such function that equals 1 if and only if the inputs in all processors are different.

Firstly, lets have a look at such a case where $n = 2^m$ - the amount of processors is the same as the number of all possible different inputs. In this case we have exactly $n!$ possible configurations at the beginning - all permutations of numbers $1...n$.

We claim that each permutation - each distribution of inputs - has a different communication. Assume the opposite - let $\Pi = (x_1, ..., x_j, ..., x_n)$ and $\Pi' = (x'_1, ..., x'_j, ..., x'_n)$ be two different permutations with the same communications on all links. Since the Π and Π' are different there must exists such index $j \in \{1, ..., n\}$ that $x_j \neq x'_j$. According to our assumption the communication is the same for both permutations on all links, therefore the input $(x_1, ..., x'_j, ..., x_n)$ will be accepted too - the coordinator does not see the inputs, only the communication sequences on all links - and this we did not change. But this is a contradiction because there exists such index $l \in \{1, ..., n\}/\{j\}$ that $x_l = x'_j$.

Thus, for different permutations of inputs we have a different communication and

$$C(f) \geq \lg(n!) \approx \lg\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) = \lg\sqrt{2\pi n} + n\lg n - n\lg e \geq n(m-2)$$

On the other hand, consider n to be small compared to m. In such a case the nondeterministic coordinator can(?) guess a vector $(i_1, ..., i_n),\ i_j \in \{1, ..., m\}$ - positions at which the inputs differ - and send this coordinates to appropriate processors. Processors, after receiving the message from the coordinator, send back the appropriate bit. Alas, this strategy will not succeed. Consider the following scenario

$$x_1 = 1111111$$
$$x_2 = 0111111$$
$$x_3 = 1011111$$

What are the coordinators options? Any index he chooses for the first input has value 1. For the second input he now needs to choose the second index - otherwise he would not be able to detect that $x_1 \neq x_2$. This index has value 0. For the third input, in order to detect that $x_2 \neq x_3$, the coordinator needs to choose an index with value 1. But choosing such index the coordinator is not able to detect that $x_1 \neq x_3$.

On the other hand, there are $\binom{n}{2}$ different pairs of inputs and to be able to detect that each pair of processors contains different input we need to know two bits - one bit from each processor. Hence

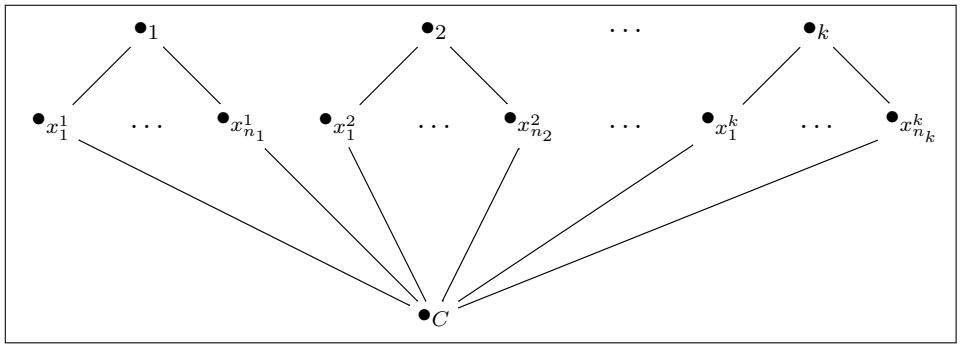
$$C(f) \leq 2\binom{n}{2}\lg m + 2\binom{n}{2} \leq n(n-1)\lg m + n(n-1)$$

2.2 Linking functions with logic operators

One of the more interesting things we can do with multiparty functions is combining them. Say we have k functions computed on k systems each with a coordinator i and its n_i processors. We want to compute the function

$$F = f_1(x_1^1, ..., x_{n_1}^1) \wedge ... \wedge f_k(x_1^k, ..., x_{n_k}^k)$$

Is it possible for some (maybe all) functions to save communication bits if instead of computing k functions separately we computed them together using only one coordinator, thus computing only one function?



Intuitively, whether this can or cannot be done will depend on whether we consider deterministic or nondeterministic protocols. Also, we can consider other logical operators such as $\vee$, $\rightarrow$, $\oplus$, $\iff$ etc.

2.2.1 Nondeterministic protocols

Theorem 2.2.1. *[ĎR98] Let $0 = n_1 < n_2 < ... < n_{k+1} = n$, $k \geq 1$, be any integers with $n_i + 2 \leq n_{i+1}$ and let f_i be an $(n_{i+1} - n_i)$-ary function with $C(f_i) > 0$ for $i = 1, 2, ..., n$. Let*

$$f(x_1, ..., x_n) = \prod_{i=1}^{k} f_i(x_{n_i+1}, ..., x_{n_{i+1}}).$$

Then

$$C(f) = \sum_{i=1}^{k} C(f_i).$$

Thus it does not matter whether we compute the functions f_i separately or somehow together, the amount of nondeterministic communication is the same.

Theorem 2.2.2. *Let $0 = n_1 < n_2 < ... < n_{k+1} = n$, $k \geq 1$, be any integers with $n_i + 2 \leq n_{i+1}$ and let f_i be an $(n_{i+1} - n_i)$-ary function with $C(f_i) > 0$*

for $i = 1, 2, ..., n$. *Let*

$$f(x_1, ..., x_n) = \max_{i \in \{1,...,k\}} \{f_i(x_{n_i+1}, ..., x_{n_{i+1}})\}.$$

Then

$$C(f) = \max_{i \in \{1,...,k\}} \{C(f_i)\}.$$

Proof. First we show that $C(f) \geq \max_{i \in \{1,...,k\}} \{C(f_i)\}$. Lets assume that $C(f_j) = \max_{i \in \{1,...,k\}} \{C(f_i)\}$ and $C(f) < C(f_j)$ for some protocol P computing the function f. We will create a protocol P' computing the function f_j with better communication complexity than $C(f_j)$.

The protocol P' computing the function f_j will simulate the protocol P in the following way. The protocol P' will start with input $(x_{n_j+1}, ..., x_{n_{j+1}})$. In the first step the protocol will nondeterministicaly guess the rest of the vector $(x_1, ..., x_n)$ in such a way that for all $j,\ j \neq i$: $f_j(x_{n_j+1}, ..., x_{n_{j+1}}) = 0$ (such inputs do exist since $C(f_i) > 0$ for all i). Then the protocol P' will follow the steps of the protocol P on the input $(x_1, ..., x_n)$ with the difference, that the only actual communication is on the links $(n_j + 1, ..., n_{j+1})$ and the rest of the communication is simulated inside the coordinator.

The protocol P' will arrive at the result

$$f(x_1, ..., x_n) = \max_{i \in \{1,...,k\}} \{f_i(x_{n_i+1}, ..., x_{n_{i+1}})\}$$

with the communication $C(f) < C(f_j)$. But since for all $j,\ j \neq i$:

$$f_j(x_{n_j+1}, ..., x_{n_{j+1}}) = 0$$

it follows that $f(x_1, ..., x_n) = f_j(x_{n_j+1}, ..., x_{n_{j+1}})$. Since the protocol P is correct by our assumption, so is the protocol P'. Thus

$$C(f) \geq \max_{i \in \{1,...,k\}} \{C(f_i)\}$$

Now we will complete the proof. Consider the following protocol P computing the function f. The protocol starts with the input $(x_1, ..., x_n)$. But instead of computing all functions f_j the protocol will nondeterministically guess such j that $f_j(x_{n_j+1}, ..., x_{n_j}) = 1$ and then compute only this function. If such j for the given input $(x_1, ..., x_n)$ exists, then this computation will eventually confirm that the guess was correct. Therefore

$$C(f) \leq \max_{i \in \{1,...,k\}} \{C(f_i)\}$$

and the proof is complete. □

Hence we can save a considerable amount of communication bits if we choose to compute the function $f(x_1, ..., x_n) = \max_{i \in \{1,...,k\}}\{f_i(x_{n_i+1}, ..., x_{n_{i+1}})\}$ with just one coordinator.

Theorem 2.2.3. *[Man97] Let $f(x_1, ..., x_k),\ g(x_{k+1}, ..., x_n)$ be Boolean functions such that $C(f)C(g) > 0$ and $C(1-f)C(1-g) > 0$. Let*

$$h(x_1, ..., x_n) = f(x_1, ..., x_k) \oplus g(x_{k+1}, ..., x_n).$$

Then

$$\begin{aligned} C(h) &\geq \max\{C(f) + C(1-g),\ C(1-f) + C(g))\} \\ C(h) &\leq \max\{C(f) + C(1-g),\ C(1-f) + C(g))\} + n. \end{aligned}$$

Therefore if we want to compute, for example, the function

$$F(x_1, ..., x_{2k}) = f(x_1, ..., x_k) \oplus f(x_{k+1}, ..., x_{2k})$$

where

$$f(x_1, ..., x_k) = 1 \text{ iff } x_1 = x_2 = ... = x_k$$

we need at most $km + 2(\lceil \log_2 m \rceil + 1)$. Since $km + 2(\lceil \log_2 m \rceil + 1) < 2km$ for $k, m > 2$, we can save some amount of communication bits.

The same applies in the following case as well.

Theorem 2.2.4. *Let $f(x_1, ..., x_k),\ g(x_{k+1}, ..., x_n)$ be Boolean functions such that $C(f)C(g) > 0$ and $C(1-f)C(1-g) > 0$. Let*

$$h(x_1, ..., x_n) = f(x_1, ..., x_k) \Leftrightarrow g(x_{k+1}, ..., x_n).$$

Then

$$\begin{aligned} C(h) &\geq \max\{C(f) + C(g),\ C(1-f) + C(1-g))\} \\ C(h) &\leq \max\{C(f) + C(g),\ C(1-f) + C(1-g))\} + 2n. \end{aligned}$$

Proof. The proof for the upper bound is basically the same as the proof for Theorem 2.2.3 from [Man97]: the coordinator guesses wether

$$\begin{aligned} (x_1, ..., x_k) &\in f^{-1}(1) \\ (x_{k+1}, ..., x_n) &\in g^{-1}(1) \end{aligned}$$

or

$$(x_1, ..., x_k) \in f^{-1}(0)$$
$$(x_{k+1}, ..., x_n) \in g^{-1}(0).$$

In the first case the coordinator sends "1" to all parties. Each party must respond after obtaining a message so all parties send back "1". After this procedure is complete, the computation follows separately for each function $f,\ g$.

In the second case, the coordinator sends "0" to all parties and all parties send back "1". After this procedure is complete, the computation follows separately for each function $1-f,\ 1-g$.

For the proof of the lower bound we can use the Theorem 2.2.3. Assume that we can compute the function $h = f \Leftrightarrow g$ with better communication complexity than $\max\{C(f) + C(g),\ C(1-f) + C(1-g))\}$.

We know that in order to compute the function $h' = f \oplus (1-g)$ we need at least

$$\max\{C(f) + C(1-(1-g)),\ C(1-f) + C(1-g))\} =$$
$$= \max\{C(f) + C(g),\ C(1-f) + C(1-g))\}$$

Since

$$h = f \Leftrightarrow g \equiv f \oplus (1-g) = h'$$

we have a contradiction:

$$C(h) < \max\{C(f) + C(g)),\ C(1-f) + C(1-g))\}$$
$$C(h') \geq \max\{C(f) + C(g),\ C(1-f) + C(1-g))\}.$$

□

Theorem 2.2.5. *Let $f(x_1, ..., x_k),\ g(x_{k+1}, ..., x_n)$ be Boolean functions such that $C(f)C(g) > 0$ and $C(1-f) > 0$. Let*

$$h(x_1, ..., x_n) = f(x_1, ..., x_k) \Rightarrow g(x_{k+1}, ..., x_n).$$

Then

$$C(h) = \max\{C(1-f),\ C(g)\}$$

Proof. This result follows from the Theorem 2.2.2 and the fact that

$$f \Rightarrow g \equiv (1 - f) \vee g.$$

The upper bound follows directly. As for the lower bound, assume that we can compute the function $h = f \Rightarrow g$ with better communication complexity than $\max\{C(1 - f),\ C(g)\}$.

We know that in order to compute the function $h' = (1 - f) \vee g$ we need at least $\max\{C(1 - f),\ C(g)\}$ communication bits. Since

$$h = f \Rightarrow g \equiv (1 - f) \vee g = h'$$

we have a contradiction:

$$C(h) < \max\{C(1 - f),\ C(g)\}$$
$$C(h') \geq \max\{C(1 - f),\ C(g)\}.$$

□

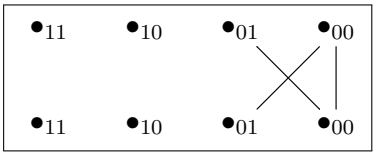
Here again we can save a great deal of communication bits by using just one coordinator.

We have seen that being able to nondeterministically guess something (e.g. input vector or output of some computation) can help us a lot. As we will see in the following section, without this ability the results are quite different.

2.2.2 Deterministic protocols

Lets for a while consider a slight modification of this model. We will restrict the coordinator in such a way, that it will not be allowed to send any information to other processors. That is, if a coordinator wishes to send some message to some party x_i it will always be one-bit signal 1. This model is sometimes called one-way as the information is allowed to flow only in one direction - from processors to the coordinator.

Let $f : \{00, 01, 10, 11\} \times \{00, 01, 10, 11\} \to \{0, 1\}$ be a function given as follows (the line between two inputs means that the function value is 1):



$$M_f = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Because the fooling set of this matrix is $\{(10, 00), (01, 01), (00, 10)\}$, the function f is hard in the two party model.

We claim that this function is hard in multiparty model too - $DC(f) = n(m+1) = 2(2+1) = 6$ (n is the number of processors and m is the length of input).

If the coordinator sends a signal to both parties, to which they respond by sending their whole input back, then 6 bits are sufficient.

Can 5 bits be enough? Firstly note that we get the most information bits if the communication has only one round, that is the coordinator sends at most two bits - one to each party. In this case we can get 3 bits of information from the processors (any more bits sent by the coordinator lessen this number).

If the protocol is correct, the coordinator would know the answer just by knowing 2 bits from, say, the first processor and 1 bit from the second processor. Since 1 bit is not enough to uniquely identify the input of the second processor, the coordinator must somehow gather this 1 bit of missing information from the bits it has got from the first processor.

The computation is deterministic and the processors do not know about each other, therefore the processor that sends only one bit is the same for all possible inputs - in our case the second (or left) processor - the second processor receives "1" from the coordinator and it does not know whether the first processor sent 1 or 2 bits - therefore neither processor can pick how many bits it sends - the first processor sends always the whole input and the second processor sends only one bit.

But this contradicts that the function f is hard in the two party model - by making the first processor virtual part of the coordinator we get a protocol for the two party model for the function f that needs to transmit only one bit (the bit from the second processor).

Theorem 2.2.6. *Let* $F(x_1, ..., x_k, ..., x_n) = f_1(x_1, ..., x_k) \wedge f_2(x_{k+1}, ..., x_n)$ *and* $DC(f_1) \cdot DC(f_2) > 0$. *Then there exists such functions that* $DC(F) < DC(f_1) + DC(f_2)$.

Proof. Let $f_1 = f$ from above and let f_2 be defined as follows

$$M_{f_2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

It is easy to see that in multiparty model $DC(f_2) = 4$ (2 bits sent and 2 bits received).

Since the last two rows (and the last two columns) of the matrix M_f are same we need to distinguish only between 3 different inputs. Therefore we can encode them as 0, 10, 11 respectively :

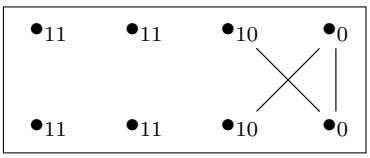


Figure 3

Instead of sending the whole input x the processor can send encoded input $c(x)$ and it will still suffice to compute the function f. Closer examination of the function f reveals, that if $f(x, y) = 1$ then $2 \leq |c(x)| + |c(y)| \leq 3$.

We can use this fact and construct the protocol for the function $F(x_1, x_2, x_3, x_4)$ in the following way:

1. Compute the function f first - send the signal vector $(1, 1, \epsilon, \epsilon)$.
2. Receive the message vector $(c(x_1), c(x_2), \epsilon, \epsilon)$ from the processors.
3. If $f(x_1, x_2) = 1$ then send the message vector $(\epsilon, \epsilon, 1, 1)$ and receive the reply vector $(\epsilon, \epsilon, c(x_3), c(x_4))$.
4. Evaluate $F(x_1, x_2, x_3, x_4)$.

Now lets look at the number of bits transmitted in each step.

1. 2 bits

2. Depending on the input pair x_1, x_2 - 2, 3 or 4 bits.

3. If $f(x_1, x_2) = 1$ then, as we have mentioned before, $2 \leq |c(x_1)| + |c(x_2)| \leq 3$ and in this case we send and receive $2 + 2 = 4$ more bits. If $f(x_1, x_2) = 0$ then no other communication is needed.

4. Altogether we need to communicate either $2+\{2,3\}+4 \leq 9$ or $2+4 = 6$ bits. In either case it is less than 10.

□

Here we get a different result compared to nondeterministic protocols. Being able to guess is a powerful help but sometimes that much power can be restrictive too.

Let $f' = 1 - f$. It follows that $DC(f') = 6$ and that if $f'(x, y) = 0$ then $2 \leq |c(x)| + |c(y)| \leq 3$. Thus

Corollary 2.2.7. *Let $F(x_1, ..., x_k, ..., x_n) = f_1(x_1, ..., x_k) \vee f_2(x_{k+1}, ..., x_n)$ and $DC(f_1) \cdot DC(f_2) > 0$. Then there exists such functions that $DC(F) < DC(f_1) + DC(f_2)$.*

As we see we can, in principle, save some communication bits if we use one coordinator instead of two. On the other hand, the amount of such bits is much smaller than as it was with nondeterministic protocols.

Note that if we had considered the original two way model we would not have had this result. The following protocol for the two way multiparty model shows that we need only 5 bits in order to compute the function f with inputs encoded according to the Figure 3.

1. Coordinator sends "1" to the left processor.

2. Left processor sends its input to the coordinator.

3. If the received input is

   - 1-bit long the coordinator sends "0" to the right processor.
   - 2-bit long the coordinator sends "1" to the right processor.

4. The right processor sends back either its input if it received "0" or only the first bit if it received "1".

5. The coordinator can now compute the function.

We see that only 5 bits were transmitted. The correctness of this protocol follows from the Figure 3.

We were not able to prove theorem 2.2.6 for the two way multiparty model.

Theorem 2.2.8. *[Ď04] Let*

$$f(x_1, ..., x_k, ..., x_n) = f_1(x_1, ..., x_k) \oplus f_2(x_{k+1}, ..., x_n).$$

Then $DC(f) = DC(f_1) + DC(f_2)$.

And since $DC(f) = DC(1 - f)$ and $(x \Leftrightarrow y) \equiv \neg(x \oplus y)$ it follows that

Corollary 2.2.9. *Let*

$$f(x_1, ..., x_k, ..., x_n) = f_1(x_1, ..., x_k) \Leftrightarrow f_2(x_{k+1}, ..., x_n).$$

Then $DC(f) = DC(f_1) + DC(f_2)$.

We see that in these two cases we cannot save one bit of communication complexity - again something different compared to nondeterministic $\oplus$ and $\Leftrightarrow$.

Theorem 2.2.10. *Let* $F(x_1, ..., x_k, ..., x_n) = f_1(x_1, ..., x_k) \Rightarrow f_2(x_{k+1}, ..., x_n)$ *and* $DC(f_1) \cdot DC(f_2) > 0$. *Then there exists such functions that* $DC(f) < DC(f_1) + DC(f_2)$.

Proof. Assume that for all functions f_1, f_2 it holds that $DC(f) = DC(f_1) + DC(f_2)$. Let f, g be the functions from the Theorem 2.2.6.

According to our assumption $DC(f \Rightarrow g) = DC(f) + DC(g)$. But

$$f \Rightarrow g \equiv (1 - f) \vee g$$

and according to the Theorem 2.2.7 (and remarks above this theorem)

$$DC((1 - f) \vee g) < DC(1 - f) + DC(g) = DC(f) + DC(g).$$

A contradiction. □

2.3 Very Hard Functions

In this section we will sum up the implications of previous two sections to hard and very hard functions. That is, if f and g are two very hard functions we would like to know if $f \circ g$ is also very hard.

Theorem 2.3.1. *Considering nondeterministic protocols, if $f(x_1, ..., x_k)$ and $g(x_{k+1}, ..., x_n)$ are two very hard functions then $f \circ g$ is also very hard where $\circ \in \{\oplus, \Leftrightarrow\}$.*

If $f(x_1, ..., x_k)$ and $g(x_{k+1}, ..., x_n)$ are two hard functions then $f \circ g$ is also hard where $\circ \in \{\wedge, \oplus, \Leftrightarrow\}$.

Theorem 2.3.2. *Considering deterministic protocols, if $f(x_1, ..., x_k)$ and $g(x_{k+1}, ..., x_n)$ are two very hard functions then $f \circ g$ is also very hard where $\circ \in \{\oplus, \Leftrightarrow\}$.*

We can also define asymptotic very hard functions as functions such that $C(f) = C(1 - f) = \Theta(mn)$ where n is the number of processors and m is the length of the input of each processor. From the practical point of view there is little difference between the very hard and asymptotically very hard functions, therefore we might be interested in ways of constructing them. There is one such easy way.

Theorem 2.3.3. *Let $f(x_1, ..., x_k)$ and $g(x_{k+1}, ..., x_n)$ be two hard functions not necessarily very hard and let $k = \Theta(n)$. Then $f \circ (1-g)$ is asymptotically very hard where $\circ \in \{\wedge, \vee, \oplus, \Leftrightarrow\}$.*

Proof. The proof follows from the Theorems 2.3.1 and 2.3.2. □

Chapter 3

Modified Multiparty Model

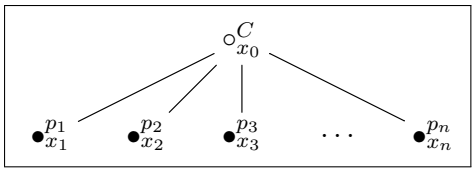
'Be what you would seem to be' - or, if you'd like it put more simply - 'Never imagine yourself not to be otherwise than what it might appear to others that what you were or might have been was not otherwise than what you had been would have appeared to them to be otherwise.'

Lewis Carroll
Alice's Adventures in Wonderland

In this chapter we introduce a modifications of the multiparty model we considered so far. In this modification we allow the coordinator to carry part of the input - in other words we merge one processor with the coordinator. Call this model $\aleph$.

We will show that this modification changes the original model quite substantially in certain ways.

The following graph shows the schema of this model:



We do not give the full formal definition of this modified model. It is basically the same as for the original multiparty model except a few changes given below:

The coordinator wishes to evaluate the function $f(x_0, x_1, ..., x_n)$ where x_0 is known only to the coordinator and x_i is known only to the processor p_i and $x_i \in \{0,1\}^m$ for all i.

A deterministic protocol P is an $(n+1)$-tuple of functions $(\phi_0, \phi_1, ..., \phi_n)$, for which the following holds: Let

$$K = \{0,1\}^m \times \{0,1,\$\}^* \times ... \times \{0,1,\$\}^* \quad (\{0,1,\$\}^* \text{ n times})$$

$$M = \{0,1\}^* \times ... \times \{0,1\}^* \quad (\text{n times})$$

(a) ϕ_0 is a function from K to $M \cup \{$"accept", "reject"$\}$. Intuitively, behaviour of the coordinator is given by ϕ_0, where the argument of ϕ_0 is a coordinator's input and a communication vector of all previous messages, with \$ serving as a delimiter between messages. The result of ϕ_0 are either the next messages sent to the parties or the coordinator stops the communication and accepts/rejects the input.

(b) For $i = 1, 2, ..., n,\ \phi_i$ is a function from $\{0,1\}^m \times \{0,1,\$\}^*$ to $\{0,1\}^+$. Intuitively, behaviour of the party $i, (1 \geq i \geq n)$, is given by ϕ_i, where the first argument of ϕ_i is the local input for the party i and the second argument of ϕ_i is a sequence of all previous messages on the link i (delimited by \$). The result of ϕ_i is the next message sent by the party i to the coordinator.

A computation under P on input $x = (x_1, ..., x_n)$ with $x_i \in \{0,1\}^m$ for each i is a communication vector $s^k = (s_1^k, ..., s_n^k)$, where $k \geq 0$, (k is the number of all phases performed on x under P), such that for every $j = 0, 1, 2, ..., k-1$ there is a communication vector $s^j = (s_1^j, ..., s_n^j)$, ($s^j$ is the communication vector after completing the j-th phase of the computation on x under P), for which (c), (d), (e) hold.

(c) $s_i^0 = \epsilon$ for $i = 1, 2, ..., n$; (the coordinator starts the communication with the empty communication vector $s^0 = (\epsilon, ..., \epsilon)$).

(d) For $j = 0, 1, 2, ..., k-1$ it holds: Let $\phi_0(x_0, s^j) = (d_1^j, ..., d_n^j)$. Then $s_i^{j+1} = s_i^j \$ d_i^j \$ \phi_i(x_i, s_i^j \$ d_i^j)$ if $d_i^j \neq \epsilon$, otherwise $s_i^{j+1} = s_i^j$ for $i = 1, 2, ..., n$.

(e) $\phi_0(x_0, s^k) \in \{$"accept", "reject"$\}$. If $\phi_0(x_0, s^k) =$"accept" $[\phi_0(x_0, s^k) =$"reject"$]$ then s^k is an accepting [rejecting] computation vector under P, (or, s^k is an accepting [rejecting] computation under P on x)

We can also consider nondeterministic protocols. In such a case, ϕ_i's are "nondeterministic functions", i.e., they may have several values (and therefore they are not functions). Moreover, they may be "partial nondeterministic functions", i.e., they may not be defined for all possible values of arguments; in such a case, the current communication is aborted.

Note that if $n = 1$ and we consider deterministic versions then this model behaves almost exactly like Yao's two party model - the only difference being that we still have a coordinator that has to start the computation. That is, the computation starts a fixed party and this might be sometimes restrictive. On the other hand, we need only one starting bit sent by the coordinator to fix this - after the only processor receives this bit the computation can go as in Yao's two party model.

We have seen that in Yao's model the trivial upper bound was $m + 1$ where m was the length of input - the $+1$ was there because we required both parties to know the output. And the same trivial upper bound holds for this new model too. Although we might need one bit start to computation, we require only coordinator to know the output.

In general, if we have a function f and some protocol P that computes this function in Yao's model with complexity $DC(P)$ then basically the same protocol $P_{\not\leftarrow}$ can compute the function f in this new model with complexity $DC(P) - 1 \leq DC(P_{\not\leftarrow}) \leq DC(P) + 1$:

- $P_{\not\leftarrow}$ cannot save more than one bit compared to P because we can use $P_{\not\leftarrow}$ almost as it is in Yao's model - the only difference is that $P_{\not\leftarrow}$ does not send the result to Bob - this is the only bit we can save. Therefore $DC(P) - 1 \leq DC(P_{\not\leftarrow})$.
- Let Alice be the coordinating party in the new model.
- If Alice always sends the first message and is always first to know the output then the only difference between P and $P_{\not\leftarrow}$ is that $P_{\not\leftarrow}$ does not send the result to Bob. Thus $DC(P_{\not\leftarrow}) = DC(P) - 1$.

- If Alice always sends the first message but sometimes Bob learns the output before Alice then the protocols are same and $DC(P_{\aleph}) = DC(P)$.

- If both Alice and Bob may start the communication and Alice is always first to know the output then $P_{\aleph}$ may need a "starting" bit but it does not need to send the output to Bob. Hence $DC(P_{\aleph}) \leq DC(P)$.

- If both Alice and Bob may start the communication and sometimes Bob learns the output before Alice then $DC(P_{\aleph}) \leq DC(P) + 1$.

Similarly we can derive a protocol P for Yao's model from $P_{\aleph}$ with complexity $DC(P_{\aleph}) - 1 \leq DC(P) \leq DC(P_{\aleph}) + 1$.

Something very similar holds for nondeterministic versions too. The only difference concerns the new model where we can actually make some use of the starting bit - the bit "responsible" for the possible one bit difference between $DC(P)$ and $DC(P_{\aleph})$.

In the trivial protocol the coordinator can guess the first bit of the processor's input and send this bit as a starting bit. If the guess was successful then the processor upon receiving this bit reponds by sending the rest of his input to the coordinator. This way we have only transmitted m bits where m is the length of the input - that is, we saved one bit compared to the deterministic version of this model.

In general, Alice can guess that Bob would start the communication in Yao's model and also what would be the first message sent by him. Then Alice will choose the first bit of this guessed message as a starting bit and send it to Bob. Bob knows that he would start the communication in Yao's model and upon receiving the starting bit and if the Alice's guess was successful he responds with the rest of the "first" message (one that Alice guessed Bob would sent had they been runnig Yao's model). After this the communication continues as in Yao's model. Again, if Alice is first to know the answer then she does not have to reveal it to Bob. But if Bob is the first to know, then he must tell Alice the result - this is same as in deterministic version mentioned above.

Thus $P_{\aleph}$ can be better by one bit compared to P - no need to send result to Bob. On the other hand P cannot be better than $P_{\aleph}$ - protocol P can be used in the new model with slight the modification of protocol's start

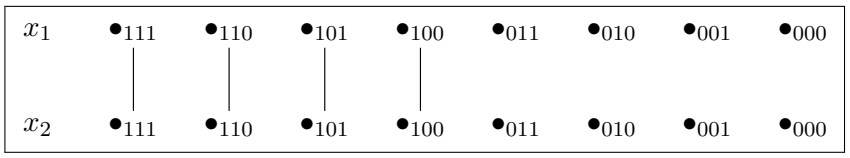
described above that does not change the communication complexity.

Hence $C(P_{\aleph}) \leq C(P) \leq C(P_{\aleph}) + 1$ and $C(P) - 1 \leq C(P_{\aleph}) \leq C(P)$.

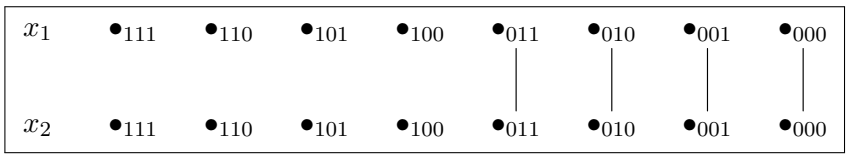
Because of this analogy, to avoid ambiguity we will use $C^2(f)$ $\left(DC^2(f)\right)$ for the two party communication complexity and $C^{\aleph}(f)$ $\left(DC^{\aleph}(f)\right)$ for this model's communication complexity.

Let $f(x_1, x_2), g(x_1, x_2) : \{000, 001, ..., 111\} \times \{000, 001, ..., 111\} \to \{0, 1\}$ be two Boolean functions given as follows:

f:



g:



What is the communication complexity of these functions in the two party model? If we look at the matrices M_f and M_g and use the Rank method with remark 1.2.10

$$M_f = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad M_g = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

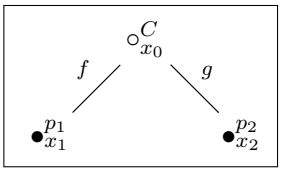
we get $rank(M_f) = rank(M_g) = 4$ and

$$DC^2(f) = C^2(f) \geq \lg(2rank(M_f) - 1) = \lg 7$$
$$DC^2(g) = C^2(g) \geq \lg(2rank(M_g) - 1) = \lg 7$$

Both functions f, g are hard and the trivial protocol is as good as any. Therefore

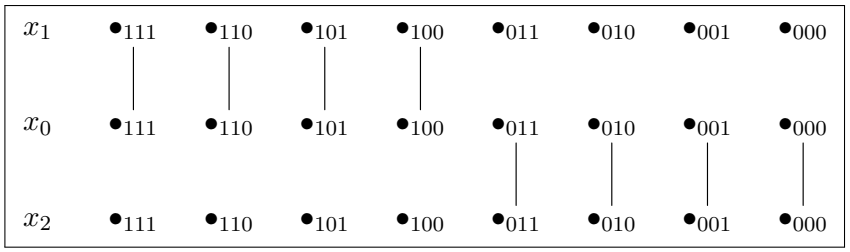
$$DC^2(f) = DC^{\aleph}(f)$$
$$DC^2(g) = DC^{\aleph}(g).$$

Now we want to compute $f(x_0, x_1) \oplus g(x_0, x_2)$ in our new model.



If we computed the functions separately we would need at least 6 bits from the processors plus 2 bits sent by the coordinator.

But since the first argument is same for both functions in model $\aleph$ the following protocol for $f(x_0, x_1) \oplus g(x_0, x_2)$ needs only 3+1 bits (3 bits sent by a processor and 1 bit sent by the coordinator).



1. If $x_0 \in \{111, ..., 100\}$ then the coordinator sends one bit to p_1 and p_1 sends back its input. The coordinator does not need to know x_2 because in this case $g(x_0, x_2) = 0$ for any x_2.

2. If $x_0 \in \{011, ..., 000\}$ then the coordinator sends one bit to p_2 and p_2 sends back its input. The coordinator does not need to know x_1 because in this case $g(x_0, x_1) = 0$ for any x_1.

3. Now the coordinator can compute $f(x_0, x_1) \oplus g(x_0, x_2)$.

Thus we can save 3+1 bits in this model (compared to separate computation). Moreover, the functions, now defined on the set $\{0,1\}^3 \times \{0,1\}^3$, can be easily extended to a set $\{0,1\}^n \times \{0,1\}^n$ for some arbitrary n. In any case

$$DC^{\aleph}(f \oplus g) = \frac{1}{2}\left(DC^{\aleph}(f) + DC^{\aleph}(g)\right)$$

By analogy we get the same result for $f \Leftrightarrow g$:

$$DC^{\aleph}(f \Leftrightarrow g) = \frac{1}{2}\left(DC^{\aleph}(f) + DC^{\aleph}(g)\right)$$

Even more interesting is $f(x_0, x_1) \wedge g(x_0, x_2)$ (same model). One can easily see that for any given x_0 either $f(x_0, x_1) = 0$ for all x_1 or $g(x_0, x_2) = 0$ for all x_2. Therefore

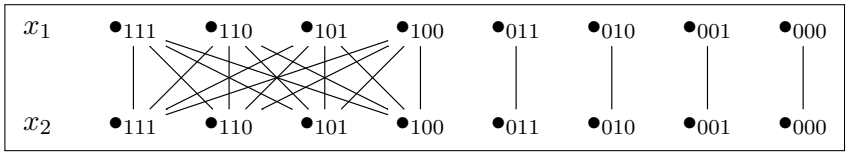
$$C^{\aleph}(f \wedge g) = 0$$

Compare this results with the original multiparty model:

$$DC(f \oplus g) = DC(f) + DC(g)$$

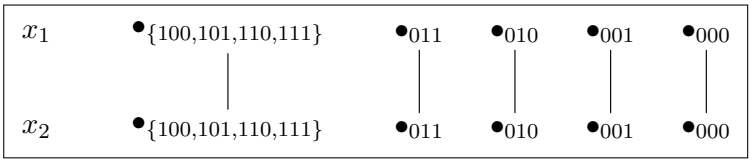
$$C(f \wedge g) = C(f) + C(g)$$

Now consider the function $h : \{0,1\}^3 \times \{0,1\}^3 \to \{0,1\}$ defined as follows h:



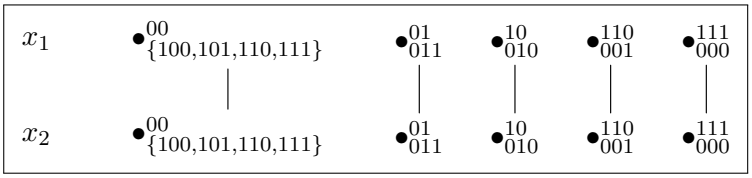
$$M_h = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

It is easy to see that $C^2(h) = 3 + 1$. Thus the best protocol is as good as the trivial one and from this it follows that $C^{\aleph}(h) = 1 + 3$. But since we cannot distinguish, for example, $h(111, x_2)$ from $h(110, x_2)$ (where x_2 is arbitrary) we can encode the inputs in more efficient way - we will consider $\{100, 101, 110, 111\}$ to be one element. Therefore we can "change" the input of function h where the element corresponding to the last row/column is now the set $\{100, 101, 110, 111\}$.
h':

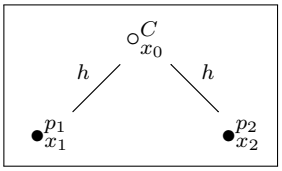


$$M_{h'} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

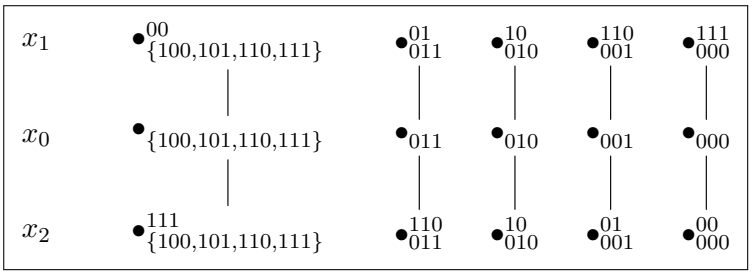
The function h' is still hard in model $\aleph$ - $C^{\aleph}(h') = 1 + 3$ - but now there are inputs for which we do not need to transmit 3 bits in order to compute h'. In the following graph the subscript corresponds to the original input and the superscript to the message sent instead of the original input.



Now we want to evaluate the function $h(x_0, x_1) \wedge h(x_0, x_2)$ in the new model.



According to our remarks above we can use the following schema:



Notice that x_0 do not have superscript - this part is known to the coordinator and therefore does not needs to be sent. Also notice that we used different encoding for x_1 and x_2. This allows us to design a protocol that needs to transmit only 7 bits to compute the function $h(x_0, x_1) \wedge h(x_0, x_2)$:

1. If $x_0 \in \{000, 001, 010\}$ send 1 to p_1 and 0 to p_2, otherwise send 0 to p_1 and 1 to p_2.

2. Each processor, upon receiving 1, sends back the whole encoded input and, upon receiving 0, sends back only the first two bits of the encoded input (superscript in the above schema).

3. The coordinator can now compute the function.

The protocol needs to transmit only 2+3+2=7 or 2+2+3=7 bits and its correctness is obvious from the above schema. On the other hand, if we computed the functions $h(x_0, x_1)$ and $h(x_0, x_2)$ separately we would need to transmit 2(1+3)=8 bits.

Moreover, the functions, now defined on the set $\{0,1\}^3 \times \{0,1\}^3$, can be easily extended to a set $\{0,1\}^n \times \{0,1\}^n$ for some arbitrary n.

In theorem 2.1.6 the function $f(x_1, ..., x_n) = 1$ iff $x_i = x_j$ for all (i, j) (where $x_i \in \{0,1\}^m$) was considered. It was shown that $nm \leq C(f)$. One can easily see that the same holds for the function $f'(x_0, x_1, ...x_n)$ computed in the new model - $nm \leq C^{\aleph}(f')$. If some protocol P' computed the function f' with less communication complexity than nm then we could simulate such protocol in the original model - the coordinator would simply guess the input x_0 and then run the protocol P'.

Now we can put the above results together:

Theorem 3.0.4 (Last theorem)**.** *There exist such functions f, g, h: $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ that $C^{\aleph}(f) = C^{\aleph}(g) = C^{\aleph}(h) = 1 + n$ and for which the following holds:*

$$C^{\aleph}(f \oplus g) = n + 1$$
$$C^{\aleph}(f \wedge g) = 0$$
$$C^{\aleph}(h \wedge h) < 2(n + 1).$$

Let F be a function defined as follows

$$F(x_0, x_1, ..., x_n) = 1 \quad \textit{iff} \quad x_0 = x_1 = ... = x_n \quad \textit{where} \quad x_i \in \{0,1\}^m.$$

Then

$$C^{\aleph}(F) \geq nm.$$

Model $\aleph$ allows us to save hell of a lot of communication bits in some cases. Hence even this slight modification creates, in principle, very different model.

Why is this model interesting? Well, we can look on it as a generalized version of the original multiparty model. All functions that we might consider for the original model can be simulated on this model - we simply add one more argument to these functions - the one that coordinator holds - and we will completely disregard it. On the other hand, the converse is not true - the original model cannot simulate the new one.

Moreover, in some cases we can regard this model as a model that is closer to real-life application, e.g. in networks. Recall the example from the very beginning about the backup server - such server holds a string which we want to update if necessary. Such string is also a variable.

Chapter 4

Epilogue

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.
"I don't much care where——" said Alice.
"Then it doesn't matter which way you go," said the Cat.
"——so long as I get somewhere," Alice added as an explanation.
"Oh, you're sure to do that," said the Cat, "if you only walk long enough."

Lewis Carroll
Alice's Adventures in Wonderland

What remains to be done, among other things, is to return to the original two way deterministic multiparty model and to determine whether we can save some communication bits when we link two functions together with $\wedge$. We were not able to resolve this and we only conjecture the even if some save up was possible (for some functions) it would not be more than some small constant (constant with respect to the length of inputs).

The newly proposed model is also worth more research. It has close ties with Yao's two party model and with the original multiparty model. It is, for example, interesting to look for some lower bound methods that would be some generalized versions of lower bound techniques from the previous models.

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[BNS89] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols and logspace hard pseudorandom sequences. *Proceedings of the 21th ACM STOC*, pages 1–11, 1989.

[CFL83] A. K. Chandra, M. L. Furst, and R. J. Lipton. Multi-party protocols. *Proceedings of the 15th ACM STOC*, pages 94–99, 1983.

[DF89] D. Dolev and T. Feder. Multiparty communication complexity. *Proceedings of the 30th IEEE FOCS*, pages 428–433, 1989.

[DF92] D. Dolev and T. Feder. Determinism vs. nondeterminism in multiparty communication complexity. *SIAM Journal on Computing 21*, pages 889–895, 1992.

[DHS96] M. Dietzfelbinger, J. Hromkovič, and G. Schnitger. A comparison of two lower-bound methods for communication complexity. *Theoretical Computer Science*, pages 39–51, 1996.

[DKW09] J. Draisma, E. Kushilevitz, and E. Weinreb. Partition arguments in multiparty communication complexity. *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, pages 390–402, 2009.

[ĎR98] P. Ďuriš and J. D. P. Rolim. Lower bounds on the multiparty communication complexity. *Information and Computation 56*, pages 90–95, 1998.

[GKP94] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-Wesley, 2nd edition, 1994.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[Kom67] J. Komlós. On the determinant of (0,1)-matrices. *Studia Scientiarum Mathematicarum Hungarica 2*, pages 7–21, 1967.

[Kom68] J. Komlós. On the determinant of random matrices. *Studia Scientiarum Mathematicarum Hungarica 3*, pages 387–399, 1968.

[Kus97] Eyal Kushilevitz. Communication complexity. 1997.

[LS81] R. J. Lipton and R. Sedgewick. Lower bounds for vlsi. *Proceedings of the 13th ACM STOC*, pages 300–307, 1981.

[Man97] J. Manuch. Nedeterministická komunikačná zložitosť na modeli typu hviezda. *Diplomová práca*, 1997.

[Mil94] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machines. *Proceedings of the 26th ACM STOC*, pages 625–634, 1994.

[MS82] K. Mehlhorn and E. Schmidt. Las-vegas is better than determinism in vlsi and distributed computing. *Proceedings of the 14th ACM STOC*, pages 330–337, 1982.

[PS82] C. H. Papadimitriou and M Sipser. Communication complexity. *Proceedings of the 14th ACM STOC*, pages 196–200, 1982.

[Tho79] C. D. Thompson. Area-time complexity for vlsi. *Proceedings of the 11th ACM STOC*, pages 81–88, 1979.

[Ď04] P. Ďuriš. Multiparty communication complexity and very hard functions. *Information and Computation 192*, pages 1–14, 2004.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing. *Proceedings of the 11th ACM STOC*, pages 209–213, 1979.