Towards Time-Space Lower Bounds On Branching Programs

Jeff Edmonds* Russell Impagliazzo†

September 6, 2016

History:

- This work was done in 1993 while Jeff Edmonds was doing a post-doctorate at International Computer Science Institute Berkeley and Russell Impagliazzo was newly at Dept. of Computer Science UCSD.

  It was recompiled in Latex in 2016 with a cover page added. The originals were cited by Pudlak and Sgall in Nov 1993. All but this cover page of the current document is a verbatim copy of this 1993 original. See http://www.cse.yorku.ca/∼jeff/research/spi

- A lower bound on the time space trade off for directed st-conn on a branching program is reduced to proving a lower bound on a simple "card" game. See http://www.cse.yorku.ca/∼jeff/research/spi/russell.comb.pdf or main.pdf Section 1.

- A lower bound on the time space trade off for some problem (not st-conn) on a branching program is reduced to proving a lower bound on a simpler card game. Some attempts at this lower bound are made. See http://www.cse.yorku.ca/∼jeff/research/spi/spi.comb.pdf or main.pdf Section 2.

- Pavel Pudlak and Jiri Sgall came up with an $k^{2/5}$ upper bound, beating the bound that we hoped to prove as a lower bound. This killed all our intuition about the problem.

- A more general card game was defined. It was conjectured that this game was actually hard. See http://www.cse.yorku.ca/∼jeff/research/spi/main.pdf Section 3.

- Pavel Pudlak and Jiri Sgall came up with a $\log^2 k$ upper bound to both the original and the more general card game. This kills any hope us using this game for lower bounds on branching programs.

- We defined an even more general card game. It is still conjectured that this game is actually hard. This game is called the Lazy Susan problem. The game is similar to the previous game, except instead of k cards, you have k lazy susans. Each lazy susan has r "sides" on which you can read bits. (Note in the card game, $r = 2$). At each round you can look at only one side on each lazy susan and then write on the black board. Nothing has been written about it.

*Now at York University Toronto jeff@cse.yorku.ca
†Dept. of Computer Science UCSD, russell@cs.ucsd.edu

Towards Time-Space Lower Bounds On Branching Programs

Jeff EdmondsWork done while at International Computer Science Institute Berkeley. Now at York University Toronto jeff@cse.yorku.ca Russell ImpagliazzoDept. of Computer Science UCSD, russell@cs.ucsd.edu Oct 1993

Abstract: Section 1: The goal is to prove a time-space tradeoff for *st*-conn on the branching program model. My parity comb graph seemed to be a good graph to consider. Russell Impagliazzo suggested a problem that seemed to characterize what was still needed to prove the result. Two steps remained. A lower bound was needed for Russell's problem and a reduction was needed from this problem to *st*-conn. The first proved to be hard. The second proved to be easy. Below I will define Russell's problem, give an easier version of it, give a more complex version of it, and give a sketch of the proof that if the conjecture about the more complex version is true, then a time-space tradeoff for *st*-conn on the branching program model follows. It interesting that if the conjecture is true than a lower bound on the time-space tradeoff for element distinctness follows as well. I have not written this up. The last section is a surprising upper bound contradicting the conjectured lower bound for a related communication game.

Section 2: I define the $n = 2$ version of this communication game. Russell pointed out that the $n = 2$ game is similar to element distinctness. In fact, a lower bound for (a slightly more complex version of) the $n = 2$ game gives a lower bound on the time-space tradeoff for element distinctness. Below are my thoughts on this game. Included is a surprising upper bound.

Section 3: This sketch of a paper defines a more general card game and proves that a non-trivial lower bound for this game with cheating gives non-trivial lower bound for the time-space trade off (for some boolean function) on a branching program.

1 *st*-Connectivity

1.1 Russell's Problem

Russell's problem is defined as follows. The input consists of k vectors of l bits each. The output is $\bigwedge_{i\in[1..k]} \bigoplus_{j\in[0..l-1]} \alpha_{\langle i,j\rangle}$, i.e. whether one of the vectors has odd parity. The computation model is a branching program that is broken into blocks of time. The blocks of time can be arbitrarily long. The goal is to prove a tradeoff between the space and the number of blocks. There is however a restriction. During each block of time the computation can access at most ϵl of the bits from each of the vectors. However, for each vector and each time block, the computation is able to non-uniformly choose which ϵl bits to read.

The conjecture is that with S space, this computation requires at least $\frac{k}{S}$ (actually $\frac{k}{\epsilon S}$) time blocks. The intuition is as follows. In order to know that all k vectors have zero parity, the parity of each vector must be computed at some point in the computation. When knowing only ϵl bits of a vector, the parity of the vector is not known. What is needed is the parity of the ϵl bits of this vector that were read during the previous block of time. This requires the use of one bit of memory between the blocks of time. There are only S bits of memory. Hence $\frac{k}{S}$ blocks are needed to compute the parities of all k vectors.

Such a lower bound would say nothing about the time to compute $\bigwedge_{i\in[1..k]} \bigoplus_{j\in[0..l-1]} \alpha_{\langle i,j\rangle}$. In fact, it is trivially in linear time, constant space. Such a bound would, however, indicate an order in which the bits must be accessed.

CK Poon, Hisao Tamaki, and I were working (without luck) on a communication game that relates to the problem when $l = 2$ and $S = 1$. The input to the game consists of k pairs of bits (or two k bit vectors). The question is whether or not the two vectors are the same (for each pair of two bits the parity is 0, hence for each pair of bits, the bits are the same.) Each time step a new player is able to read either the left or the right bit of each pair (but never both). The player then communicates one bit of information to the world. In the end, the answer must be determined by the bits communicated. Note that if we restrict the game further so that the players must either read the left bit of each pair or the right bit of each pair, then the game becomes the Karchmer Wigderson universal communication game.

I think that this is a great game, because it characterizes a feature of information that intuitively should be true. The t^{th} player only knows half the bits and t communicated bits. How can he compare his half to the other half unless $t = k$?

A lower bound of k for this communication game gives a lower bound for Russell's game for $n = 2$ and any space S. Given an algorithm to Russell's game for $n = 2$ and space S. The player communicates what

is written in the S bits of memory after each time block. This depends only on what was in memory at the beginning of the time block (which is included in the communication) and on the bits read during the time block. The communication game requires k bits to be communicated, so Russell's game requires $\frac{k}{S}$ time blocks.

In Section 2.3, an algorithm is proven to exist for which $O\left(\sqrt{k \log k}\right)$ bits are communicated. This does not give an algorithm for Russell's game. Because in Russell's game what is done next can only depend on the S bits saved, not on the entire previous communication.

The more complex game is the same except for the following. The computation is **legally** able to access all but 1 bit of each vector during a single block of time. Some times the computation may read all the bits of a vector within one block, but this is considered **cheating**. The amount of cheating is restricted in the following way. Uniformally at random choose an input conditional on the fact that the parity of each vector is 0, i.e. $\bigwedge_{i \in [1..k]} \bigoplus_{j \in [0..l-1]} \alpha_{\langle i,j \rangle} = 0$. The branching program must be such that the probability is less than 2^{-S} of the computation cheating on this input for more than $\frac{k}{2}$ vectors.

Conjecture 1 *The number of time blocks required for the more complex version of Russell's problem is* $\Omega\left(\frac{k}{S}\right)$.

1.2 *st*-connectivity

Theorem 1 *If Conjecture 1 is true then st-connectivity on a branching program requires* $T \times S^{\frac{1}{3}} \in \Omega\left(m^{\frac{2}{3}} n^{\frac{2}{3}}\right)$.

Proof of Theorem 1: The proof is very similar to the NNJAG result in my thesis. Hence, the following only sketches the differences between the proofs.

1.3 Parity Comb Graphs

The family of graphs considered are the **parity comb graphs**. As before, a parity comb graph is composed of a back, $v_1, \ldots, v_n$, and χ teeth of length $l = \frac{n}{\chi}$. Again, the variables $y_1, \ldots, y_m \in [1..\chi]$ will be used to specify which tooth each of the m connecting edges leads to. The new feature of the graph is that each tooth, for $r \in [1..\chi]$, is formed from two columns of nodes $u_{\langle r,1,0 \rangle}, \ldots, u_{\langle r,l,0 \rangle}$ and $u_{\langle r,1,1 \rangle}, \ldots, u_{\langle r,l,1 \rangle}$. The variables $\alpha_{\langle r,0 \rangle}, \ldots, \alpha_{\langle r,l-1 \rangle} \in \{0,1\}$ indicate how these nodes are connected. If $\alpha_{\langle r,0 \rangle} = 0$, then each back node v_j for which $y_j = r$, is connected to the left node on the top of the r^{th} tooth by the edge $\left\langle v_j, u_{\langle r,1,0 \rangle} \right\rangle$, while if $\alpha_{\langle r,0 \rangle} = 1$, they are connected to the right top node $u_{\langle r,1,1 \rangle}$. For each $k \in [1..l-1]$, if $\alpha_{\langle r,k \rangle} = 0$, then two directed edges go directly down, $\left\langle u_{\langle r,k,0 \rangle}, u_{\langle r,k+1,0 \rangle} \right\rangle$ and $\left\langle u_{\langle r,k,1 \rangle}, u_{\langle r,k+1,1 \rangle} \right\rangle$, while if $\alpha_{\langle r,k \rangle} = 1$, the two directed edges cross over $\left\langle u_{\langle r,k,0 \rangle}, u_{\langle r,k+1,1 \rangle} \right\rangle$ and $\left\langle u_{\langle r,k,1 \rangle}, u_{\langle r,k+1,0 \rangle} \right\rangle$. Finally, the right node on the bottom of the r^{th} tooth is connected to t with the edge $\left\langle u_{\langle r,l,1 \rangle}, t \right\rangle$. It follows that s **is connected to** t **via the** r^{th} **tooth** iff $\bigoplus_{k \in [0..l-1]} \alpha_{\langle r,k \rangle} = 1$. This means that $st\text{-conn} = \bigwedge_{r \in [1..\chi]} \bigoplus_{k \in [0..l-1]} \alpha_{\langle r,k \rangle}$. See the figure below.

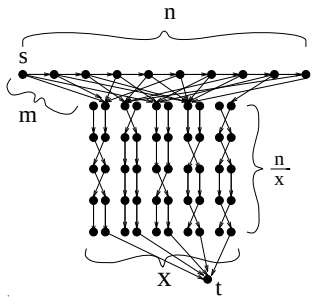

Figure 1: A parity comb graph

1.4 The Branching Program

Each node of a branching program queries an input variable, i.e. specifies one node of the input graph and one edge label and queries to determine which node is adjacent along this edge. By querying for the end point of the connecting edge e_j, $j \in [1..m]$, the branching program learns that it is connected to the node $u_{\langle r,1,a\rangle}$ for some $r \in [1..\chi]$ and $a \in [0,1]$. This implies that $y_j = r$ and $\alpha_{\langle r,0\rangle} = a$. When this occurs, we say that the computation **queries** $\alpha_{\langle r,0\rangle}$. Note that the branching program is not able to directly query $\alpha_{\langle r,0\rangle}$, but must search connecting edges e_j until one is found that is attached to the r^{th} tooth. The $\alpha_{\langle r,k\rangle}$ for $k \neq 0$ are easier to query. By querying either node $u_{\langle r,k,0\rangle}$ or node $u_{\langle r,k,1\rangle}$, for $r \in [1..\chi]$ and $k \in [1..l-1]$, the branching program learns the value of $\alpha_{\langle r,k\rangle}$.

1.5 The Probability Distribution $\mathcal{D}$ on Parity Comb Graphs

The probability distribution $\mathcal{D}$ is defined by constructing parity comb graphs as follows. Randomly partition the teeth into two equal size subsets $easyteeth_G$ and $hardteeth_G \subseteq [1..\chi]$. Randomly choose $\frac{\chi}{2}$ of the m connecting edges and put the associated variables y_j in the set $hardedges_G$. Randomly attach each of these "hard" connecting edges to one of the "hard" teeth in a one-to-one way. The set $easyedges_G$ is defined to contain the remaining y_j variables. Independently assign each $y_j \in easyedges_G$ a tooth from $easyteeth_G$ chosen uniformly at random.

The variables $\alpha_{\langle r,k\rangle}, r \in [1..\chi], k \in [0..l-1]$ are chosen independently at random, subject to the condition that $\bigoplus_{k\in[0..l-1]} \alpha_{\langle r,k\rangle} = 0$ for each tooth.

1.6 The Definition of Progress

As before, the branching program $\mathcal{P}$ is broken into sub-branching programs $\widehat{\mathcal{P}}$ of height h. We will say that the amount of **progress** made is the number of hard teeth, i.e. $r \in hardteeth_G$, for which the computation has accessed each of the variables $\alpha_{\langle r,0\rangle}, \ldots, \alpha_{\langle r,l-1\rangle}$ within the same block of time.

1.7 Progress within Sub-Branching Program

The key step is to prove that a shallow sub-branching program cannot make much progress for many inputs. Consider one of the sub-branching programs $\widehat{\mathcal{P}} \in \mathcal{P}$. We will determine how much progress it makes for every input $G \in \mathcal{D}$ (even if the computation on input G never reaches the root of $\widehat{\mathcal{P}}$).

For each input $G \in \mathcal{D}$, define $\mathcal{C}_G \subseteq [1..\chi]$ to be the set of teeth r for which the computation during the block of time specified by $\widehat{\mathcal{P}}$, on input G, accesses each of the variables $\alpha_{\langle r,0\rangle}, \ldots, \alpha_{\langle r,l-1\rangle}$. The teeth have length l and the computation by $\widehat{\mathcal{P}}$ performs only h steps; therefore $|\mathcal{C}_G| \leq \frac{h}{l}$. Let $c = \frac{h}{l}$ denote this bound. Clearly, $|\mathcal{C}_G \cap hardteeth_G|$ is the amount of progress made by the sub-branching program $\widehat{\mathcal{P}}$ on input G.

Lemma 1 *If* $h \leq \frac{|easyteeth_G|}{2}$, *then* $\Pr_{G\in\mathcal{D}}\Big[\ |\mathcal{C}_G \cap hardteeth_G| \geq 2\rho c\Big] \leq 2^{-0.38\rho c}$, *where* $\rho = \frac{\chi}{m}$.

Proof of Lemma 1: Note that the set of teeth $\mathcal{C}_G$ depends on the input G only as far as which computation path γ it follows. Therefore, it is well defined to instead refer to the set $\mathcal{C}_\gamma$. Because every input follows one and only one computation path γ through $\widehat{\mathcal{P}}$, it is sufficient to prove that for every path γ, a lot of progress is made for very few of the inputs that follow the computation path γ. Specifically, for each path γ through $\widehat{\mathcal{P}}$, we prove that $\Pr_{G\in\mathcal{D}}\Big[\ |\mathcal{C}_\gamma \cap hardteeth_G| \geq 2\rho c \ \Big|\ G \text{ follows } \gamma\Big] \leq 2^{-0.38\rho c}$.

Each tooth $r \in \mathcal{C}_\gamma$ can be thought of as a trial. The r^{th} trial consists of choosing a random input G subject to the condition that G follows the computation path γ. The trial succeeds if the r^{th} tooth is hard. For each trial $r \in \mathcal{C}_\gamma$ and each $\mathcal{O} \in \{succeeds, fails\}^{\mathcal{C}_\gamma - \{r\}}$, let $Z_{\langle\gamma,r,\mathcal{O}\rangle} = \Pr_{G\in\mathcal{D}}\Big[\ r \in hardteeth_G \ \Big|\ G \text{ follows } \gamma \text{ and } \text{ satisfies } \mathcal{O}\Big]$, where $\mathcal{O}$ indicates the condition that the other trials have the stated outcome. We will proceed to prove that for each r and each $\mathcal{O}$, $Z_{\langle\gamma,r,\mathcal{O}\rangle} \leq \frac{\chi}{m} = \rho$.

In order to bound $Z_{\langle\gamma,r,\mathcal{O}\rangle}$, fix $r \in \mathcal{C}_\gamma$ and the outcomes $\mathcal{O} \in \{0,1\}^{\mathcal{C}_\gamma - \{r\}}$ for the other trials. The first step is to understand the condition "G follows γ". A computation path γ can be specified by stating the collection of variables queried and their values. For example, $\gamma = \{y_j = r, \ldots, y_{j'} = r', \alpha_{\langle r'',k\rangle} = 0, \ldots, \alpha_{\langle r''',k'\rangle} = 0\}$.

Because $r \in \mathcal{C}_\gamma$, we know that each of the variables $\alpha_{\langle r,0\rangle}, \ldots, \alpha_{\langle r,l-1\rangle}$ are queried. Recall that the branching program is not able to directly query $\alpha_{\langle r,0\rangle}$, but must search the connecting edges e_j until one is found that is attached to the r^{th} tooth. Querying $\alpha_{\langle r,0\rangle}$ means that $y_j = r$ for some $j \in [1..m]$ was also queried. Hence, the condition that "G follows γ" includes the condition that $y_j = r$.

How does this condition by itself affect the probability that r is in $hardteeth_G$? From the definition of $hardedges_G$, we know that if $y_j = r$, then $y_j \in hardedges_G$ if and only if $r \in hardteeth_G$. This gives us that $\Pr_{G\in\mathcal{D}}\left[\, r \in hardteeth_G \;\middle|\; y_j = r \,\right] = \Pr_{G\in\mathcal{D}}\left[\, y_j \in hardedges_G \;\middle|\; y_j = r \,\right]$. This is equal to $\Pr_{G\in\mathcal{D}}\left[\, y_j \in hardedges_G \,\right]$, because we know that y_j has some value and there is a symmetry amongst all the possible values. Hence, telling you that $y_j = r$ gives you no information about whether $y_j \in hardedges_G$. Finally, $\Pr_{G\in\mathcal{D}}\left[\, y_j \in hardedges_G \,\right] = \frac{\chi/2}{m}$, because $\frac{\chi}{2}$ of the variables $y_1, \ldots, y_m$ are randomly chosen to be in $hardedges_G$.

What remains is to consider the additional effects of the other conditions. See my thesis for this. The only difference is the addition of the $\alpha_{\langle r,k\rangle}$ variables, but these are completely independent of which teeth are hard. ■

1.8 Reducing to Russell's Problem

The thesis explains how to use the bound on the amount of progress achieved in a subprogram to obtain a bound on the total progress made. The only difference here is that the constants need to be changed by a factor of 2 so that we bound the probability that the branching program makes progress for only half the hard teeth instead of all the hard teeth. The lemma obtained is the following.

Lemma 2 $\Pr_{G\in\mathcal{D}}\left[\, \textit{more than } \frac{\chi}{4} \textit{ progress is made for } G \,\right] \;\leq\; 2^{-S}$.

Fix one partition of the teeth into $hardteeth_G$ and $easyteeth_G$, fix one setting of the variables $y_1, ..., y_m$, and fix one setting of the variables $\alpha_{\langle r,0\rangle}, \ldots, \alpha_{\langle r,l-1\rangle}$ for each of the easy teeth $r \in easyteeth_G$ such that

$$\Pr_{G\in\mathcal{D}}\left[\, \text{more than } \tfrac{\chi}{4} \text{ progress is made for } G \,\middle|\, \text{the fixed stuff} \,\right] \;\leq\; 2^{-S}.$$

Consider the branching program restricted by the fixed information. The only parts of the input remaining to be chosen are the values of $\alpha_{\langle r,0\rangle}, \ldots, \alpha_{\langle r,l-1\rangle}$ for the $\frac{\chi}{2}$ hard teeth. The output of the restricted branching program must compute $\bigwedge_{r\in hardteeth_G} \bigoplus_{k\in[0..l-1]} \alpha_{\langle r,k\rangle}$. This restricted branching program solves the more complex version of Russell's game with $k' = \frac{\chi}{2}$ vectors each of length l. Note that making progress in the st-conn setting is the same as cheating in Russell's setting. The input distribution $\mathcal{D}$ restricted to the fixed information is simply choosing Uniformally at random a input to Russell's game conditional on the fact that the parity of each vector is 0. It follows that the restricted branching program has the property that the probability of cheating on more than $\frac{k}{2}$ vectors is less than 2^{-S}. The space of this branching program is no more than S and the number of time blocks is no more than $\frac{T}{h}$. In order to contradict Conjecture 1, the number of blocks must be less than $\Omega\left(\frac{k}{S}\right)$. In fact, $\frac{T}{h\in\Omega\left(\frac{k}{S}\right)}$, when plugging in the setting of the parameters used in the thesis. Recall, $h = \frac{\chi}{4}$, $\chi = 2.77 m^{\frac{1}{3}} n^{\frac{1}{3}} S^{\frac{1}{3}}$, and $T_{max} = 0.09 \frac{m^{\frac{2}{3}} n^{\frac{2}{3}}}{S^{\frac{1}{3}}}$. In conclusion, if there is a branching program contradicting the time-space trade off for st-connectivity, then there is one contradicting the block-space trade off for Russell's problem. ■

1.9 Open Problems

It would be interesting to get a similar sort of result for CK Poon's NNJAG result. The graph to consider might be two copies of his graph laid on top of each other. For each edge $\langle u, v\rangle$ of his graph, there is a parity variable $\alpha_{\langle u,v\rangle}$. If $\alpha_{\langle u,v\rangle} = 0$, then the edge $\langle u, v\rangle$ appears in the top copy of the graph and the corresponding edge $\langle u', v'\rangle$ appears in the bottom copy. If $\alpha_{\langle u,v\rangle} = 1$, then cross over edges $\langle u, v'\rangle$ and $\langle u', v\rangle$ are given instead. A "pebble" on this parity graph would give both the location of the node in the original graph and the parity of the path from s to the pebble. In the above proof, I charge only one bit for each "pebble", considering only the parity bit. This can not be done for a $\Omega\left(\log^2 n\right)$ space bound.

2 Element Distinctness

The goal is to prove a time-space tradeoff on the branching program model. I have a lower bound for *st*-connectivity on the NNJAG model. Russell Impagliazzo suggested a problem that seemed to characterize what was still needed to prove the same result for branching programs. In fact, I can reduce *st*-connectivity to (a slightly more complex version of) Russell's game. Below I define the $n = 2$ version of this communication game. Russell pointed out that the $n = 2$ game is similar to element distinctness. In fact, a lower bound for (a slightly more complex version of) the $n = 2$ game gives a lower bound on the time-space tradeoff for element distinctness. Below are my thoughts on this game. Included is a surprising upper bound.

2.1 The $n = 2$ Communication Game

The $n = 2$ Communication Game is defined as follows. The input consists of k pairs of bits (or two k bit vectors). The question is whether or not the two vectors are the same. Each time step a new player is able to read either the left or the right bit of each pair (but never both). The player then communicates one bit of information to the world. In the end, the answer must be determined by the bits communicated.

More formally, suppose the input is $\langle u, v\rangle \in \{0,1\}^{2k}$. Based on the sequence of bits communicated so far, the algorithm specifes a set $\Pi \subseteq [1..k]$ and a set $S \subseteq \{0,1\}^k$. The player reads the j^{th} bit of u if $j \in \Pi$, otherwise he reads the j^{th} bit of v. Denote the k bit vector that is read by $\Pi(u, v)$. The player communicates a 1 if $\Pi(u, v) \in S$ otherwise he communicates a 0.

A story to make the game more appealing is that we have k pieces of paper, with a bit on each side. The pages are laying on the floor. You can flip them over, but at any one point in time you can only see one side of each page. You can make notes on your pad, but you have no memory about what you have seen before. How many bits must you write down before you know whether each page has the same bit on both sides.

Note that if we restrict the game further so that the players must either read the left bit of each pair or the right bit of each pair, then the game becomes the Karchmer Wigderson universal communication game.

I think that this is a great game, because it characterizes a feature of information that intuitively should be true. The t^{th} player only knows half the bits and t communicated bits. How can he compare his half to the other half unless $t = k$?

2.2 $O\left(k^{1/2}\log^{1/2} k\right)$ Algorithm for the $n = 2$ Communication Game

Mike Luby helped with some of the probability calculations.

Theorem 2 *There exists a* $O\left(\sqrt{k\log k}\right)$ *Algorithm for the* $n = 2$ *game.*

Proof of Theorem 2: Let a, T, and $\tilde{T}$ be parameters to be chosen at the end. Let $C_1, \ldots, C_{k^a}$ be partition of $\{0,1\}^k$ such that for each set C_i, and each pair of distinct vectors $u, v \in C_i$, the hamming distance $H(u, v)$ is at least a.

Aside, I am pretty sure that only k^a *such sets are needed to cover all of* $\{0,1\}^k$*. This is standard coding theory. If anyone is sure about this let me know.*

Let $S_1, \ldots, S_{2^{\tilde{T}}}$ be partition of $\{0,1\}^k$ chosen randomly as follows. For each $u \in \{0,1\}^k$, randomly choose one of the sets for it to be in. Clearly for each set S_i and each vector $u \in \{0,1\}^k$, the probability that $u \in S_i$ is $2^{-\tilde{T}}$.

Let $\Pi_1, \ldots, \Pi_T \subseteq [1..k]$ be chosen randomly as follows. Independently, for each Π_i and each $j \in [1..k]$, add j to Π_i with probability $\frac{1}{2}$.

Given these constructs the algorithm is as follows. Let $\langle u, v\rangle \in \{0,1\}^{2k}$ be the input. First u is read, i.e. $\Pi = [1..k]$. Communicate which C_i that u is contained in. Then read v, i.e. $\Pi = \emptyset$. One bit suffices to tell whether u and v are in the same set. If they are in different C_i sets, then we are done: $u \neq v$. Otherwise, assume that they are in the same C_i. It follows that either $u = v$ or $H(u, v) \geq a$.

The second step is to do the same for the S_i sets. Specifically, which set S_i that u is in and which that v is in. Again assume that u and v are in the same set. Denote the set containing both u and v by S_i.

The last step is as follows. For each Π_t, communicate whether or not $\Pi_t(u, v) \in S_i$. Note that if $u = v$, then $\Pi_t(u, v) = u \in S_i$. Therefore, if $\Pi_t(u, v) \notin S_i$, then $u \neq v$ and we are done.

If all of these tests to prove that $u \neq v$ fail, then assume that $u = v$. The total number of bits communicated is $2a\log(k) + 2\tilde{T} + T$. Hence, we might as well assume that $a\log(k) = \tilde{T} = T$.

Fix an input $\langle u, v\rangle \in \{0,1\}^{2k}$. We need to prove that the probability that the randomly chosen deterministic algorithm fails on input $\langle u, v\rangle$ is small. If $u = v$, the algorithm works. If u and v are in different C_i sets or in different S_i sets, the algorithm works. Therefore, assume that $u \neq v$, $H(u,v) \geq a$, and $u, v \in S_i$.

Consider the set of vectors $\{\Pi_1(u,v), \ldots, \Pi_T(u,v)\}$ that are read in the third step. If this set contains fewer than $T/2$ vectors, then we will assume that the chosen algorithm fails on input $\langle u, v\rangle$. The probability of this can be bound as follows. Suppose that u and v differ in $a' \geq a$ places. For each of these a' places, Π_t selects the bit in u or in the bit in v with equal probability. Each of the $2^{a'}$ of these possibilities leads to a different vector $\Pi_t(u,v)$. Choosing a random Π_t effectively randomly chooses one vector from the set of $2^{a'}$ possibilities. We are selecting such a vector T times. We what to know the probability of getting fewer then $T/2$ distinct vectors. If we do then there is a set of vectors $Q \subseteq \{0,1\}^k$, such that $|Q| = T/2$ and $\{\Pi_1(u,v), \ldots, \Pi_T(u,v)\} \subseteq Q$. Fix a set Q of size $T/2$. Each $\Pi_t(u,v)$ is chosen randomly from a set of size $2^{a'}$ and hence is in Q with probability $\left(\frac{T/2}{2^{a'}}\right)$. Hence, $\{\Pi_1(u,v), \ldots, \Pi_T(u,v)\} \subseteq Q$ with probability $\left(\frac{T/2}{2^{a'}}\right)^T$. However, there are $\binom{2^{a'}}{T/2}$ different set Q. This gives the bound $\left(\frac{T/2}{2^{a'}}\right)^T \binom{2^{a'}}{T/2} \leq 2^{-aT/2}$.

Now assume that $\langle u, v\rangle$ is such that $|\{\Pi_1(u,v), \ldots, \Pi_T(u,v)\}| \geq T/2$. The algorithm fails only if for each Π_t, it is the case that $\Pi_t(u,v) \in S_i$. Consider a $\Pi_t(u,v)$ that is different than u and v. There are at least $T/2 - 2$ such vectors. This vector was put in a random set $S_{i'}$. Hence, the probability that $\Pi_t(u,v) \in S_i$ is $2^{-\tilde{T}}$. Hence, the probability that $\{\Pi_1(u,v), \ldots, \Pi_T(u,v)\} \subseteq S$ is no more than $\left(2^{-\tilde{T}}\right)^{T/2-2}$.

In conclusion, the probability that the algorithm fails on the input $\langle u, v\rangle$ is no more than $2^{-aT/2} + 2^{-\tilde{T}(T/2-2)}$. Hence, the probability that it fails on some input in $\{0,1\}^{2k}$ is bounded by $2^{2k}\left[2^{-aT/2} + 2^{-\tilde{T}(T/2-2)}\right]$. However, if $a\log(k) = \tilde{T} = T \in O\left(\sqrt{(k\log k)}\right)$, then this probability is strictly less than 1. Hence, there exists a deterministic algorithm which works for each input $\langle u, v\rangle$. The number of bits communicated is $2a\log(k) + 2\tilde{T} + T \in O\left(\sqrt{(k\log k)}\right)$. ■

2.3 A $2k^{1/2}$ Algorithm for the $n = 2$ Communication Game

Hey! Nathan Linial was visiting and helped come up with a better algorithm.

Theorem 3 *There is a $2\sqrt{k}$ Algorithm for the $n = 2$ game.*

Proof of Theorem 3: Partition $[1..k]$ into $\sqrt{k}$ equal parts of size $\sqrt{k}$. Let $u = \langle u_1, \ldots, u_{\sqrt{k}}\rangle$ be the partitioning of the vector $u \in \{0,1\}^k$. The algorithm is as follows. Read u. Communicate the sum of the parts $u_1 + \ldots + u_{\sqrt{k}}$. Here sum can either be the bit-wise sum mod 2 or think of u_t as an integer up to $2^{\sqrt{k}}$ and compute the sum mod $2^{\sqrt{k}}$. For $t \in [1..\sqrt{k}]$, let Π_t be such that v_t and $u_1, \ldots, u_{t-1}, u_{t+1}, \ldots, u_{\sqrt{k}}$ are read. Read $\Pi_t(u,v)$ and communicate whether the sum $u_1 + \ldots + u_{t-1} + v_t + u_{t+1} + \ldots + u_{\sqrt{k}}$ is the same. Note that if $u \neq v$ then there is a t such that $u_t \neq v_t$. In this case, $u_1 + \ldots + u_{\sqrt{k}} \neq u_1 + \ldots + u_{t-1} + v_t + u_{t+1} + \ldots + u_{\sqrt{k}}$. ■

2.4 Lower Bounds

Def^n: We say that the probes $\Pi_1, \ldots, \Pi_T \subseteq [1..k]$ **covers** the set $S \subseteq \{0,1\}^k$ if for all distinct pairs of vectors $u, v \in S$, there exists a Π_i such that $\Pi(u,v) \notin S$.
An equivalent definition is
Def^n: Let $\Pi_{-1} = \emptyset$ and $\Pi_0 = [1..k]$. We say that the probes $\Pi_{-1}, \Pi_0, \Pi_1, \ldots, \Pi_T \subseteq [1..k]$ **covers'** all the vectors $\{0,1\}^k$ with the set $\overline{S} \subseteq \{0,1\}^k$ if for all distinct $u, v \in \{0,1\}^k$, there exists a Π_i such that $\Pi(u,v) \in \overline{S}$.
Def^n: For all $\tilde{T}$, let $T(\tilde{T})$ be the minimum T for which there is a set S of size $2^{k-\tilde{T}}$ that can be covered with only T Π's.
Def^n: For all T, let $\tilde{T}(T)$ be the minimum $\tilde{T}$ for which there is a set S of size $2^{k-\tilde{T}}$ that can be covered with only T Π's.

Claim 1 *The functions $T(\tilde{T})$ and $\tilde{T}(T)$ are both monotone non-increasing.*

Proof of Claim 1: If $\Pi_1, \ldots, \Pi_T \subseteq [1..k]$ covers the set S then it covers any subset of S. ■

Lemma 3 *The number of bits of communication required for the $n = 2$ Communication Game is at least $\min_T \max\{T, \tilde{T}(T)\}$, i.e. the point at which $\tilde{T} = T(\tilde{T})$.*

Proof of Lemma 3: Consider an algorithm that communicates at most C bits. For every string of bits $\alpha \in \{0,1\}^C$ communicated, there is a set of inputs $\mathcal{S}_\alpha \in \left(\{0,1\}^k\right)^2$ for which this string is communicated. On input $\langle u, v\rangle$, the bits α communicated must determine whether $u = v$. Hence, each $\mathcal{S}_\alpha$ is either completely on or completely or off the diagonal. Each of the 2^k inputs $\langle u, u\rangle$ on the diagonal must be in one of these sets. Because $\alpha \in \{0,1\}^C$, there are at most 2^C of these sets. Hence, there must be a set $\mathcal{S}_{\alpha'}$ on the diagonal of size at least 2^{k-C}. Let $S \subseteq \{0,1\}^k$ be the set of vectors u such that $\langle u, u\rangle \in \mathcal{S}_{\alpha'}$. Let $\tilde{T}$ be such that $|S| = 2^{k-\tilde{T}}$. Clearly, $C \geq \tilde{T}$.

Let $\Pi_1, \ldots, \Pi_T \subseteq [1..k]$ be the set of probes that the algorithm uses when the string α' is communicated. Clearly, $C \geq T$. What remains is to prove that $\Pi_1, \ldots, \Pi_T$ cover the set S. From this, the lemma follows.

By way of contradiction, assume that $\Pi_1, \ldots, \Pi_T$ does not cover the set S. Then there are distinct vectors $u, v \in S$ such that $\Pi_1(u, v) \in S \wedge \ldots \wedge \Pi_T(u, v) \in S$. Because $u \neq v$, the string α' cannot be communicated on input $\langle u, v\rangle$. Hence, there is a first time step $t \in [1..C]$ on which the communication deviates. Because the communication is the same until time t, the probe Π_t used at time t on input $\langle u, v\rangle$ is the same used at time t with communication α' and in both cases the bit communicated is a fixed function of the string $\Pi_i(u, v)$ read. At time t, the communications for input $\langle u, v\rangle$ deviates from α'. Hence, is the algorithm reads the string $\Pi_t(u, v)$ at time t, it must not write the bit α'_t. On the other hand, by the choice of u and v, $\Pi_t(u, v) \in S$. By the definition of S, on input $\langle \Pi_t(u, v), \Pi_t(u, v)\rangle$, the string α' is communicated. On this input, the probe reads $\Pi_t(\Pi_t(u, v), \Pi_t(u, v)) = \Pi_t(u, v)$. Hence, if the algorithm reads the string $\Pi_t(u, v)$, it must write the bit α'_t. This is a contradiction. ■

Claim 2 *If the set $S \subseteq \{0,1\}^k$ is covered by some list of probes, then S has minimum hamming distance 2, i.e. for all distinct $u, v \in S$, $H(u, v) \geq 2$.*

Proof of Claim 2: Suppose that $u, v \in S$ are distinct and $H(u, v) = 1$. For any probe $\Pi \subseteq [1..k]$, $\Pi(u, v)$ is either u or is v. Hence, $\Pi_i(u, v) \in S$ for each Π_i in the list of probes. ■

Claim 3 *Every set S can be covered with only k probes.*

Therefore, $T(1) = k$. For example, S could be all vectors with even parity.

Proof of Claim 3: For $i \in [1..k]$, let $\Pi_i = \{i\}$, i.e. the player reads all of v except instead of v_i he reads u_i. Consider any distinct vectors $u, v \in S$. Let i be an index on which they are different. Clearly, $H(\Pi_i(u, v), v) = 1$. By Claim 2, $v \in S$ implies that $\Pi_i(u, v) \notin S$. ■

Claim 4 *If S contains all vectors with even parity, then it requires $\Omega(k)$ probes to cover it. If S contains all bits with l ones, then it requires $\Omega\left(\frac{\log\binom{k}{l}}{\log l}\right)$ probes to cover it. For $l = k/2$, this is $\Omega\left(\frac{k}{\log k}\right)$.*

Proof of Claim 4: The proof for parity is easy. The proof for $\binom{k}{l}$ uses the fact that it requires $\log|S|$ bipartite graphs to cover all the edges in the complete graph with vertex set S.

Suppose $S = \binom{k}{l}$ is covered by $\Pi_1, \ldots, \Pi_T$. Think of the graph on vertex set S. Each edge must be covered by some Π_t. Let $E_t = \{\{ab, cd\} \mid ab, cd \in S \text{ and } \Pi_t(ab, cd) = ad \notin S\}$ be the set of edges covered by Π_t. We will break this set into $2\log l$ parts. These parts must 1) cover all edges covered by E_t and 2) each be bipartite. This give a covering of S with $T \times 2\log l$ bipartite graphs. $\log|S|$ bipartite graphs are needed. The result follows.

Let $f : \{a\} \to [1..l]$ and $g : \{b\} \to [1..l]$ with the property that $ab \in S \to f(a) = g(b)$ and $ab \notin S \to f(a) \neq g(b)$. For $S = \binom{k}{l}$, $f(a)$ is the number of 1's in a and $g(b)$ is l minus the number of 1's in b.

For $j \in [1..l]$, define $E_{t,j,0} = \{ab \mid ab \in S \text{ and } [f(a)]_j = 0\} \times \{cd \mid cd \in S \text{ and } [g(d)]_j = 1\}$ and $E_{t,j,1} = \{ab \mid ab \in S \text{ and } [f(a)]_j = 1\} \times \{cd \mid cd \in S \text{ and } [g(d)]_j = 0\}$.

1) These parts cover all edges covered by E_t. If $\{ab, cd\} \in E_t$, then $\Pi_t(ab, cd) = ad \notin S$, then $f(a) \neq g(b)$, then there is and index j in which they are different ...

2) These parts are each bipartite. Specifically, $\{ab \mid ab \in S \text{ and } [f(a)]_j = 0\}$ and $\{cd \mid cd \in S \text{ and } [g(d)]_j = 1\} = \{ab \mid ab \in S \text{ and } [g(b)]_j = 1\}$ are disjoint. This is because if $ab \in S$, then $f(a) = g(b)$.

(The first constraint could be relaxed to $ab \in S' \subseteq S \rightarrow f(a) = g(b)$ as long as the same S' is used for each Π_t. Then we cover the graph on vertex set S'.) ■

Claim 5 $T(\tilde{T}) \in O\left(\frac{k}{\tilde{T}}\right)$

Both Theorem 2 and Theorem 3 give example S and Π.

Claim 6 *If S is a randomly chosen set of size $2^{k-\tilde{T}}$, then $T \in \Omega\left(\frac{k}{\tilde{T}}\right)$ probes are needed.*

The proof is the same as for Theorem 2.

Conjecture 2 $T(\tilde{T}) \in \Omega\left(\frac{k}{\tilde{T}}\right)$.

I.e. the worst cases are random sets S and those S give by Theorem 3. This result would give a $\Omega\left(\sqrt{k}\right)$ lower bound for the $n = 2$ Communication Game.

Lemma 4 *Fix a set S and a probe Π.* $\Pr_{u,v \in S}[\Pi(u,v) \in S] \geq \Pr_{w \in \{0,1\}^k}[w \in S]$.

The argument is as follows. Let $u = \langle a, b\rangle$, $u = \langle c, d\rangle$, and $\Pi(u,v) = \langle a, d\rangle$. The prefix a is chosen randomly by choosing u uniformally from S. Hence, the distribution on a is biased towards those a which have lots of extensions b such that $\langle a, b\rangle \in S$. Similarly for d. Hence the probability that $\langle a, d\rangle \in S$ is higher than if a and d were chosen randomly. Mike Luby, Russell, and I came up with an ugly proof of this (with slight handwaving). There should be an easier proof. However, the next conjecture is even harder.

Conjecture 3 *Fix a set S and probes $\Pi_1, \ldots, \Pi_T$.* $\Pr_{u,v \in \{0,1\}^k}[u \in S \wedge v \in S \wedge \Pi_1(u,v) \in S \wedge \ldots \wedge \Pi_T(u,v) \in S] \in \Omega\left(\Pr_{u,v,w_1,\ldots,w_T \in \{0,1\}^k}[u \in S \wedge v \in S \wedge w_1 \in S \wedge \ldots \wedge w_T \in S]\right)$.

The difficulty in proving this is dealing with the dependencies between the different Π's. However, this would prove Conjecture 4.

Proof: Suppose that S is covered by $\Pi_1, \ldots, \Pi_T$. Let $s = |S|$. $\frac{s}{2^k}\frac{1}{2^k} = \Pr_{u,v \in \{0,1\}^k}[u \in S \wedge u = v]$ $= \Pr_{u,v \in \{0,1\}^k}[u \in S \wedge v \in S \wedge \Pi_1(u,v) \in S \wedge \ldots \wedge \Pi_T(u,v) \in S] \geq \Pr_{u,v,w_1,\ldots,w_T \in \{0,1\}^k}[u \in S \wedge v \in S \wedge w_1 \in S \wedge \ldots \wedge w_T \in S] = \left(\frac{s}{2^k}\right)^{T+2}$. This gives $|S| \leq 2^{\frac{T}{T+1}k} = 2^{k - \frac{1}{T+1}k} = 2^{k-\tilde{T}}$, where $\tilde{T} = \frac{1}{T+1}k$. ■

One strategy is to understand which sets S can be covered by one Π and then by two. This is all that I know so far.

Lemma 5 *$\tilde{T}(1) = 2^{\frac{1}{2}k}$, i.e. the largest S that is covered by 1 Π is of size $2^{\frac{1}{2}k}$.*
$2^{\frac{2}{3}k} \leq \tilde{T}(2) \leq 2^{\frac{3}{4}k}$.

Proof of Lemma 5: There is an S of size $2^{\frac{2}{3}k}$ that can be covered by two Π's. This is given in Theorem 3. Therefore, $\tilde{T}(2) \geq 2^{\frac{2}{3}k}$. $\tilde{T}(2) \leq 2^{\frac{3}{4}k}$ is because the two Πs partition the index set $[1..k]$ into 4 pieces. Suppose that the fourth is the largest. Knowing 3 of the pieces of a $u \in S$ must determine the fourth. Otherwise, there are two vectors $\langle abcd\rangle$ and $\langle abcz\rangle$ both in S. $\Pi(\langle abcd\rangle, \langle abcz\rangle)$ reads either all of d or all of z. Hence, what is read is either $\langle abcd\rangle$ or $\langle abcz\rangle$. However, both are in S. ■
The next goal is to prove that $\tilde{T}(2) = 2^{\frac{2}{3}k}$.

3 New Card Game

Previously a card game was considered. It was proved that a non-trivial lower bound for this game with cheating gives a non-trivial lower bound for the time-space trade off for element distinctness on a branching program. The impressive upper bound by Pavel Pudlák and Jiri Sgall dashed any hope of us finding a lower bound for this card game. This sketch of a paper defines a more general card game and proves that a non-trivial lower bound for this game with cheating gives non-trivial lower bound for the time-space trade off (for some boolean function) on a branching program.

3.1 The More General Card Game

Let $f : \{0,1\}^q * \{0,1\}^q \to \{0,1\}$ define the communication game in which the A-player is given $u \in \{0,1\}^q$, the B-player is given $v \in \{0,1\}^q$, they communicate, and then answer $f(u,v)$. This function f can be any function of your choice for which you can get a result. My intuition is that $q = 2$ bits are sufficient for our purposes. For example, let $f(u,v) = [u]_1 \wedge [v]_1 \vee [u]_2 \wedge [v]_2$. The communication complexity of this game is 2.

From this communication game f, we form the following card game. The inputs to the card game are n independent inputs to the communication game, i.e. $u =< u_1, u_2, \ldots, u_n >$ and $v =< v_1, v_2, \ldots, v_n > \in [\{0,1\}^q]^n$. What needs to be computed is the boolean answer $f(u_1,v_1) \wedge \ldots \wedge f(u_n,v_n)$. There is a different player for each time step. At time t, the t^{th} player first reads the blackboard so that he knows what the previous players have communicated. Then for each of the communication games $i \in [1..n]$, he either reads the u_i input or the v_i input, but not both. The idea is that there are n cards. On one side of the i^{th} card is u_i and on the other is v_i. The player can look at one side of each card. He makes this selection by specifying a function $\Pi : [1..n] \to \{u,v\}$. Based on all this information, he writes a single bit on the black board. At the end, the last player must determine whether $f(u_1,v_1) \wedge \ldots \wedge f(u_n,v_n)$. The complexity is the number of bits written on the black board.

The original card game is the above game in which $q = 1$ and $f(u,v) = 1$ iff $u = v$. Pavel Pudlák and Jiri Sgall gives the complexity of this game to be at most $n^{2/5}$. The question is what can be said about this card game for different functions f.

The $2\sqrt{qn}$ upper bound still holds for an arbitrary game f. I don't under stand Jiri's paper well enough to know whether his $n^{2/5}$ holds for an arbitrary game f, but I doubt it.

3.2 The $2(qn)^{\frac{1}{2}}$ Upper Bound

The $2\sqrt{qn}$ upper bound to the card game for a general communication game f goes as follows. Partition the index set $[1..n]$ into $\sqrt{qn}$ parts of size $\sqrt{n/q}$ each. Use this partition to partition the input $u = \langle u_1, \ldots, u_n \rangle \in [\{0,1\}^q]^n$ into the vectors $\langle \overline{u}_1, \ldots, \overline{u}_{\sqrt{qn}} \rangle$. Think of each $\overline{u}_i = \left\langle u_j, \ldots, u_{j+\sqrt{n/q}} \right\rangle \in [\{0,1\}^q]^{\sqrt{n/q}}$ as a number between 0 and $2^{\sqrt{qn}} - 1$. Similarly partition $v =< v_1, v_2, \ldots, v_n >$. In the first $\sqrt{qn}$ time steps, the player reads all of $u = \langle \overline{u}_1, \ldots, \overline{u}_{\sqrt{qn}} \rangle$ and communicates $[\overline{u}_1 + \ldots + \overline{u}_{\sqrt{qn}}]$ *mod* $2^{\sqrt{qn}}$. In the t^{th} remaining step, $t \in [1..\sqrt{qn}]$, the player learns $[\overline{u}_1 + \ldots + \overline{u}_{\sqrt{qn}}]$ *mod* $2^{\sqrt{qn}}$ from the black board and he reads $\overline{v}_t$ and $\overline{u}_1, \ldots, \overline{u}_{t-1}, \overline{u}_{t+1}, \ldots, \overline{u}_{\sqrt{qn}}$. From this he computes $\overline{u}_t = [\overline{u}_1 + \ldots + \overline{u}_{\sqrt{qn}}] - [\overline{u}_1 + \ldots + \overline{u}_{t-1} + \overline{u}_{t+1} + \ldots + \overline{u}_{\sqrt{qn}}]$ *mod* $2^{\sqrt{qn}}$. He now knows both $\overline{u}_t$ and $\overline{v}_t$. These comprise the $\langle u_i, v_i \rangle$ for $\sqrt{n/q}$ of the communication games. He computes $f(u_i,v_i)$ for each of these and writes on the black board one bit indicating $f(u_i,v_i) \wedge \ldots \wedge f(u_{i+\sqrt{n/q}}, v_{i+\sqrt{n/q}})$. From this, the last player knows $f(u_i,v_i) \wedge \ldots \wedge f(u_n,v_n)$.

3.3 The Card Game with Cheating

The card game with cheating is the same as the card game, except some cheating is allowed. An algorithm, on a given input, cheats with a card $i \in [1..n]$ if ever a player looks at both sides of the card at the same time. An algorithm for the card game with cheating requires that for a random input $\langle u,v \rangle$, the probability that cheating is done on more than ϵn cards is no more than $\frac{1}{2}$. Here the distribution on inputs $\langle u,v \rangle$ can be anything you like. For example, for the original game, there was an motivation for looking only at the cheating done on random inputs of the form $\langle u,u \rangle$.

3.4 Branching Program Lower Bound

Theorem 4 *A lower bound of n^ω for the card game with cheating gives a lower bound of $TS^{1/2} \in \Omega\left(n^{1+\omega/2}\right)$ for some boolean function on oblivious branching programs.*

Proof of Theorem 4: The function F computed by the branching program has inputs $\langle \pi, u, \tau, v \rangle$, where $u = \langle u_1, \ldots, u_n \rangle, v = \langle v_1, \ldots, v_n \rangle \in [\{0,1\}^q]^n$ and $\pi = \langle \pi(1), \ldots, \pi(n) \rangle$ and $\tau = \langle \tau(1), \ldots, \tau(n) \rangle$ are each permutatutions of $[1..n]$. It is defined by

$$F(\pi, u, \tau, v) \;=\; f\left(u_{\pi^{-1}(1)}, v_{\tau^{-1}(1)}\right) \wedge \ldots \wedge f\left(u_{\pi^{-1}(n)}, v_{\tau^{-1}(n)}\right).$$

When the permutations π and τ are understood, let $\hat{u}$ be such that $\hat{u}_j = u_{\pi^{-1}(j)}$ ($u_i = \hat{u}_{\pi(i)}$) and similarly $\hat{v}_j = v_{\tau^{-1}(j)}$ ($v_i = \hat{v}_{\tau(i)}$). Another way of thinking of F is that it input is formed as follows. Choose an input $\hat{u} = \langle \hat{u}_1, \ldots, \hat{u}_n \rangle, \hat{v} = \langle \hat{v}_1, \ldots, \hat{v}_n \rangle \in [\{0,1\}^q]^n$ to the card game. F will have the same output as the card game does on $\langle \hat{u}, \hat{v} \rangle$. Form the matrix with n rows and two columns:

$\langle 1, \hat{u}_1 \rangle$	$\langle 1, \hat{v}_1 \rangle$
$\langle 2, \hat{u}_2 \rangle$	$\langle 2, \hat{v}_2 \rangle$
$\vdots$	$\vdots$
$\langle j, \hat{u}_j \rangle$	$\langle j, \hat{v}_j \rangle$
$\vdots$	$\vdots$
$\langle n, \hat{u}_n \rangle$	$\langle n, \hat{v}_n \rangle$

Now choose two permutations π^{-1} and τ^{-1} and use these to permute the two columns, i.e. if $\pi(i) = j$ then the i^{th} row of the first column becomes $\langle j, \hat{u}_j \rangle = \langle \pi(i), u_i \rangle$. To simplify the proof (and not hurt the lower bound by more than a factor of $\log n$), we consider branching programs which each node of the computation are able to access a complete matrix entry $\langle \pi(i), u_i \rangle$ (or $\langle \tau(i), v_i \rangle$) for some fixed $i \in [1..n]$.

By way of contradiction, consider a branching program that computes F and for which $TS^{1/2} < \sqrt{\frac{\epsilon}{2}} n^{1+\omega/2}$. From this branching program we construct a card game algorithm with cheating. This algorithm will contradict the claimed lower bound. To construct the algorithm, fix two permutations $\langle \pi, \tau \rangle$. Given these, there is a fixed 1-1 mapping between the inputs $\langle u, v \rangle$ to the branching program and the inputs $\langle \hat{u}, \hat{v} \rangle$ to the card game. Recall that $\hat{u}_j = u_{\pi^{-1}(j)}$ ($u_i = \hat{u}_{\pi(i)}$) and similarly for $\hat{v}$.

Break the branching program into T/h stages (layers) of height h, where $h = \frac{\epsilon}{2} n^2/T$. This breaks the branching program into at most 2^S sub-branching programs of height h. The card game algorithm has T/h stages as well. Each stage, the players write on the black board the index of the sub-branching program that the branching program computation is at the top of at the end of the stage. It requires S bits to write down this index. Therefore, each stage of the card game algorithm consists of S time steps, for a total of ST/h times steps. Plugging in $h = \frac{\epsilon}{2} n^2/T$ gives $\frac{2}{\epsilon} T^2 S/n^2$. Plugging in $TS^{1/2} < \sqrt{\frac{\epsilon}{2}} n^{1+\omega/2}$, gives that card game algorithm requires fewer than n^ω times steps. This contradicts the claimed lower bound.

By reading the black board, the first player in a stage knows which sub-branching program the computation is at the top of. Because the sub-branching program is oblivious, it access a fixed set of h branching program variables $\langle \pi(i_1), u_{i_1} \rangle$ (or $\langle \tau(i_1), v_{i_1} \rangle$), ..., $\langle \pi(i_h), u_{i_h} \rangle$ (or $\langle \tau(i_h), v_{i_h} \rangle$). Because the permutations $\langle \pi, \tau \rangle$ are fixed, this corresponds to accessing a fixed set of h card game variables $\hat{u}_{j_1}$ (or $\hat{v}_{j_1}$), ..., $\hat{u}_{j_h}$ (or $\hat{v}_{j_h}$). The first player in the stage accesses these variables. Note that if the branching program reads both $\langle \pi(i), u_i \rangle$ and $\langle \tau(i'), v_{i'} \rangle$ and it happens that $j = \pi(i) = \tau(i')$, then the player reads both $\hat{u}_j$ and $\hat{v}_j$. This of course cheating, but the card game is allowed a cirtain amount of cheating. The amount of cheating will be bound latter.

After the player reads these variable, he knows how the branching program computation will proceed through the sub-branching program and hence he knows which sub-branching program the stage will end at. The index of this sub-branching program needs to be written on the black board. This player can only write one bit, so he writes the first bit in the index. The remaining $S-1$ players in the stage read the same variables, gaining the same information, and write the remaining bits of the index. (Note by the definition of cheating, the algorthm is penalized once for j, even though $\hat{u}_j$ and $\hat{v}_j$ are both read by many players.) This completes the definition of the card game algorithm.

What remains is choose the fixed permutations $\langle \pi, \tau \rangle$ so that the probability that card game algorithm cheats on more than ϵn cards is no more than $\frac{1}{2}$ when its input $\langle \hat{u}, \hat{v} \rangle$ is choosen randomly. Consider some fixed sub-branching program. We say that the sub-branching program makes progress if it reads the same value j on both sides of the input matrix. Clearly progress for the branching program corresponds to cheating for the card game algorithm. The sub-branching program reads h entries in matrix. For any fixed value $j \in [1..n]$, the probability (over the random $\langle \pi, \tau \rangle$ and $\langle u, v \rangle$) of the sub-branching program reading j on

both sides of the input matrix is at most $(h/n)^2$. Therefore, the expected number of such j is at most h^2/n. Chernoff says the probabity of getting twice the expected at most $2^{-c*expected}$ for some constant c. Therefore, the probability that at least $2h^2/n$ progress is made in this sub-branching program is at most $2^{-ch^2/n}$. Because there are fewer then 2^S different sub-branching programs, the probability that there exists a sub-branching program in which at least $2h^2/n$ progress is made is at most $2^S 2^{-ch^2/n}$. We want this probability to be at most $\frac{1}{2}$. This requires that $S < c'h^2/n$. Plugging in $h = \frac{\epsilon}{2}n^2/T$ gives the requirement $S < \epsilon' n^3/T^2$ or that $TS^{1/2} < \epsilon'' n^{3/2}$. However, we assumed that $TS^{1/2} < \sqrt{\frac{\epsilon}{2}} n^{1+\omega/2} << \epsilon'' n^{3/2}$, so the requirement is met.

The progress made for an input in the entire branching program is the sum of the progress made in each sub-branching program that the input passes through. An input passes through at most T/h stages. Therefore, if an input makes ϵn progress total then there exists a sub-branching program in which it makes at least $\epsilon nh/T$ progress. Plugging in $h = \frac{\epsilon}{2}n^2/T$ gives $\epsilon nh/T = 2(h^2)/n$. As said above, the probability (over the random $\langle \pi, \tau \rangle$ and $\langle u, v \rangle$) that at least $2(h^2)/n$ progress is made in some sub-branching program is at most $\frac{1}{2}$. Therefore, this is the probabilty that ϵn progress is made in the entire branching program. Fix some two permutations $\langle \pi, \tau \rangle$ such that the probability (over the input $\langle u, v \rangle$) is still at most $\frac{1}{2}$. These are the permutations that are used in the algorithm above. ■

4 Bibliography

- P. Pudlak, J. Sgall: *An upper bound for a communication game related to space-time tradeoffs,* In The Mathematics of Paul Erdos, volume I, eds. Ronald L. Graham and Jaroslav Nesetril, pages 393-399. Springer, 1996.