Lectures on the Fusion Method and Derandomization

Avi Wigderson
Hebrew University of Jerusalem

January 28 – February 3, 1994

Technical Report SOCS-95.2
School of Computer Science
McGill University

Foreword

December 15, 1994

The 1994 McGill-Montréal Invitational Workshop on Complexity Theory was held at McGill University's Bellairs Research Institute in Barbados from January 28 to February 3, organized by Pierre McKenzie (U. de Montréal) and Denis Thérien (McGill U.).

The core of this year's workshop was a series of lectures given by Avi Wigderson from the Hebrew University of Jerusalem. Two main themes were treated: the fusion method for proving lower bounds, and the question of deterministic amplification of probabilistic algorithms using expanders and universal hashing. The participants thus had the privilege of being given a solid coverage of many beautiful recent results.

Early on in the week, a consensus was reached that a permanent record of the lectures should be kept as it would be of interest to several researchers in the community. This idea led to the present technical report in which the content of the five two-hour lectures given by Avi can be found in the form of notes taken by attendees. The presentation has been kept rather informal and no attempt has been made to overpolish the material.

Warm thanks go to the note-takers: E. Allender (Rutgers), F. Lemieux (McGill), D. M. Barrington (U. Mass at Amherst), A. Maciel (McGill), R. Gavaldà (U. Politècnica de Catalunya), P. McKenzie (U. de Montréal), D. Thérien (McGill U.), R. Beigel (Yale), H. Caussinus (U. de Montréal) and M. Saks (Rutgers). Above all, our very deep and sincere appreciation to Avi Wigderson for a most enjoyable week.

Denis Thérien

Contents

The Fusion Method I 1
The Fusion Method II 9
Expanders and Pairwise Independence 18
Improving Randomized Algorithms via Universal Hashing 31
Applications of Universal Hashing 42

The Fusion Method I

Notes by Eric Allender and François Lemieux

Saturday Evening, January 29th

1 Introduction

The *fusion method* is an approach for proving lower bounds on the size of circuits and branching programs. Basically, it proceeds by reducing a lower bound problem to a set covering problem. In these notes, we study the relationship between these two problems in five different contexts. The results we present appear (sometimes only implicitely) in [4], [5], [1], [2] and [3].

A (deterministic) *Straight Line Program* (*SLP*) $P = \langle g_1, g_2, \ldots, g_t \rangle$ over an input $x = x_1 \cdots x_n$ is a sequence of instructions (or gates) $g_i : \{0,1\}^n \to \{0,1\}$ of three different types:

1. $g_i = x_i$, for $1 \leq i \leq n$,
2. $g_{n+i} = \bar{x}_i$, for $1 \leq i \leq n$, and
3. $g_i = g_{i_1} \circ_i g_{i_2}$, for $2n < i \leq t$, $i_1, i_2 < i$, and $\circ_i \in \mathrm{OP}$, the set of binary Boolean operations used by P. (Sometimes, it will also be convenient to use the constants 0 and 1 instead of g_{i_1} and/or g_{i_2}.)

The computation of P on x is defined as $P(x) = g_1(x) \cdots g_t(x) \in \{0,1\}^t$. P is said to compute a function $f : \{0,1\}^n \to \{0,1\}$ if $g_t(x) = f(x)$ for all $x \in \{0,1\}^n$.

Both Boolean circuits and Branching Programs (BP) can be represented as *SLP*. For circuits, it suffices to take $\mathrm{OP} = \{\wedge, \vee\}$. For BP we also take $\mathrm{OP} = \{\wedge, \vee\}$ but we restrict the instructions of the third type by forcing the operation $\wedge$ to be of the form $g_i = g_j \circ_i x_k^\epsilon$, where $j < i$, $1 \leq k \leq n$, $\epsilon \in \{0,1\}$, $x_i^0 = \overline{x_i}$, and $x_i^1 = x_i$. (We use here a characterization of BP in term of *skew*

circuits, i.e., those circuits where $\wedge$-gates have at least one input that is a variable or a constant.) Monotone version of both models can be defined simply by removing the instructions of the second type. Nondeterministic Straight Line Programs will be discussed in the next lecture.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, define $U = f^{-1}(0)$, and let $\Omega \subseteq \{\omega : 2^U \to \{0,1\}\}$ where 2^U is the power set of U seen as either the Boolean lattice $\{0,1\}^U$ or the vector space $GF(2)^U$. In particular, given a *SLP* P, we denote by g_i the set of elements $u \in U$ such that $g_i(u) = 1$, and by x_i the set of strings in U having a 1 in their i-th position. A function $\omega \in \Omega$ is said to *define* $z \in \{0,1\}^n$ if for any i such that $1 \leq i \leq n$, we have $\omega(g_i) = z_i$ and $\omega(g_{n+i}) = \bar{z}_i$. We will be interested to determine if there is some $\omega \in \Omega$ having the following properties.

1. ω is *rejecting*, namely $\omega(\bar{0}) = 0$
2. ω *defines a 1* of f, namely ω defines some $v \notin U$.
3. ω is *consistent*, namely $\omega(g_{i_1}) \circ_i \omega(g_{i_2}) = \omega(g_{i_1} \circ_i g_{i_2})$ for $n < i \leq t$.

For any *SLP* we have the following result.

Lemma 1.1 *If P rejects precisely U then, there is no $\omega \in \Omega$ that satisfies all three properties.*

Proof. Suppose that $\omega \in \Omega$ satisfies all three properties, and consider the following table of computations of P over U.

	g_1	$\cdots$	g_n	$\cdots$	g_i	$\cdots$	g_t
u_1	0	$\cdots$	1	$\cdots$	1	$\cdots$	0
u_2	1	$\cdots$	0	$\cdots$	1	$\cdots$	0
$\vdots$							$\vdots$
$u_{\lvert U\rvert}$							0
	$\omega(g_1)$	$\cdots$	$\omega(g_n)$	$\cdots$	$\omega(g_i)$	$\cdots$	0

The last row $\omega(g_1)\cdots\omega(g_t)$ is such that $\omega(g_t) = 0$ and $\omega(g_1)\cdots\omega(g_n) = v \notin U$. Furthermore, ω is consistent. This means that P rejects the input v, contradicting the fact that P rejects only U. ∎

Observe that the first two properties depend only on f (not on P). Hence, we can define $\Omega_f = \{\omega \in \Omega : \omega$ satisfies properties 1 and 2$\}$ and reformulate the above lemma as follows.

Lemma 1.2 *If P rejects precisely U then, no $\omega \in \Omega_f$ can satisfy property 3.* ■

Let $g, h \in \{0,1\}^U$ and $\circ \in OP$. A triple $\langle g, h, \circ \rangle$ is said to *cover* $\omega \in \Omega_f$ if and only if $\omega(g) \circ \omega(h) \neq \omega(g \circ h)$. We denote by $\rho_\Omega(f)$ the minimum number of triples needed to cover Ω_f. Let $s_{\mathrm{OP}}(f)$ be the size of the smallest *SLP* that computes f. When $\mathrm{OP} = \{\vee, \wedge\}$, we simply write $s(f)$.

Theorem 1.3 $s(f) \geq \rho_\Omega(f)$

Proof. If $P = \langle g_1, \ldots, g_t \rangle$ is a *SLP* computing f then, by Lemma 1.2, no function in Ω_f is consistent with P. Also, each gate $g_i = g_j \circ g_k$ corresponds to a triple $\langle g_j, g_k, \circ \rangle$. Hence, taking all such triples we get a cover set for Ω_f of size bounded by $\rho_\Omega(f)$. ■

In the case where $OP = \{\wedge, \vee\}$, and every $\omega \in \Omega$ is monotone (i.e. $\Omega = \mathit{FILTERS}$), it is sufficient to consider triples of the form $\langle g, h, \wedge \rangle$. Hence, defining $\widehat{\rho}_{\mathit{FILTERS}}(f)$ to be the minimum number of such triples that cover $\mathrm{FILTERS}_f$ (i.e. Ω_f with $\Omega = \mathrm{FILTERS}$), we get a stronger version of Theorem 1.3.

Theorem 1.4 $s(f) \geq \widehat{\rho}_{\mathit{FILTERS}}(f)$

Proof. Let $P = \langle g_1, \ldots, g_t \rangle$ be a *SLP* computing f. Let $\omega \in \mathrm{FILTERS}_f$. So, ω defines a 1 (say v) of f and $\omega(\bar{0}) = 0$. Considering the proof of Theorem 1.3, it suffices to show that if a $\vee$-gate covers ω then, ω is also covered by an $\wedge$-gate. Suppose the contrary: let S be the set of gates in P that cover ω, and assume that S contains no $\wedge$-gate.

By the definition of cover we have that for each gate $g_i = g_j \vee g_k$ in S, $\omega(g_j) \vee \omega(g_k) \neq \omega(g_j \vee g_k)$, and since ω is monotone then the only possibility is that $\omega(g_j) = \omega(g_k) = 0$ and $\omega(g_j \vee g_k) = 1$.

Let P' be a *SLP* identical to P except that each gate $g_i = g_j \vee g_k$ in S is replaced by the instruction $g'_i = 1 \vee 1$.

Two things must be observed concerning P'. First, on input v, the ouput of P' must be 1, like P. This is because we have only changed gates g_i, whose values were 0 on input v, for gates g'_i with constant value 1. Since the programs use only $\vee$-gates and $\wedge$-gates, which are monotone, the output of P' cannot be 0 while that of P is 1. Furthermore, the computation of P'

on input v is $\omega(g_1)\omega(g_2)\cdots\omega(g_t)$. This is easily proved by induction on the position of the gates in P'.

Putting together our observations, we get that $g'_t(v) = g_t(v) = 1$, on one side, and $g'_t(v) = \omega(g_t) = 0$, on the other side. Thus, we have a contradiction, showing that S cannot contain only $\vee$-gates. This concludes the proof. ∎

Knowing that $s(f) \geq \rho(f)$ implies that in proving a lower bound on the circuit complexity of f, it suffices to bound $\rho(f)$. One possible approach would be to express this as an integer programming problem, and then try to bound the associated linear program (i.e. changing the set cover to a fractional cover). However, Razborov [4] showed that this approach can never prove a nonlinear bound on $\rho(f)$.

In some case, we will also be able to prove a "converse" of the form $s(f) \leq \rho(f)^{O(1)}$. The next table shows some results of this form that have been obtained, by varying the class of functions Ω used in defining Ω_f, and by varying the model of computation used in defining the Straight-Line Programs.

	Circuits	Branching Programs
FILTERS	P/poly	NL/poly
AFFINE	NP/poly	$\oplus L$/poly
ULTRAFILTERS	NP/poly	?

By circuits we mean deterministic, ordinary, circuits in line 1, but in line 2 and 3 we mean nondeterministic circuits, where there are nondeterministic inputs y_i corresponding to guessed bits.

Nondeterministic Branching programs are directed acyclic graphs with a start and a final vertex, where each edge is labelled "1", "$x_i = 1$" or "$x_i = 0$". The program accepts input x iff there is a path from the start to the final vertex where each edge on the path evaluates to true on input x. (This is equivalent to a skew circuit where edges correspond to $\wedge$-gates, where one input to the $\wedge$-gate is either $x_i, \bar{x}_i$, or 1, and vertices correspond to $\vee$-gates whose inputs are the edges leading into the vertex.) This model is easily seen to characterize NL/poly. If, instead of $\vee$-gates, we have $\oplus$-gates, it characterizes $\oplus L$/poly.

By *FILTERS* we mean nothing more than the set of all monotone functions ω such that $\omega(\emptyset) = 0$.

By *ULTRAFILTER* we mean all $\omega \in$ *FILTERS* that satisfy the additional property that $\omega(S) \neq \omega(\bar{S})$, for each $S \subseteq U$. (The name *ULTRAFILTERS*

is normally used only in the context of infinite domains U, where they have been used in many classical proofs in logic. Using the name *ULTRAFILTERS* in this finite context is a recent development.)

By *AFFINE* we mean all functions on 2^U that can be expressed as taking the parity of certain odd number of bits position. This corresponds to a class of linear functions on $\mathrm{GF}(2)^{|U|}$. (More precise definitions will be given later.)

As a consequence of these characterizations, $P \neq NP$ if and only if there is an f whose associated set of ultrafilters is easy to cover, but whose associated set of filters is hard to cover. (The set of filters is much larger.)

2 Circuits and filters

Now we give a first example of a converse of Theorem 1.3 for the choice $\Omega = \mathit{FILTERS}$ and Model=Circuits.

Theorem 2.1 ([4]) $s(f) \leq \widehat{\rho}_{\mathit{FILTERS}}(f)^2$

Proof. Given an (unordered, unstructured) collection $\mathcal{C} = \{\langle g_1, h_1\rangle, \ldots, \langle g_t, h_t\rangle\}$ constituting a cover (all operations are $\wedge$), we need to build a circuit. It will be convenient to consider the following $|U| \times (2n + 2^U)$ matrix M, the columns of which represent all possible inputs to $\omega \in \Omega_f$.

	$x_1 x_2$	$\cdots$	x_n	$\bar{x}_1 \bar{x}_2$	$\cdots$	$\bar{x}_n$	$g_1 g_2$	$\cdots$	$g_{2^{\lvert U\rvert}}$
u_1							11		0
$\vdots$		$\vdots$			$\vdots$		$\vdots$	$\cdots$	$\vdots$
u_i	$u_{i_1} u_{i_2}$	$\cdots$	u_{i_n}	$\bar{u}_{i_1} \bar{u}_{i_2}$	$\cdots$	$\bar{u}_{i_n}$	10	$\cdots$	0
$\vdots$		$\vdots$			$\vdots$		$\vdots$	$\cdots$	$\vdots$
$u_{\lvert U\rvert}$							01		0

The principle that will guide the operation of the circuit in this construction (as well as in the proofs of the other "converses") is captured in the following lemma:

Lemma 2.2 *$f(z) = 0$ iff there exists ω defining z such that ω is monotone, $\omega(\bar{0}) = 0$, and ω is consistent with cover $\mathcal{C}$.*

Proof. ($\Rightarrow$) Assume $f(z) = 0$. Define $\omega_z(g)$ to be $g(z)$ for any $g \subseteq U$. (i.e., ω_z outputs the bit in row z of the column g given as input.) Clearly ω_z is monotone, defines z, and for every g, h (not just those in $\mathcal{C}$), $\omega_z(g) \wedge \omega_z(h) = \omega_z(g \wedge h)$.

($\Leftarrow$) Let $f(z) = 1$. Then, by definition of a cover, for all ω defining z such that ω is monotone and $\omega(\bar{0}) = 0$, ω is not consistent with $\mathcal{C}$. ∎

Thus, the task our circuit will perform, on input z, is to determine if there is a monotone, nontrivial ω defining z, such that ω is consistent with $\mathcal{C}$. Now, we will explain how to build the circuit.

Let $\mathcal{S} = \{\emptyset, x_1 x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots \bar{x}_n, g_1, h_1, g_1 \cap h_1, \ldots, g_t, h_t, g_t \cap h_t\}$. (Note that we consider elements of $\mathcal{S}$ to be subsets of U, instead of elements of $\{0,1\}^U$.)

Our circuit will be given (for free) the following information about $\mathcal{S}$ (i.e., information about $\mathcal{C}$):

For all $S, T \in \mathcal{S}$, we are told if $S \subseteq T$ and/or $T \subseteq S$. Note that this is $O(n+t)^2$ bits of information that is "built into" the circuit.

Using that information, the circuit executes the following program:

Begin
If $z_i = 1$ **then** set $\omega(x_i) \leftarrow 1$;
If $z_i = 0$ **then** set $\omega(\bar{x}_i) \leftarrow 1$;
  Repeat
  (1) **If** ($\exists S, T \in \mathcal{S}$, $T \subseteq S$, $\omega(T) = 1$, $\omega(S)$ not set)
    then set $\omega(S) \leftarrow 1$;
  (2) **If** ($\exists i$, $\omega(g_i) = \omega(h_i) = 1$, $S = g_i \cap h_i$, $\omega(S)$ not set)
    then set $\omega(S) \leftarrow 1$;
  Until a pass through the loop is made
    with no $\omega(S)$ getting set;
Return 1 iff ($\omega(\emptyset) = 1$) or ($\exists i$ $\omega(x_i) = \omega(\bar{x}_i)$)
End.

Clearly the number of iterations of the **Repeat** loop is $O(n+t)$. Thus, there is a circuit of size about $O(n+t)^2$ executing the program. The important fact about the operation of the program can be stated as follows.

Fact 2.3 *Our program sets $\omega(S) = 1$ if and only if for every monotone ω' such that*

1. $\omega'(x_i) = 1$ *if* $z_i = 1$

2. $\omega'(\bar{x}_i) = 1$ *if* $z_i = 0$

3. ω' *is consistent with* $\mathcal{C}$

it is the case that $\omega'(S) = 1$.

To see that Fact 2.3 is true, note that $\omega(S) = 1$ iff ($S = x_i$ and $z_i = 1$), ($S = \bar{x}_i$ and $z_i = 0$), or $\omega(S) = 1$ is a logical consequence of either monotonicity (line (1) in the program), or consistency with $\mathcal{C}$ (line (2) in the program). That is, the ω constructed by the program has the properties in Fact 2.3, and no ω' having the properties in Fact 2.3 has $\omega'(S) < \omega(S)$ for any $S \in \mathcal{S}$. (If you are concerned that ω is defined only on $S \in \mathcal{S}$, then define $\omega(g) = 1 \Leftrightarrow (\exists S \in \mathcal{S},\ S \subseteq g,\ \omega(S) = 1)$. Then ω is monotone, consistent with $\mathcal{C}$, and $\omega(x_i) = 1$ if $z_i = 1$, and $\omega(\bar{x}_i) = 1$ if $z_i = 0$.)

Now, using Fact 2.3 let's check that our program is correct. That is, we need to show:

$$f(z) = 1 \Leftrightarrow (\omega(\emptyset) = 1) \text{ or } (\exists i \leq n,\ \omega(x_i) = \omega(\bar{x}_i) = 1)$$

($\Rightarrow$) If $f(z) = 1$ then, by Lemma 2.2, every monotone ω' that is consistent with $\mathcal{C}$ either fails to define z or has $\omega'(\bar{0}) = 1$ (i.e., $\omega'(\emptyset) = 1$). Thus every monotone ω' such that $(z_i = 1 \Rightarrow \omega'(x_i) = 1) \wedge (z_i = 0 \Rightarrow \omega'(\bar{x}_i) = 1)$ and ω' is consistent with $\mathcal{C}$ either has $\omega'(\emptyset) = 1$ or it fails to define z. (You can easily verify that if such an ω' fails to define z then, there exists an i such that $\omega'(x_i) = \omega'(\bar{x}_i) = 1$.) Since this is true for any such ω', it is also true for the ω constructed by our program.

($\Leftarrow$) If $f(z) = 0$, then by Lemma 2.2 there is a monotone, consistent ω' defining z, and $\omega'(\emptyset) = 0$. Since this ω' defines a string z then, by definition, we have that for all i, $\omega'(x_i) \neq \omega'(\bar{x}_i)$. Thus, by Fact 2.3 (letting $S = \emptyset$), $\omega(\emptyset) = 0$. It remains to show that there is no i such that $\omega(x_i) = \omega(\bar{x}_i) = 1$. Suppose the contrary. Then, any monotone ω' satisfying the properties of Fact 2.3 must be such that $\omega'(x_i) = \omega'(\bar{x}_i) = 1$. Hence, no such ω' defines z and, by Lemma 2.2, $f(z) = 1$, a contradiction.

This concludes the proof of Theorem 2.1. ∎

References

[1] M. Karchmer. On proving lower bounds for circuit size. In *Proceedings of the 8th Annual Symposium on Structure in Complexity Theory*, 1993.

[2] M. Karchmer and A. Wigderson. Characterizing non-deterministic circuit size. In *Proceedings of the 25th STOC*, 1993.

[3] M. Karchmer and A. Wigderson. On span programs. In *Proceedings of the 8th Annual Symposium on Structure in Complexity Theory*, 1993.

[4] A. Razborov. On the method of approximation. *21st STOC*, pages 167–176, 1989.

[5] A. Razborov. Lower bounds on the size of switching-and-rectifier networks for symmetric Boolean functions (in Russian). *Mathematical Notes of the Academy of Sciences of the USSR*, 48(6):79–91, 1990.

The Fusion Method II

Notes by David Barrington and Alexis Maciel

Sunday Morning, January 30th

1 Nondeterministic Branching Programs and Filters

We now consider the nondeterministic branching program model in the context of filters. We will obtain a characterization of NL as those functions that have polynomial cover size [4].

A nondeterministic branching program is a directed graph with two distinguished nodes s and t and some edges labeled by x_i^1 or x_i^0, $1 \leq i \leq n$. Every input $z = z_1 \cdots z_n$ determines *true* edges and *false* edges: edges labeled x_i^ε are true if and only if $z_i = \varepsilon$, and unlabeled edges are always true. An input *reaches* a node v if there is a path of true edges from s to v. A node v computes the function f_v defined by $f_v(z) = 1$ if and only if z can reach v. The function computed by the branching program is $f = f_t$. The *size* of a branching program is the number of labeled edges. Let $\mathrm{BP}(f)$ be the minimal size of a branching program computing f. Consider $U = f^{-1}(0)$ and let $g_v = \{u \in U : u \text{ can reach } v\}$. Of course, $g_s = U$ and $g_t = \emptyset$.

As was discussed yesterday, we consider the problem of covering all non-trivial filters in 2^U that define ones of f with pairs $\langle g, x_i^\varepsilon \rangle$, where $g \subseteq U$ and $x_i^\varepsilon = \{u \in U : u_i = \varepsilon\}$, $\varepsilon = 0, 1$. Recall that $\langle g, x_i^\varepsilon \rangle$ covers ω if $\omega(g) \wedge \omega(x_i^\varepsilon) \neq \omega(g \cap x_i^\varepsilon)$. Let $\rho(f)$ be the minimum number of pairs needed to cover all non-trivial filters ω that define some $z \notin U$.

We first show that the branching program complexity of a function is bounded below by its cover size, i.e. $\mathrm{BP}(f) \geq \rho(f)$. Then, we show a "converse" of this, namely $\mathrm{BP}(f) \leq \rho(f) + 4n$.

Theorem 1 $\mathrm{BP}(f) \geq \rho(f)$.

Proof Consider a branching program for f and the set $\{\langle g_v, x_i^\varepsilon\rangle : v$ has an outgoing edge labeled $x_i^\varepsilon\}$. We will show that this set is a cover of all non-trivial filters defining ones of f.

Suppose that ω is a non-trivial filter defining z such that $f(z) = 1$. Consider an accepting path for z. On this path, if an edge is labeled x_i^ε, then this edge is true and $\omega(x_i^\varepsilon) = 1$, since ω defines z. Since $\omega(g_s) = \omega(U) = 1$ and $\omega(g_t) = \omega(\emptyset) = 0$, there must be a true edge (v, v') labeled x_i^ε such that $\omega(g_v) = 1$ and $\omega(g_{v'}) = 0$. We therefore have that $\omega(x_i^\varepsilon) = 1$ and that $\omega(g_v) \wedge \omega(x_i^\varepsilon) \neq \omega(g_{v'})$. Since $g_{v'} = g_v \cap x_i^\varepsilon$, this implies that $\langle g_v, x_i^\varepsilon\rangle$ covers ω. □

Theorem 2 $\mathrm{BP}(f) \leq \rho(f) + 4n$.

Proof Let $\{\langle g_k, x_{i_k}^{\varepsilon_k}\rangle : 1 \leq k \leq t\}$ be a cover of all non-trivial filters that define ones of f. We will build a branching program that computes f. The nodes are subsets of U; the start node is U, the final node is $\emptyset$. The edges are of four types:

1. (S, T) with no label, for every $S \subseteq T$
2. (U, x_i^ε) with label x_i^ε, for $1 \leq i \leq n$, $\varepsilon = 0, 1$
3. $(x_i^\varepsilon, \emptyset)$ with label $x_i^{1-\varepsilon}$, for $1 \leq i \leq n$, $\varepsilon = 0, 1$
4. $(g_k, g_k \cap x_{i_k}^{\varepsilon_k})$ with label $x_{i_k}^{\varepsilon_k}$, for $1 \leq k \leq t$

Notice that the size of the branching program is the size of the cover plus $4n$.

To prove that the branching program computes f, first suppose that $f(u) = 0$ and assume that there is an accepting path for u in the branching program. Define ω_u by $\omega_u(S) = 1$ if and only if $u \in S$. Clearly, $\omega_u(U) = 1$, $\omega_u(\emptyset) = 0$, and $\omega_u(T) = 1$ whenever $\omega_u(S) = 1$ and $S \subseteq T$. (ω_u is called the principal filter generated by u.) It is also clear that $\omega_u(g) \wedge \omega_u(h) = \omega_u(g \cap h)$ for every g and h.

Now consider an accepting path for u, i.e. a path of true edges from U to $\emptyset$. Since $\omega_u(U) = 1$ and $\omega_u(\emptyset) = 0$, there must be a true edge (S, T) such that $\omega_u(S) = 1$ but $\omega_u(T) = 0$. It is easy to see that such an edge can only be of the form $(g, g \cap x_i^\varepsilon)$ with x_i^ε true. However, x_i^ε is true for u if and only if

$u \in x_i^\varepsilon$. Therefore, $\omega_u(g) \wedge \omega_u(x_i^\varepsilon) = 1$ while $\omega_u(g \cap x_i^\varepsilon) = 0$. This contradicts what we had previously established.

For the second part of the proof, suppose that $f(z) = 1$ and assume that there is no accepting path for z. Let ω be defined by $\omega(S) = 1$ if and only if z can reach S. Notice that ω is a non-trivial filter since $\omega(U) = 1$, $\omega(\emptyset) = 0$, and $\omega(T) = 1$ whenever $\omega(S) = 1$ and $S \subseteq T$. Moreover, the edges of type 2 and 3, together with the fact that there is no accepting path for z, guarantee that x_i^ε is true for z if and only if $\omega(x_i^\varepsilon) = 1$. This means that ω defines z and also implies that $\omega(g) \wedge \omega(x_i^\varepsilon) = \omega(g \cap x_i^\varepsilon)$, for every $\langle g, x_i^\varepsilon \rangle$ in the cover. However, since $f(z) = 1$, this contradicts the fact that ω must be covered by some pair in the cover. □

Note that Theorem 1 leads to a lower bound on the size of nondeterministic branching programs computing the majority function, namely BP(majority) $\in \Omega(n \log\log\log^* n)$ [4].

2 Nondeterministic Circuits

We now turn to nondeterministic circuits. We think of such a circuit as a straight-line program whose inputs are the real inputs $x_1, \ldots, x_n$ together with a set of "guessed" variables $y_1, \ldots, y_m$:

$$(x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n, y_1, \ldots, y_m, \overline{y}_1, \ldots, \overline{y}_m, g_1, \ldots, g_t).$$

The function f computed by such a program P is defined by $f(x) = 0$ if and only if $\exists y$ such that $P(x, y) = 0$.[1] Let $U = f^{-1}(0)$ and consider the following table, where the row for $u \in U$ describes a 0-computation of P on u:

	$x_1 \cdots x_n$	$\overline{x}_1 \cdots \overline{x}_n$	$y_1 \cdots y_m$	$\overline{y}_1 \cdots \overline{y}_m$	$g_1 \cdots g_t$
U					0
					0
					⋮
					0
					0

[1] What we have just defined can be called a *co-nondeterministic* circuit. This will lead to characterizations of coNP. To characterize NP, simply define f in the usual way, let $U = f^{-1}(1)$ in what follows, and carry out the dual argument.

Once more, the idea behind the fusion method is to find ω that defines v and z such that $v \notin U$ but $P(v,z) = 0$. This would contradict the fact that f is computed by P. To this end, we still need that $\omega(\emptyset) = 0$ and that ω be consistent with P. However, the new row produced by ω will give a contradiction only if $\omega(y_j) \neq \omega(\overline{y}_j)$ for $1 \leq j \leq m$. Therefore, and in order that this condition be independent from P, we will also require that $\omega(S) \neq \omega(\overline{S})$ for every $S \subseteq U$, since every S is a possible column for y_j.

We will look at two restricted sets of possible ω's, each of which will lead to a characterization of coNP as those functions which have polynomial size covers:

1. Our possible ω's will be all *ultrafilters*, which are filters (monotone functions) with this additional property that $\omega(S) \neq \omega(\overline{S})$.

2. Our possible ω's will be all *affine* functions, where $\omega(S)$ will be defined as the $\mathbf{Z}_2$ inner product of ω and S viewed as vectors in ${\mathbf{Z}_2}^U$, and furthermore $\omega(\mathbf{1}) = 1$. This version will also have a branching program analogue, characterizing $\oplus$L. These all have $\omega(S) \neq \omega(\overline{S})$.

In both cases, the now-standard technique of constructing a cover from a correct straight-line program will yield the result that $\mathrm{SIZE}_{\mathrm{ND}}(f) \geq \rho_\Omega(f)$, where furthermore $\mathrm{SIZE}_{\mathrm{ND}}(f)$ will be the minimal number of AND-gates in a nondeterministic circuit computing f. (In the affine case we will show explicitely that we need deal only with AND-gates—in the ultrafilter case, the argument used yesterday for deterministic circuits and filters can be easily generalized to nondeterministic circuits.)

2.1 The ultrafilter case

We begin with the proof that in the ultrafilter case (Ω is the set of all ultrafilters), $\mathrm{SIZE}_{\mathrm{ND}}(f) \leq \rho_\Omega(f)^{O(1)}$ [1]. That is, given a cover $\{\langle g_1, h_1\rangle, \ldots, \langle g_t, h_t\rangle\}$, such that for all $\omega \in \Omega$ such that ω defines some $v \notin U$, for some i $\omega(g_i) \wedge \omega(h_i) \neq \omega(g_i \cap h_i)$, we must construct a nondeterministic circuit of size $t^{O(1)}$ computing $f(z)$ on any input $z \in \{0,1\}^n$. We again have:

Lemma 3 *$f(z) = 0$ iff $\exists \omega \in \Omega$ defining z and consistent with the cover.*

Proof As before in the filter case. If $f(z) = 0$, take ω to be the principal ultrafilter generated by z (which sets $\omega(S) = 1$ iff $z \in S$, i.e., the output row

is a copy of the row for $z \in U$). This defines z and is consistent because it is a row of the table. If $f(z) = 1$, any ω defining z must be inconsistent with the cover, by definition of "cover". $\square$

We proceed to use this lemma to compute $f(z)$ on arbitrary input z. The idea will be to try to construct $\omega \in \Omega$ which defines z and is consistent with the cover. If we succeed $f(z) = 0$. If we fail, we want to be sure that $f(z) = 1$. It suffices to know the behaviour of ω on $3t + 2n$ special values: $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n, g_1, h_1, g_1 \cap h_1, \ldots, g_t, h_t, g_t \cap h_t\}$. We will guess a bit for each value in this list. We can then verify that the resulting (partially-defined) ω defines z and is consistent with the cover (i.e., that for all i $\omega(g_i) \wedge \omega(h_i) = \omega(g_i \cap h_i)$). If it is, we still need to verify that this guess can be extended to an ultrafilter.

Lemma 4 *A set $S \subseteq \mathcal{P}(U)$ can be extended to an ultrafilter iff S has the pairwise intersection property, i.e., $\forall A, B \in S$, $A \cap B \neq \emptyset$.*

Proof $\Rightarrow$: By contrapositive. If $\exists A, B$ with $A \cap B = \emptyset$, any filter containing A and B will also contain $\overline{A}$ (as $B \subseteq \overline{A}$) and thus fail to be an ultrafilter.

$\Leftarrow$: Consider the filter obtained from S by closing it upward under containment, i.e., $\{s' : \exists s \in S \; s \subseteq s'\}$. This is a filter which contains S and has the pairwise intersection property. By greedily adding sets to the filter for as long as possible, create a filter ω which is maximal among those which contain S and have the pairwise intersection property. This ω is an ultrafilter, for if any pair $A, \overline{A}$ were both not in ω, we could find a set $B \in \omega$ with $B \cap A = \emptyset$ (because if not we could add A to ω) and a set $B' \in \omega$ with $B' \cap \overline{A} = \emptyset$ (because if not we could add $\overline{A}$ to ω). $B \cap B' = \emptyset$ in this case, violating the pairwise intersection property. $\square$

Now we can conclude $\mathrm{SIZE}_{\mathrm{ND}}(f) \leq \rho_\Omega(f)^{O(1)}$, because the pairwise intersection property for the guessed subset of $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n, g_1, h_1, g_1 \cap h_1, \ldots, g_t, h_t, g_t \cap h_t\}$ can be verified from the disjointness relation among these sets, and this information can be built into the circuit. We can guess a subset extendible to an ultrafilter iff such an ultrafilter exists, and we can verify that the guess is consistent and extendible with $O(t^2)$ gates in a circuit.

2.2 The affine function case

We now turn to our other set Ω, the affine functions $\{\omega \in \{0,1\}^U : \langle \omega, \mathbf{1} \rangle = 1\}$ where the function $\omega(g)$ is given by the $\mathbf{Z}_2$ inner product $\langle \omega, g \rangle$ [2]. We revise the basis of our circuits to be $\{\wedge, \oplus, 1\}$, a complete basis. The function $\mathbf{1}$ (a function from $\{0,1\}^U$ to $\{0,1\}$ which is constant) must also now appear in our straight-line program. Any consistent ω must thus have $\omega(\mathbf{1}) = 1$. This forces the property we wanted, that $\omega(g) \neq \omega(\overline{g})$ for any $g \in \{0,1\}^U$. Also, for free, we get that $\omega(g \oplus h) = \omega(g) \oplus \omega(h)$ because $\langle \omega, g \oplus h \rangle = \langle \omega, g \rangle \oplus \langle \omega, h \rangle$ by the linearity of the inner product. This means that *all* ω we consider are consistent with *all* $\oplus$-instructions in the program, so we may think of the covering as consisting only of $\wedge$-gates. As before, $\rho_\Omega(f)$ for $f : \{0,1\}^n \to \{0,1\}$ is the size of a minimal set of pairs $\langle g, h \rangle$ such that any affine ω defining some $z \notin U = f^{-1}\{0\}$ must have $\omega(g) \wedge \omega(h) \neq \omega(g \wedge h)$ for some pair $\langle g, h \rangle$ in the cover.

Theorem 5 (Lower bound) $\mathrm{SIZE}_{\mathrm{ND}}(f) \geq \rho_\Omega(f)$.

Proof As was done yesterday, we form a cover from the pairs occuring in the $\wedge$-instructions of the straight-line program. □

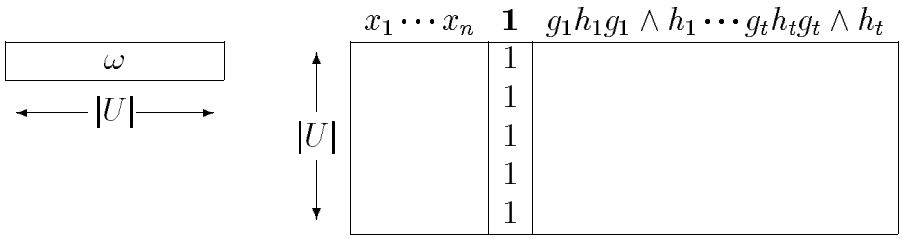
Theorem 6 (Upper bound) $\mathrm{SIZE}_{\mathrm{ND}}(f) \leq \rho_\Omega(f)^{O(1)}$.

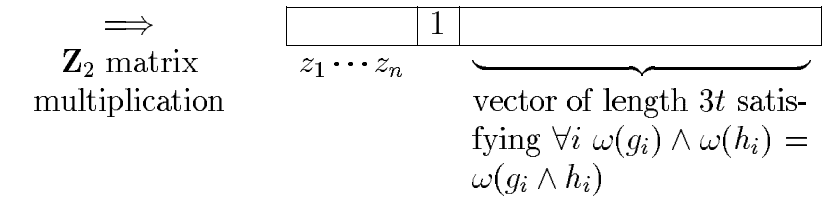
Proof Again we have a cover $\{\langle g_1, h_1 \rangle, \ldots, \langle g_t, h_t \rangle\}$ and we must use it to create a nondeterministic circuit computing $f(z)$ for any input $z \in \{0,1\}^n$. Once again we have:

Lemma 7 $f(z) = 0 \Leftrightarrow \exists \omega \in \Omega$ *such that* ω *defines* z *and is consistent with the cover.* □

Once again we can list the properties we require of such an ω. We must have $\forall i \ \langle \omega, x_i \rangle = z_i$, so that ω defines z. Also, we must have $\forall 1 \leq i \leq t$, $\omega(g_i) \wedge \omega(h_i) = \omega(g_i \wedge h_i)$, so ω is consistent. Essentially, then, we want to multiply the row vector ω with the matrix M for our program, obtain a

length $3t+n+1$ row vector as a result, and test some easy properties of that result:





We would like to just guess ω, do the multiplication, and test the result. Unfortunately, $|U|$ is exponential in n so that even for polynomial t, this could not be done by a polynomial-size circuit. However, any product of any ω and M is a $\mathbf{Z}_2$-linear combination of some subset of the rows of M. Any such combination is also obtainable from any basis of those rows. As M has only $3t+n+1$ columns, we can (non-uniformly) pick a subset of at most $3t+n+1$ of its rows which are a basis. Then it suffices to guess a subset of *these* rows (an ω which is zero except possibly in $3t+n+1$ positions corresponding to these rows), multiply this ω by M (in time polynomial in t) and test the result. If we choose ω so that ωM passes the test, $f(z)=0$. If we cannot possibly choose a length-$(3t+n+1)$ ω, no ω of any size exists and thus $f(z)=1$. □

Clearly the argument extends to using any finite field in place of $\mathbf{Z}_2$. It is less clear that we could use a ring like $\mathbf{Z}_6$.

3 A Combinatorial Problem

Steven Rudich asked for a simple combinatorial problem the solution to which would yield lower bounds on the cover size. Avi provided the following:

An affine ω assigns numbers to subsets of U, by fixing $\omega(g)$ to be $|\omega \cap g| \bmod 2$. Our consistency condition is that if $\langle g, h \rangle$ is a pair, then $|\omega \cap g|$ and $|\omega \cap h|$ are both odd iff $|\omega \cap g \cap h|$ is odd. This is a condition on how the partition of U into $g \cap h$, $g \setminus h$, $h \setminus g$, and $\overline{g \cap h}$ divides the elements of ω into odd or even sets. Avi made the following without proof:

Claim Define a 3-partition to be a division of U into three disjoint sets A, B, and C. If ω is an affine function (an odd-size subset of U), we say $U = A \dot{\cup} B \dot{\cup} C$ *covers* ω if exactly one of the sets $\omega \cap A$, $\omega \cap B$, and $\omega \cap C$ is odd. (The alternative is that all three are odd.) Then $\rho_{\text{AFFINE}}(f)$ is the minimum number of 3-partitions needed to cover all of $\Omega_f = \{\omega : \omega \text{ odd}, \omega \text{ defines some } z \notin U\}$. □

Example Let U be the set of Hamiltonian graphs, so that f is a co-NP complete function. Then Ω_f is the set of odd-size collections ω of Hamiltonian graphs such that the graph obtained by XOR-ing the graphs in the collection (i.e., $G_\omega = \{(u, v) :$ an odd number of graphs in ω contain the edge $(u, v)\}$) is not Hamiltonian. We want to cover this set Ω_f by 3-partitions of the set of Hamiltonian graphs. If we could prove that a superlinear number of 3-partitions were required, we would have a superlinear lower bound on the circuit size of a problem in co-NP. Of course a superpolynomial bound would prove NP $\neq$ co-NP.

It might be easier to work with a more algebraic NP-complete problem. Also, it's annoying that the size of the universe to cover is double exponential —Razborov has done some work in which this universe is much smaller.

4 Span Programs

We briefly review the work of Karchmer and Wigderson [3]. Given a 0–1 matrix M, with columns each labelled by a literal, we define a function f as follows. On input $\sigma \in \{0,1\}^n$, let M_σ be the submatrix of M consisting only of those columns whose literals agree with σ. Then $f(\sigma) = 1$ iff the columns of M_σ span the column vector $\mathbf{1}$. M is a *span program* for f, and $\text{SIZE}_{\text{SP}}(f)$ is the minimum number of columns in a correct span program for f (the number of rows is arbitrary.)

Span programs are essentially what emerge if we combine the branching program framework of the first half of this lecture with the restriction to affine functions ω. By methods similar to the previous cases, we can show

$$\begin{aligned} \mathrm{SIZE}_{\oplus\mathrm{BP}}(f) &\geq \mathrm{SIZE}_{\mathrm{SP}}(f) \\ \mathrm{SIZE}_{\oplus\mathrm{BP}}(f) &\leq \mathrm{SIZE}_{\mathrm{SP}}(f)^{O(1)} \end{aligned}$$

so that the functions with poly-size span programs are exactly those with poly-size parity branching programs, or $\oplus$L.

One can obtain superlinear lower bounds on $\mathrm{SIZE}_{\mathrm{SP}}(f)$ for f the majority function.

References

[1] Karchmer, M.: On proving lower bounds for circuit size. In *Proc. of the 8th Annual Structure in Complexity Theory Conference*, 1993, 112–118

[2] Karchmer, M., Wigderson, A.: Characterizing non-deterministic circuit size. In *Proc. of the 25th Annual ACM Symp. on the Theory of Computing*, 1993, 532–540

[3] Karchmer, M., Wigderson, A.: On span programs. In *Proc. of the 8th Annual Structure in Complexity Theory Conference*, 1993, 102–111

[4] Razborov, A.: Lower bounds on the size of switching-and-rectifier networks for symmetric Boolean functions. (In Russian) *Mathematical Notes of the Academy of Sciences of the USSR* **48:6** (1990) 79–91

Expanders and Pairwise Independence

Notes by Ricard Gavaldà and Pierre McKenzie

Monday Morning, January 31st

In the remaining talks we will review the basic properties of two important tools: expanders and universal hashing. These tools should belong to the fundamental bag of tricks of every computer scientist. Avi likes to introduce the techniques from the viewpoint of a single problem, that of

Deterministic Amplification

Suppose we have a randomized algorithm A that takes two inputs, namely some string x and a string of "random" bits $r \in \{0,1\}^n$, and outputs a bit $A(x,r)$ indicating acceptance or rejection of x.

For every x, we define the set of witnesses for x

$$W_x = \{\ r \in \{0,1\}^n \mid A(x,r) = 1\ \}$$

and

$$\mu(W_x) = \frac{\|W_x\|}{2^n}$$

where $\|\cdot\|$ denotes cardinality (we will use μ similarly to denote the size of a set with respect to the universe in which it naturally fits; we will specify this universe when it is not clear from the context).

The classes RP and BPP can be defined by placing easy conditions on $\mu(W_x)$, where $\alpha > 0$ is any constant:

	RP	BPP
$x \in L$	$\geq 1/2$	$\geq 1/2 + \alpha$
$x \notin L$	0	$\leq 1/2 - \alpha$

The problem of **deterministic amplification** is: given A, build an equivalent algorithm with smaller error probability.

Straightforward Solution:

Draw $Y = \{ y_1, y_2, \ldots, y_k \} \subseteq \{0,1\}^n$ independently.
Test whether each $y_i \in W_x$.

For RP: accept iff $Y \cap W_x \neq \emptyset$. This gives Prob $[error] \leq 2^{-k}$.

For BPP: accept iff $||Y \cap W_x|| \geq \frac{1}{2} \cdot ||Y||$. In order to bound the error probability, define the discrepancy of sample Y as

$$disc(Y) = ||\mu(W_x) - \mu(Y \cap W_x)||,$$

i.e., the deviation from what we would expect if Y was a "perfect sample". Note here that $\mu(W_x) = ||W_x||/2^n$ whereas $\mu(Y \cap W_x) = ||Y \cap W_x||/||Y||$. Using the Chernoff bounds[1], we obtain

$$\text{Prob } [error] = \text{Prob } [disc(Y) \geq \alpha] \leq 2^{-\alpha^2 k}.$$

So the straightforward solution reduces the error probability exponentially at the cost of using $k \cdot n$ random bits. We will present two methods that achieve quite good error rates but use *much less randomness*: The Karp-Pippenger-Sipser scheme [5] based on expanders, and the Chor-Goldreich scheme [2] based on universal hashing. These schemes will achieve the following:

[1] Let $Z_1, Z_2, \ldots, Z_k$ be random Bernoulli variables satisfying Prob $[Z_i = 1] = \frac{1}{2}$. Writing $Z = \Sigma_i Z_i$, the Chernoff bounds state that

$$\text{Prob } [|Z - E(Z)| \geq \epsilon k] \leq 2^{\epsilon^2 k/3}.$$

In our application, $Z_i = 1$ iff $y_i \in W_x$, $\mu(W_x) = \frac{E(Z)}{k}$, and $\mu(Y \cap W_x) = \frac{Z}{k}$.

	# random bits	Error Prob.
Independent choice	$k \cdot n$	2^{-k}
with $k = n$	n^2	2^{-n}
[KPS 85]	n	n^{-c}
[CG 89]	$2n$	n^{-c}

Note that the Karp-Pippenger-Sipser scheme will have *no loss at all* in randomness. Note also that to reduce the error probability to $2^{-\alpha^2 k} \in O(n^{-c})$ by the straightforward independent choice solution, we need $k \in O(\log n)$, so a total of $O(n \cdot \log n)$ random bits (compared with $O(n)$ in both [KPS] and [CG]).

Later on, methods to achieve even better trade-offs have been developed, based either on hashing or on expanders. We will talk about them in subsequent lectures.

The Karp-Pippenger-Sipser Scheme: Expanders

Informal presentation for RP

Let $V = \{0,1\}^n$ and $N = ||V||$. We impose on V a k-regular graph called an *expander*, which is informally defined by the property:

the set of neighbors of every "sufficiently large"
set of vertices has size $> ||V||/2$.

The new algorithm acts as follows:

pick a random point $y \in V$;
compute $\Gamma(y) =$ set of neighbors of y;
accept iff $\Gamma(y) \cap W_x \neq \emptyset$;

To see why this works, note that we make no errors on $x \notin L$. For $x \in L$, define the "bad set"

$$B_x = \left\{ y \mid \Gamma(y) \cap W_x = \emptyset \right\}$$

and note that Prob $[error] = \mu(B_x) = \|B_x\|/\|V\|$.

Now, $\mu(B_x)$ has to be "small". Otherwise, if $\mu(B_x)$ is "sufficiently large" as in the definition of expander, then

$$\mu(\Gamma(B_x)) = \mu\Big(\bigcup_{y \in B_x} \Gamma(y) \Big) > 1/2.$$

But for $x \in L$, $\mu(W_x) \geq 1/2$, so $\Gamma(B_x) \cap W_x \neq \emptyset$. This contradicts the definition of B_x.

Formal development

Let M be a $N \times N$ symmetric matrix. Then there exist

* *eigenvectors* $v_1, v_2, \ldots, v_N \in \mathbb{R}^N$, and
* *eigenvalues* $\lambda_1, \lambda_2, \ldots, \lambda_N \in \mathbb{R}$,

such that for every i,

1. $\|v_i\|_2 = 1$,
2. $v_i \perp v_j$ for every $j \neq i$ (i.e., the v_i's form an orthonormal basis), and
3. $M \cdot v_i = \lambda_i \cdot v_i$
   (i.e., v_i does not change direction by transformation M).

For convenience, we arrange these values in order so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$.

Let $G = (V, E)$ be a d-regular graph, and M be the adjacency matrix of G. We write $\overline{1}$ for the all-ones vector. Because each row of M contains exactly d ones, it is easy to verify that

$$\lambda_1 = d \quad \text{and} \quad v_1 = \frac{\overline{1}}{\sqrt{N}}$$

namely, that $M \cdot v_1 = \lambda_1 \cdot v_1$ and that, for every v and λ, if $M \cdot v = \lambda \cdot v$ then $\lambda \leq d$.

So, for all other eigenvectors v_i, $v_i \perp \overline{1}$.

Define

$$\lambda = \max_{2 \le i \le N} |\lambda_i|$$

which is either $|\lambda_2|$ or $|\lambda_N|$ by our ordering of the λ_i.

For $A, B \subseteq V$, define

$$E(A,B) = \text{ \# edges from } A \text{ to } B$$

(more precisely, the number of 1's in the submatrix of M obtained by taking rows in A and columns in B).

If G, A, and B were chosen at random, we would expect each vertex in A to be connected to $d/N \cdot \|B\|$ vertices in B, so $E(A,B)$ would be $\|A\| \cdot \|B\| \cdot d/N$. The following fundamental lemma shows that the deviation from this value depends strongly on the second largest eigenvalue.

Lemma. For every $A, B \subseteq V$,

$$\left| E(A,B) - \frac{\|A\| \cdot \|B\| \cdot d}{N} \right| \le \lambda \cdot \sqrt{\|A\| \cdot \|B\|}.$$

Proof. Let χ_A and χ_B be the characteristic vectors of A and B. Note that

$$E(A,B) = \chi_A \, M \, \chi_B = \sum_{i,j=1}^{N} (\chi_A)_i \, M_{i,j} \, (\chi_B)_j$$

and now decompose χ_A and χ_B with basis the set of eigenvectors: For some α_i's and β_i's we have

$$\chi_A = \sum_{i=1}^{N} \alpha_i \, v_i \quad \text{and} \quad \chi_B = \sum_{i=1}^{N} \beta_i \, v_i.$$

Then

$$\begin{aligned}
E(A,B) &= \chi_A \, M \, \chi_B = \left(\sum_{i=1}^{N} \alpha_i \, v_i \right) M \left(\sum_{i=1}^{N} \beta_i \, v_i \right) = \\
&= \left(\sum_{i=1}^{N} \alpha_i \, v_i \right) \left(\sum_{i=1}^{N} \beta_i \, M \, v_i \right) = \qquad (v_i\text{'s are eigenvectors}) \\
&= \left(\sum_{i=1}^{N} \alpha_i \, v_i \right) \left(\sum_{i=1}^{N} \beta_i \, \lambda_i \, v_i \right) = \qquad (v_i\text{'s are orthonormal}) \\
&= \sum_{i=1}^{N} \lambda_i \, \alpha_i \, \beta_i.
\end{aligned}$$

To bound this sum, compute

$$\alpha_1 = \chi_A \cdot v_1 = \frac{\|A\|}{\sqrt{N}}, \qquad \beta_1 = \frac{\|B\|}{\sqrt{N}}, \qquad \lambda_1 \alpha_1 \beta_1 = \frac{\|A\| \cdot \|B\| \cdot d}{N} \ .$$

Then

$$\sum_{i=1}^{N} \lambda_i \alpha_i \beta_i = \frac{\|A\| \cdot \|B\| \cdot d}{N} + \sum_{i=2}^{N} \lambda_i \alpha_i \beta_i \ .$$

So

$$\begin{aligned} \left| E(A,B) - \frac{\|A\| \cdot \|B\| \cdot d}{N} \right| &= \sum_{i=2}^{N} \lambda_i \alpha_i \beta_i \leq \lambda \left| \sum_{i=2}^{N} \alpha_i \beta_i \right| \\ &\leq \lambda \sqrt{\left(\sum_{i=2}^{N} \alpha_i^2 \right) \cdot \left(\sum_{i=2}^{N} \beta_i^2 \right)} \end{aligned}$$

by the Cauchy-Schwartz inequality and, observing that

$$\sum_{i=1}^{N} \alpha_i^2 = (\|\chi_A\|_2)^2 = \|A\|, \qquad \sum_{i=1}^{N} \beta_i^2 = (\|\chi_B\|_2)^2 = \|B\|,$$

we bound this quantity by $\lambda \cdot \sqrt{\|A\| \cdot \|B\|}$. □

In fact, we want to use the following, *weaker*, version of the lemma.

Lemma. For every $A, B \subseteq V$,

$$|\mu(E(A,B)) - \mu(A)\mu(B)| \leq \frac{\lambda}{d}$$

(where the first μ is taken over all edges, and the other μ's over the set V).

Proof. By the definition of the μ's,

$$\begin{aligned} |\mu(E(A,B)) - \mu(A)\mu(B)| &= \left| \frac{E(A,B)}{dN} - \frac{\|A\| \cdot \|B\|}{N^2} \right| = \\ &= \frac{1}{dN} \left| E(A,B) - \frac{\|A\| \cdot \|B\| \cdot d}{N} \right| \leq \frac{\lambda}{dN} \sqrt{\|A\| \, \|B\|} \leq \frac{\lambda}{d} \ . \end{aligned}$$

(The first inequality holds by the previous lemma, and the last one because $\|A\|, \|B\| \leq N$.) $\square$

Definition. Avi defines an expander to be a graph for which $\frac{\lambda}{d} < 1$.

Note. There are other definitions of expanders, concerning their expansion ratio. Our definition implies the others by arguments like the following. Alon has shown that the converse implication also holds (but the proof is harder).

Now we formalize the argument for RP and BPP. We assume that there are expanders with conveniently small ratio λ/d, and, furthermore, that given a vertex of the expander we can list its neighborhood efficiently. (This is a claim to be proved later).

For RP:

We can make mistakes on $x \in L$, but only when the y picked is in the "bad" set

$$B_x = \left\{ y \mid \Gamma(y) \cap W_x = \emptyset \right\}.$$

Clearly, $E(W_x, B_x) = 0$, so using the lemma we have

$$\begin{aligned} \frac{\lambda}{d} &\geq |\mu(E(W_x, B_x)) - \mu(W_x)\,\mu(B_x)| = \\ &= \mu(W_x)\,\mu(B_x) \geq \frac{1}{2}\,\mu(B_x) \end{aligned}$$

(where the last inequality holds because $x \in L$). Taking an expander for which $\frac{\lambda}{d} \leq \frac{1}{2n^c}$,

$$\text{Prob}\,[error] = \mu(B_x) \leq \frac{2\lambda}{d} \leq n^{-c}, \qquad \text{as promised.}$$

For BPP:

Assume $x \notin L$ (the case $x \in L$ is symmetric). Then

$$\mu(W_x) \leq \frac{1}{2} - \alpha, \qquad B_x = \left\{ y \mid \|\Gamma(y) \cap W_x\| \geq \frac{1}{2}\,d \right\}$$

so

$$\mu(E(W_x, B_x)) = \frac{E(W_x, B_x)}{dN} \geq \frac{\frac{1}{2}\, d\, \|B_x\|}{dN} = \frac{1}{2}\, \mu(B_x) \ .$$

Therefore,

$$\begin{aligned} \frac{\lambda}{d} &\geq |\, \mu(E(W_x, B_x)) - \mu(W_x)\, \mu(B_x)\, | \geq \\ &\geq \frac{1}{2}\, \mu(B_x) - \left(\frac{1}{2} - \alpha\right) \mu(B_x) = \alpha\, \mu(B_x) \ . \end{aligned}$$

And, taking an expander for which $\frac{\lambda}{d} \leq \frac{\alpha}{n^c}$,

$$\text{Prob}\,[error] = \mu(B_x) \leq \frac{\lambda}{d\alpha} \leq n^{-c}, \qquad \text{as promised.}$$

Note. To further improve the error probability, it suffices to choose expanders for which the ratio λ/d is smaller than $n^{-O(1)}$. Unfortunately, with the construction of expanders given below, each vertex gets a superpolynomial number of neighbors, hence the running time becomes superpolynomial too.

Existence of Expanders

Theorem [6] (stated without proof). For each function $d' : \mathbb{N} \to \mathbb{N}$, for each $N' \in \mathbb{N}$, there exists a $d(N) = \theta(d'(N))$-regular graph G on $N = \theta(N')$ vertices such that for each vertex v and for each i, $1 \leq i \leq d(N)$, it is possible to compute the i^{th} neighbor of v in space $\log(\log(\#\text{ of vertices in } G))$, and $\lambda \leq [d(N)]^{1-\beta}$ for some $\beta > 0$, where λ is the second largest eigenvalue of G in absolute value. □

The Ramanujan graph family [6] is an expander graph family with $\beta = \frac{1}{2}$. In many applications of the above theorem, $N = 2^n$ where n is the input size of the problem under consideration. In the deterministic amplification application to RP presented above, we let $d = k = 4n^{2c} = 4(\lg N)^{2c}$, which yields $\lambda \leq 2n^c$ and thus $\frac{\lambda}{d} \leq \frac{1}{2n^c}$ in the case of Ramanujan graphs. Writing $\epsilon = \frac{1}{n^c}$, the number k of sample points is therefore proportional to $1/\epsilon^2$. Appreciate that we have managed to get by with only the n random bits

necessary to specify one of the 2^n nodes. (For the application to BPP, we would pick $d = k = \frac{n^{2c}}{\alpha^2}$, so that $\lambda \leq \frac{n}{\alpha}$ and $\frac{\lambda}{d} \leq \frac{n}{\alpha} \cdot \frac{\alpha^2}{n^{2c}} = \frac{\alpha}{n^c}$.)

We mention incidentally the Gaber-Galil graphs [3] (see also Margulis [7] who proved $\lambda < d$ for those graphs, and Jimbo and Maruoka [4]; a good survey on related matters is authored by Buck [1]). These graphs are 8-regular and are defined on $V = Z_m \times Z_m$ as follows: for each $x \in Z_m$ and each $y \in Z_m$, an edge exists between (x, y) and each of the four vertices $(x, x+y)$, $(x, x+y+1)$, $(x+y, y)$, $(x+y+1, y)$. It can be shown that the second largest eigenvalue λ is at most 7.9. Note that degrees larger than 8 can be obtained by taking powers of the graph G so defined. Indeed G^t has degree 8^t and its second largest eigenvalue is at most 7.9^t.

Expanders and Random Walks

The next lemma is a revealing aspect of expanders. Let us perform a random walk on the graph. If the expander is d–regular and M is its adjacency matrix, let

$$T = M/d$$

be the transition matrix of the walk. At time t, $P_t \in \mathbb{R}^N$ contains the probabilities of being at node i, $1 \leq i \leq N =$ the number of vertices in the graph. Then

$$P_{t+1} = TP_t.$$

Now T is "doubly stochastic", i.e. the sum of its entries along each line and the sum along each column are 1. Hence

$$T\bar{1} = \bar{1}$$

and furthermore T preserves the L_1-norm of any vector (i.e. the sum of the absolute values of its coefficients). Because the original graph is an expander, 1 is the largest eigenvalue of T, and all other eigenvalues are at most $\lambda/d < 1$ in absolute value, where λ is the second largest eigenvalue of M. Now we do not worry about technicalities and assume that the random walk described by T converges to a unique probability distribution. The point we wish to make in the next lemma is that the random walk quickly converges to the *uniform* distribution.

Let $P_0 \in \mathbb{R}^N$ be any vector with L_1-norm $\|P_0\|_1 = 1$. We can write

$$P_0 = v + u$$

where $v \parallel \bar{1}$ and $u \perp \bar{1}$. It follows from $u \perp \bar{1}$ that $\|u\|_1 = 0$. Thus $\|v\|_1 = 1$, so that $v = \frac{\bar{1}}{N}$. Now consider

$$P_t = T^t P_0.$$

We have

$$P_t = T^t(v+u) = T^t v + T^t u = v + T^t u$$

because $T\bar{1} = \bar{1}$. Now u can be written as a linear combination of all eigenvectors of T^t except $\bar{1}$. Clearly the second largest eigenvalue of T^t is at most $(\lambda/d)^t$. Hence $\|T^t u\|_2 \leq (\lambda/d)^t$ and by the Cauchy-Schwartz inequality, $| T^t u |_1 \leq \sqrt{N}(\lambda/d)^t$. This proves the

Lemma. $\||P_t - \frac{\bar{1}}{N}\|_2 \leq (\lambda/d)^t$ and $\||P_t - \frac{\bar{1}}{N}\|_1 \leq \sqrt{N}(\lambda/d)^t$. □

In other words, regardless of the initial probability vector P_0, the probability that we find ourselves at a given node after t steps of the random walk is close to $1/N$. This is because of the expander property, namely, $(\lambda/d) < 1$.

The Chor-Goldreich Scheme: Pairwise Independence

We now return to deterministic amplification. We will show that $2n$ random bits suffice to obtain an error probability of $1/n^c$, without the need for expanders. We will make use of pairwise independence instead. This is a much simpler tool, due to Chor and Goldreich [CG 89].

Fix any finite field V. Think of V as $\{0,1\}^n$. Now pick a and b at random from V (this requires the $2n$ random bits). Define the set of sample points

$$Y = (\{y_1, y_2, \ldots, y_k\}$$

with $y_i = a + bi$, $1 \leq i \leq k$. Note here that i is an element of V and thus bi involves a product in V. An important observation which we will use below is that

$$(\forall i, j)\ [y_i \text{ and } y_j \text{ are independent}].$$

This is because

$$\begin{pmatrix} y_i \\ y_j \end{pmatrix} = \begin{pmatrix} 1 & i \\ 1 & j \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

and the square matrix is invertible (of course $i \neq j$ in this statement).

For BPP:

As before, we use the y_i as sample points and accept the input x iff the majority of trials accept. Again we write W_x for the set of witnesses of the fact that $x \in L$. Now define, for $1 \leq i \leq k$,

$$z_i = \begin{cases} 0 & \text{if } y_i \notin W_x \\ 1 & \text{if } y_i \in W_x. \end{cases}$$

Note that

$$E[z_i] = \mu(W_x).$$

Define

$$Z = \sum_{i=1}^{k} z_i.$$

Hence Z is the number of witnesses encountered after the k sample points, i.e. $Z = \|Y \cap W_x\|$. Now suppose that x does not belong to L. Then if we make an error, it is because

$$\frac{\|Y \cap W_x\|}{k} \geq 1/2$$

which implies

$$\frac{\|Y \cap W_x\|}{k} - \mu(W_x) \geq 1/2 - \mu(W_x) \geq \alpha$$

which implies

$$\|Y \cap W_x\| - E[Z] \geq \alpha k.$$

An analogous computation in the case in which x does belong to L shows that an error by the "majority" algorithm implies

$$E[Z] - \|Y \cap W_x\| \geq \alpha k.$$

Hence we take care of both $x \in L$ and $x \notin L$ at once as follows.

$$
\begin{aligned}
\text{Prob}\,[\,|\,Z - E[Z]\,|\,\geq \alpha k] &\leq \frac{\text{Var}[Z]}{\alpha^2 k^2} && \text{(Chebychev)} \\
&= \frac{\sum_{i=1}^{k} \text{Var}[z_i]}{\alpha^2 k^2} && (y_i \text{ and } y_j \text{ independent}) \\
&= \frac{k\ \text{Var}[z_1]}{\alpha^2 k^2} \\
&= \frac{k[\mu(W)\ (1-\mu(W))]}{\alpha^2 k^2} && \text{(Variance(Bernoulli))} \\
&\leq \frac{1}{k\alpha^2}.
\end{aligned}
$$

Note that the above is useful when α is constant or $(\text{poly}(n))^{-1}$. Then we obtain Prob $[error] \leq n^{-c}$ by setting $k = n^c/\alpha^2$. In terms of $\epsilon = n^{-c}$, the number of sample points for the Chor-Goldreich method is thus $1/(\epsilon\alpha^2)$. This compares with $k = 1/(\epsilon^2\alpha)$ in the KPS method described earlier.

Remark also that we only have pairwise independence, so we can only apply Chebychev's inequality and get $1/k\alpha^2$. If we had full independence, we could use Chernoff's bounds instead and get $2^{-\Omega(\alpha^2 k)}$.

References

[1] M.W. Buck, Expanders and Diffusers, *SIAM J. Alg. Disc. Meth.* **7**, 2 (1986).

[2] B. Chor and O. Goldreich, On the power of two-point based sampling, *Journal of Complexity,* Vol. 5, 1989, pp. 96-106.

[3] O. Gaber and Z. Galil, Explicit Constructions of Linear Size Superconcentrators, *Journal of Computer and Systems Science*, **22** (1981), pp. 407-420.

[4] S. Jimbo and A. Maruoka, Expanders Obtained from Affine Transformations, *Combinatorica* **7** (1987), pp. 343-355.

[5] R.M. Karp, N. Pippenger and M. Sipser, A Time-Randomness Tradeoff, *AMS Conference on Probabilistic Computational Complexity*, Durham, New Hampshire (1985).

[6] A. Lubotzky, R. Phillips and P. Sarnak, Explicit Expanders and the Ramanujan Conjectures, *Proc. 18th STOC*, 1986, pp. 240-246.

[7] G.A. Margulis, Explicit Construction of Concentrators, *Prob. Per. Infor.* 9 (4) (1973), 71–80. (English translation in *Problems of Infor. Trans.* (1975), 325–332.)

Improving Randomized Algorithms via Universal Hashing

Notes by Denis Thérien and Richard Beigel

Tuesday Morning, February 1st

1 Introduction

Recall the situation being considered: a probabilistic algorithm A decides if an input x is in L by generating n random bits y and accepting x iff $A(x,y) = 1$. Let $W_x = \{\ y \in \{0,1\}^n \mid A(x,y) = 1\ \}$. The class RP is defined as those languages for which an A exists such that

$$x \in L \Rightarrow ||W_x|| \geq 2^{n-1}$$

and

$$x \notin L \Rightarrow ||W_x|| = 0.$$

The class BPP is defined by the condition that there is some constant $0 < \alpha \leq 1/2$ such that

$$x \in L \Rightarrow ||W_x|| \geq (1/2 + \alpha)2^n$$

and

$$x \notin L \Rightarrow ||W_x|| \leq (1/2 - \alpha)2^n.$$

(We will sometimes use the notation $\mu(W_x)$ to stand for $\frac{||W_x||}{2^n}$.)

The goal is to decrease the probability of error while using as few random bits as possible. We will describe a scheme due to Impagliazzo and Zuckerman [2] which allows the error probability to be as small as $2^{-\Omega(\sqrt{n})}$ while using less than $5n$ random bits. Note that to obtain the same accuracy by running the algorithm $\sqrt{n}$ times would "cost" $n\sqrt{n}$ random bits. The idea of this method is that, starting with a random string $y = y_1$, it is possible to use a small number of new random bits to generate, via hashing, strings $y_2, \ldots, y_{\sqrt{n}}$ which will appear random to A, i.e. the outputs $A(x,y_1), \ldots, A(x,y_{\sqrt{n}})$ will look almost the same as if the y_i's had been generated truly at random.

2 Pairwise independence

We first impose a field structure on our universe $\{0,1\}^n$: this can be done by identifying a binary string $b_0 \ldots b_{n-1}$ with the polynomial $b_0 + b_1 x + \ldots + b_{n-1}x^{n-1} \in Z_2[x]$ and performing the arithmetic modulo some fixed irreducible polynomial of degree n. An alternative approach would be to embed $\{0,1\}^n$ into Z_p where p is the smallest prime greater than 2^n: this would have the disadvantage of complicating somewhat the upcoming discussion by introducing a small distorting factor in the notion of pairwise independence.

We consider the class H of functions from $\{0,1\}^n$ to $\{0,1\}^n$ which have the form $h(x) = a + bx$ for some a, b in $\{0,1\}^n$. Note that any h in H is specified by the $2n$ bits defining a and b.

Lemma 2.1 *(statement of pairwise independence) For all* $x \neq y, \alpha, \beta \in \{0,1\}^n$, *we have* Prob $[h(x) = \alpha \bigwedge h(y) = \beta] = \frac{1}{2^{2n}}$.

Proof. We can write

$$\begin{pmatrix} h(x) \\ h(y) \end{pmatrix} = \begin{pmatrix} 1 & x \\ 1 & y \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

Since $x \neq y$ the inverse of the matrix exists: there is thus a 1-1 correspondence between the pairs (a, b) and $(h(x), h(y))$. Hence Prob $[h(x) = \alpha \bigwedge h(y) = \beta]$ is equal to the probability of picking the unique (a, b) corresponding to (α, β), which is $1/2^{2n}$. □

Note that we get nothing more than pairwise independence, since any two values $h(x), h(y)$ determine h completely.

More generally, we will consider $H = \{\ h : \{0,1\}^n \to \{0,1\}^m \mid h(x) = a + bx \bmod 2^m\ \}$, i.e. for fixed $a, b \in \{0,1\}^n$, the value $h(x)$ is obtained by taking the m low-order bits of $a+bx$. For any $x \neq y \in \{0,1\}^n, \alpha, \beta \in \{0,1\}^m$, we clearly have as above that Prob $[h(x) = \alpha \bigwedge h(y) = \beta] = 1/2^{2m}$.

3 The leftover hash lemma

Let U be the uniform distribution on a (finite) set S, i.e. any element of S can be selected with probability $1/\|S\|$. Let P, Q be distributions on S: the distance between P and Q is defined as $d(P,Q) = \max_{A \subseteq S} |\text{Prob}_P[A] - \text{Prob}_Q[A]|$.

Lemma 3.1 *Let p_i, q_i denote the probability of selecting element $i \in S$ according to distribution P, Q respectively. Then $d(P,Q) = \frac{1}{2}\sum_{i \in S} |p_i - q_i|$*

Proof. The value of $d(P,Q)$ is obtained by selecting $A = \{\ i \mid p_i > q_i\ \}$ or $A = \{\ i \mid p_i < q_i\ \}$. Since $\sum_{i \in S} |p_i - q_i| = \sum_{p_i > q_i} |p_i - q_i| + \sum_{p_i < q_i} |p_i - q_i|$ and the two terms in the right-hand sum are equal, the result follows. □

Definition 3.2 *A distribution P on S is ϵ-quasi-random if $d(P,U) \leq \epsilon$ (i.e. the probability of an event A occurring is within ϵ of $||A||/||S||$).*

The idea of the leftover hash lemma is originally due to Impagliazzo, Levin and Luby [1]. The stronger version given here is credited to Rackoff in [2].

Lemma 3.3 *(Leftover Hash Lemma) Let $W \subseteq \{0,1\}^n, l \geq 0$; if $||W|| \geq 2^{m+2l}$ the distribution of $\langle h, h(x)\rangle$ in $H \times \{0,1\}^m$ when $h \in_R H, x \in_R W$ is 2^{-l}-quasi-random.*

Thus the lemma intuitively says the following: we use our special functions from H to hash a subset W of $\{0,1\}^n$ in a table 2^{2l} times smaller. For any W (even if not random at all!), the distribution in the table of the hashed values will be close to uniform, for most hash functions h.

It is quite surprising that a property as weak as pairwise independence gives such a strong result (close to the uniform distribution). Even a truly random choice of h among all functions from $\{0,1\}^n$ to $\{0,1\}^m$ would not be expected to do much better.

The proof of the leftover hash lemma will make use of the following

Definition 3.4 *The collision probability $C(P)$ of a distribution P is the probability of getting the same element twice in two independent draws from the set S. Thus $C(P) = \sum_{i \in S} p_i^2$. Note that $C(U) = \sum_{i \in S} \frac{1}{||S||^2} = \frac{1}{||S||}$.*

Lemma 3.5 *If $C(P) \leq \frac{1}{||S||}(1 + 2\epsilon^2)$ then P is ϵ-quasi- random.*

Proof.

$$\begin{aligned}
\sum_{i\in S}|p_i - \frac{1}{||S||}| &\leq \sqrt{||S||}\sqrt{\sum_{i\in S}(p_i - \frac{1}{||S||})^2} \text{ (by Cauchy-Schwartz)} \\
&= \sqrt{||S||}\sqrt{\sum_{i\in S} p_i^2 + \sum_{i\in S}\frac{1}{||S||^2} - \frac{2}{||S||}\sum_{i\in S} p_i} \\
&\leq \sqrt{||S||}\sqrt{(1+2\epsilon^2)\frac{1}{||S||} + \frac{1}{||S||} - \frac{2}{||S||}} \\
&\quad \text{(using the hypothesis and } \sum_{i\in S} p_i = 1) \\
&= \sqrt{||S||}\sqrt{2\epsilon^2\frac{1}{||S||}} \\
&= \sqrt{2}\epsilon \\
&\leq 2\epsilon.
\end{aligned}$$

Hence the distance between P and the uniform distribution is at most ϵ. □

Proof. (leftover hash lemma) We prove that the distribution of $\langle h, h(x)\rangle$ is 2^{-l}-quasi- random by estimating the collision probability.

$$\begin{aligned}
\text{Prob } [\langle h, h(x)\rangle = \langle h', h'(x')\rangle] &= \text{Prob } [h = h'] \cdot \text{Prob } [h(x) = h'(x') | h = h'] \\
&= \text{Prob } [h = h'] \cdot \text{Prob } [h(x) = h(x')] \\
&\leq \frac{1}{||H||}(\text{Prob } [x = x'] + \\
&\quad \text{Prob } [h(x) = h(x') | x \neq x']) \\
&= \frac{1}{||H||}(\frac{1}{||W||} + \frac{1}{2^m}) \\
&\quad \text{because of pairwise independence} \\
&\leq \frac{1}{||H||2^m}(2^{-2l} + 1) \\
&\leq \frac{1}{||H||2^m}(2 \cdot 2^{-2l} + 1)
\end{aligned}$$

By the above lemma, the distribution is 2^{-l}-quasi-random. □

Corollary 3.6 *For all but a 2^{-l} fraction of functions $h \in H$, the distribution on $h(x)$ obtained by picking x uniformly at random from W is 2^{-l}-quasi-random.*

Proof. Intuitively, if too many h are too far from being uniform, the distribution of $\langle h, h(x)\rangle$ cannot be close to uniform. □

4 The Impagliazzo-Zuckerman method

A random algorithm A, given input x, randomly selects n bits y and accepts x iff $A(x,y) = 1$. In other words, the only information extracted from y is the one bit describing if $y \in W_x = \{y \mid A(x,y) = 1\}$. To increase the probability of the algorithm correctly deciding the answer for x, we wish to execute it with several random values of y. The method we are about to describe shows that one can reuse random bits to construct strings that will "appear" to the algorithm as if they had been freshly randomly generated.

The idea is as follows: we will construct $\sqrt{n}$ different vectors $y_1, \ldots, y_{\sqrt{n}} \in \{0,1\}^n$ using only $O(n)$ random bits. We then run the algorithm $\sqrt{n}$ times to compute $A(x,y_1), \ldots, A(x,y_{\sqrt{n}})$. In the RP case, we accept x if at least one $A(x,y_i)$ is equal to 1; in the BPP case, we accept x if the majority of the $A(x,y_i)$'s are 1. More precisely, let $h \in_R H = \{h : \{0,1\}^n \to \{0,1\}^{n-2\sqrt{n}} \mid h(x) = a + bx \bmod 2^{n-2\sqrt{n}}\}, y \in_R \{0,1\}^n$ and $r_2, \ldots, r_{\sqrt{n}} \in_R \{0,1\}^{2\sqrt{n}}$; thus $5n - 2\sqrt{n}$ random bits suffice to specify these parameters. We construct $y_1, \ldots, y_{\sqrt{n}}$ by setting $y_1 = y$ and for $i = 2, \ldots, \sqrt{n}$, $y_i = h(y_{i-1})r_i$ (i.e. we "hash down" the n bits of y_{i-1} to $n-2\sqrt{n}$ bits and pad with the fresh random bits of r_i.

To complete the analysis of the error probability for this scheme, we wish to show that the execution of the algorithm with the constructed y_i's behaves essentially as if these vectors had been chosen truly at random.

Fix the input x and let $W = W_x = \{y \mid A(x,y) = 1\}$. We define the random variables

$$z_i = \begin{cases} 0 & \text{if } y_i \notin W \\ 1 & \text{if } y_i \in W \end{cases}$$

Let also

$$b_i = \begin{cases} 0 & \text{with probability } 1 - \mu(W) \\ 1 & \text{with probability } \mu(W) \end{cases}$$

It will be enough to show that the distance between the distribution on $\{0,1\}^{\sqrt{n}}$ induced by the z_i's and the distribution induced by the b_i's is small.

Theorem 4.1 *If* $d(\langle z_1,\ldots,z_{\sqrt{n}}\rangle,\langle b_1,\ldots,b_{\sqrt{n}}\rangle) \leq 2^{-\Omega(\sqrt{n})}$, *then the Impagliazzo-Zuckerman method works with error probability* $2^{-\Omega(\sqrt{n})}$.

Proof. We will argue only in the RP case, the other one being quite similar. If x is not in L, then W is empty and thus $z_i = 0$ for each i. It follows that the Impagliazzo-Zuckerman algorithm correctly rejects x. If x is in L, an error is made only when all z_i's are 0. For $\gamma \in \{0,1\}^{\sqrt{n}}$, denote by p_γ the probability that the vector $\langle z_1,\ldots,z_{\sqrt{n}}\rangle$ is equal to γ; let q_γ represent the corresponding probability for the b_i's. We thus want to bound p_α, where $\alpha = 0^{\sqrt{n}}$. We have

$$\begin{aligned}
p_\alpha &= q_\alpha + p_\alpha - q_\alpha \\
&\leq q_\alpha + |p_\alpha - q_\alpha| \\
&\leq 2^{-\sqrt{n}} + d(\langle z_1,\ldots,z_{\sqrt{n}}\rangle,\langle b_1,\ldots,b_{\sqrt{n}}\rangle) \text{ by the definition of distance} \\
&\leq 2^{-\sqrt{n}} + 2^{-\Omega(\sqrt{n})} \\
&= 2^{-\Omega(\sqrt{n})}
\end{aligned}$$

□

We finally proceed to verify that the hypothesis of the above theorem is indeed satisfied. In the argument to be given, we will assume with no loss of generality that $||W||$ and $||\overline{W}||$ are both at least $\frac{2^n}{4}$. Indeed suppose first that A is a BPP algorithm, i.e. that for some constant α, we have

$$x \in L \Rightarrow ||W_x|| \geq (1/2+\alpha)2^n$$

and

$$x \notin L \Rightarrow ||W_x|| \leq (1/2-\alpha)2^n.$$

We modify A into A' as follows: A' flips a coin; if 0 then simulate A, if 1 then flip another coin; if 0 then flip $n-1$ coins and accept x in all cases, if 1 then flip $n-1$ coins and reject x in all cases. This new algorithm uses $n+1$ random bits and clearly $||W||$ and $||\overline{W}||$ are both at least $2^{n-1} = \frac{2^{n+1}}{4}$. Moreover, if $x \in L$, we have $||W|| \geq (\frac{1}{2}+\alpha)2^n + 2^{n-1} = (\frac{1}{2}+\frac{\alpha}{2})2^{n+1}$; similarly, if $x \notin L$, we have $||W|| \leq (\frac{1}{2}-\frac{\alpha}{2})2^{n+1}$. We thus have again a BPP algorithm

(although the constant has been changed from α to $\frac{\alpha}{2}$). Suppose next that A is an RP algorithm, i.e.

$$x \in L \Rightarrow ||W_x|| \geq 2^{n-1}$$

and

$$x \notin L \Rightarrow ||W_x|| = 0.$$

We modify A into A' as follows: A' flips a coin; if 0 then simulate A, if 1 then flip n coins and reject x in all cases. If x is not in L, we still have that W is empty; if x is in L, we do have $||W||$ and $||\overline{W}||$ of size at least $\frac{2^{n+1}}{4}$. Note that in this case we only guarantee that $\mu(W) \geq 1/4$ instead of $1/2$ but this is unimportant in the definition of an RP algorithm.

Lemma 4.2 *For* $i = 1, \ldots, \sqrt{n}-1$, $d(\langle h, y_{i+1}, z_1, \ldots, z_i\rangle, \langle h, y, b_1, \ldots, b_i\rangle) \leq 2i2^{\frac{-\sqrt{n}}{2}}$.

Proof. The proof will make use of five properties of the distance function:

P1 for any distributions $P, Q, R, d(P, Q) \leq d(P, R) + d(R, Q)$

P2 if U is the uniform distribution then $d(P \times U, Q \times U) = d(P, Q)$

P3 for any function f, $d(\langle x, y\rangle, \langle u, v\rangle) \geq d(\langle x, f(y)\rangle, \langle u, f(v)\rangle)$

P4 $d(\langle x, z\rangle, \langle y, z\rangle) = d(\langle x\rangle, \langle y\rangle)$

P5 $d(\langle x\rangle, \langle u\rangle) \leq d(\langle x, y\rangle, \langle u, v\rangle)$

Base of induction: $i = 1$
We have to show

$$d(\langle h, y_2, z_1\rangle, \langle h, y, b_1\rangle) \leq 2 \cdot 2^{\frac{-\sqrt{n}}{2}}$$

Let $y = uv$ where $|v| = 2\sqrt{n}$ and recall that $y_2 = h(y)r_2$ where r_2 also has length $2\sqrt{n}$. Since v and r_2 consist of truly random bits, we can apply P2 to get

$$d(\langle h, y_2, z_1\rangle, \langle h, y, b_1\rangle) = d(\langle h, h(y), z_1\rangle, \langle h, u, b_1\rangle)$$

It follows from lemma 3.1 that we can split the RHS in two cases, namely $z_1 = 1$ (i.e. $y \in W$) and $z_1 = 0$ (i.e. $y \notin W$). This leads to

$$\text{Prob}\,[y \in W] \cdot d(\langle h, h(y)\rangle_{y \in W}, \langle h, u\rangle) + \text{Prob}\,[y \notin W] \cdot d(\langle h, h(y)\rangle_{y \notin W}, \langle h, u\rangle)$$

Recall the assumption that $\|W\|, \|\overline{W}\|$ are each at least $2^n/4$, or equivalently that $\mu(W), \mu(\overline{W})$ are each at most $3/4$. We can apply the leftover hash lemma with $m = n - 2\sqrt{n}, l = \sqrt{n} - 1$ to get

$$d(\langle h, h(y)\rangle_{y\in W}, \langle h, u\rangle) \leq 2^{1-\sqrt{n}}$$

$$d(\langle h, h(y)\rangle_{y\notin W}, \langle h, u\rangle) \leq 2^{1-\sqrt{n}}$$

Altogether we thus obtain an upper bound of

$$2 \cdot 3/4 \cdot 2^{1-\sqrt{n}} \leq 3 \cdot 2^{-\sqrt{n}} \leq 2 \cdot 2^{\frac{-\sqrt{n}}{2}} \text{ (for } n \geq 4)$$

Inductive step: $i - 1 \rightarrow i$

We assume inductively that

$$d(\langle h, y_i, z_1, \ldots, z_{i-1}\rangle, \langle h, y, b_1, \ldots, b_{i-1}\rangle) \leq 2(i-1)2^{\frac{-\sqrt{n}}{2}}$$

and we want to prove that

$$d(\langle h, y_{i+1}, z_1, \ldots, z_i\rangle, \langle h, y, b_1, \ldots, b_i\rangle) \leq 2i2^{\frac{-\sqrt{n}}{2}}$$

Let us denote by T_1 the term we wish to bound. We first apply the triangle inequality P1 to get

$$T_1 \leq d(\langle h, y_{i+1}, z_1, \ldots, z_i\rangle, \langle h, y_i, z_1, \ldots, z_{i-1}, b_i\rangle) + d(\langle h, y_i, z_1, \ldots, z_{i-1}, b_i\rangle, \langle h, y, b_1, \ldots, b_i\rangle)$$

The second term of the sum is equal, using P4, to

$$d(\langle h, y_i, z_1, \ldots, z_{i-1}\rangle, \langle h, y, b_1, \ldots, b_{i-1}\rangle).$$

By the induction hypothesis, this term is bounded by $2(i-1)2^{\frac{-\sqrt{n}}{2}}$. It thus remains to show that $d(\langle h, y_{i+1}, z_1, \ldots, z_i\rangle, \langle h, y_i, z_1, \ldots, z_{i-1}, b_i\rangle) \leq 2 \cdot 2^{\frac{-\sqrt{n}}{2}}$. Let us denote this term by T_2.

Using P4 again, it suffices to show that

$$d(\langle h, y_{i+1}, z_i\rangle, \langle h, y_i, b_i\rangle) \leq 2 \cdot 2^{\frac{-\sqrt{n}}{2}}$$

But

$$\begin{aligned} d(\langle h, y_{i+1}, z_i\rangle, \langle h, y_i, b_i\rangle) &= d(\langle h, h(y_i)r_{i+1}, z_i\rangle, \langle h, h(y_{i-1})r_i, b_i\rangle) \\ &= d(\langle h, h(y_i), z_i\rangle, \langle h, h(y_{i-1}), b_i\rangle) \text{ by P2} \\ &\leq d(\langle h, y_i, z_i\rangle, \langle h, y_{i-1}, b_i\rangle) \text{ by P3} \end{aligned}$$

Iterating this process $i-1$ times, we get to

$$d(\langle h, h(y), z_i\rangle, \langle h, u, b_i\rangle)$$

As in the basis case, we can split this last distance in two parts namely $z_i = 1$ (i.e. $y_i \in W$) and $z_i = 0$ (i.e. $y_i \notin W$). This leads to

$$\text{Prob}\,[y_i \in W]\cdot d(\langle h, h(y)\rangle_{y_i \in W}, \langle h, u\rangle) + \text{Prob}\,[y_i \notin W]\cdot d(\langle h, h(y)\rangle_{y_i \notin W}, \langle h, u\rangle)$$

Applying P5 to the inductive hypothesis, we get $d(\langle y_i\rangle, \langle y\rangle) \leq 2(i-1)2^{\frac{\sqrt{n}}{2}}$, from which we deduce Prob $[y_i \in W] \leq 3/4 + 2(i-1)2^{\frac{\sqrt{n}}{2}}$, and Prob $[y_i \notin W] \leq 3/4 + 2(i-1)2^{\frac{\sqrt{n}}{2}}$.

Using the leftover hash lemma as in the basis case, we thus get

$$T_2 \leq 2(3/4 + 2(i-1)2^{\frac{-\sqrt{n}}{2}})2^{1-\sqrt{n}}$$

which is clearly less than $2 \cdot 2^{\frac{-\sqrt{n}}{2}}$ since i is at most $\sqrt{n}$.

□

Observe that the $O(n)$ bits defining $r_2, \ldots, r_{\sqrt{n}}$ are used effectively. If we were using these bits to build fewer strings of longer length, the algorithm A would be executed fewer times, hence the error probability would not decrease as much. On the contrary, if we were using more strings of shorter length, the constructed y_i's would not look random enough

5 How to Use a Weak Random Source

Definition 5.1 $\langle X_1, \ldots, X_r\rangle$ *is a* weak random source *(w.r.s) if*

$$(\forall i)(\forall(\alpha_1, \ldots, \alpha_{i-1}) \in \{0,1\}^{i-1})\left[\frac{1}{4} \leq \text{Prob}\,[X_i = 1 \mid X_1 = \alpha_1, \ldots, X_{i-1} = \alpha_{i-1}] \leq \frac{3}{4}\right]$$

(even given previous events, X_i is fairly random).

Observation: If $X = \langle X_1, \ldots, X_r\rangle$ is a w.r.s. then $C(X) \leq \left(\frac{3}{4}\right)^r$.

Lemma 5.2 (Reheated Leftover Hash) *Let H be the canonical set of hash functions from $\{0,1\}^n \to \{0,1\}^m$. Assume $C(X) \leq 2^{-(m+2l)}$. Then $\langle x, h(x)\rangle_{h \in_R H}$ is 2^{-l}-quasi-random.*

Proof. Immediate from the proof of the leftover hash lemma (where we just used the fact that Prob $[X = X']$ is small, but X didn't have to be uniform random). □

We will use a w.r.s. (and no other randomness) in simulating a BPP algorithm (which expects perfectly random bits). Take a w.r.s. $\langle X_1, \ldots, X_{10n} \rangle$. Let $k = (c+1)\log n$ and break the w.r.s. into n/k pieces of length $10k$, i.e., let $Y_1 = \langle X_1, \ldots, X_{10k} \rangle$, $Y_2 = \langle X_{10k+1}, \ldots, X_{20k} \rangle$, etc.

Now we use hash functions $h : \{0,1\}^{10k} \to \{0,1\}^k$ (note that there are only $2^{20(c+1)\log n} = n^{O(1)}$ of these functions). Let $r = h(Y_1)h(Y_2)\cdots h(Y_{n/k})$ (n bits). We assert that $\langle h, r \rangle$ is close to uniform random, i.e.,

$$d(\langle h, r \rangle, U) \leq \frac{n}{k} 2^{-k} \leq n^{-c}.$$

Proof of assertion. Observe that $C(Y_i \mid Y_1, \ldots, Y_{i-1})$ is small, i.e.,

$$\text{Prob } [Y_i = Y_i' \mid Y_1, \ldots, Y_{i-1}] \leq \left(\frac{3}{4}\right)^{10k}.$$

By the reheated leftover hash lemma, $d(Y_i, U) \leq 2^{-k}$ even if we condition on $h, Y_1, \ldots, Y_{i-1}$. Add up the distances for each Y_i to get $\frac{n}{k} 2^{-k}$.

This completes the proof that $\langle h, r \rangle$ is n^{-c} quasi-random. If we could pick a random h we would just use $\langle h, r \rangle$ in our BPP algorithm and thereby introduce negligible error. Of course we cannot pick a random h, but we can simulate the BPP algorithm with each of them and take the majority vote. **Note:** We cannot in general use a w.r.s. to get a single random bit with bias less than $\frac{1}{4}$, i.e.,

$$(\forall f : \{0,1\}^r \to \{0,1\})(\exists \text{ a w.r.s. } \langle X_1, \ldots, X_r \rangle) \left[\text{Prob } [f(X_1, \ldots, X_r) = 1] = \frac{1}{4}\right].$$

Thus we simulated BPP without simulating even one good coin flip!

References

[1] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random generation from one-way functions. *21st STOC*, pages 12–24, 1989.

[2] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. *30th FOCS*, pages 248–253, 1989.

[3] D. Zuckerman. Simulating BPP Using a General Weak Random Source. *32nd FOCS*, pages 79–89, 1991.

Applications of Universal Hashing

Notes by Hervé Caussinus and Mike Saks

Tuesday Evening, February 1st

1 Summary of results on amplification

Recall the "amplification" problem: given a randomized poly-time algorithm for a language L that, on input $x \notin L$ always rejects and for $x \in L$ accepts with probability at least $1/2$, using $n = p(|x|)$ random bits, we want to modify the algorithm so that the probability of error (false rejection) is reduced. The new algorithm should still be poly-time, and should use as few random bits as possible. The table below summarizes the known results that were surveyed in the previous lecture.

Number of random bits	Pr($error$)	Reference
kn	2^{-k}	folklore
n	n^{-c}	Karp-Pippenger-Sipser [5]
$2n$	n^{-c}	Chor-Goldreich [2]
$5n$	$2^{-\sqrt{n}}$	Impagliazzo-Zuckerman [2]
n	$2^{-\frac{n}{\lg n}}$	Nisan [13]
$O(n)$	2^{-n}	Cohen-Wigderson ...' [4]

2 Discrete Random Variables

If X is a random variable, then, as usual, $\mathbf{E}(X)$ denotes its expectation, and $\sigma^2(X)$ denotes its variance, $\mathbf{E}((X - \mathbf{E}(X))^2)$, and $\sigma(X) = \sqrt{\sigma^2(X)}$ is its standard deviation.

Definition 2.1 Let $Z_1, \ldots, Z_n$ be random variables, Z_i taking values in a finite set S_i. $Z_1, \ldots, Z_n$ are *mutually independent* if

$$\forall (s_1, \ldots, s_n) \in S_1 \times \cdots \times S_n, \Pr(Z_1 = s_1, \ldots, Z_n = s_n) = \prod_{i=1}^{n} \Pr(Z_i = s_i)$$

Definition 2.2 $Z_1, \ldots, Z_n$ are *pairwise independent* if

$$\forall i, j \in \{1, \ldots, n\},\ i \neq j,\ \Pr(Z_i = s_i, Z_j = s_j) = \Pr(Z_i = s_i) \Pr(Z_j = s_j)$$

Note that mutual independence implies pairwise independence.

A "tail inequality" for a random variable X provides bounds on the probability that it deviates from the mean. Three standard tail inequalities: Markov's Inequality, Chebyshev's Inequality and the Chernoff bound.

Theorem 2.3 *(**Markov's Inequality**) Let X be a random variable taking values in the positive reals, then*

$$\Pr(X \geq t) \leq \frac{\mathbf{E}(X)}{t}$$

.

Proof. Let $F(x)$ be the probability distribution function of X, i.e., $F(x)$ =Prob$[X \geq x]$.

$$\mathbf{E}(X) = \int_0^\infty x dF(x) \geq \int_t^\infty x dF(x) \geq \int_t^\infty t dF(x) \geq t \int_t^\infty dF(x) = tF(t)$$

Of course, this inequality is only interesting when $t > \mathbf{E}(x)$.

Theorem 2.4 *(**Chebyshev's Inequality**) For any random variable X and positive real number λ,*

$$\Pr(|X - \mathbf{E}(X)| \geq \lambda) \leq \frac{\sigma^2(X)}{\lambda^2}$$

Proof.

$$\begin{aligned}\Pr(|X - \mathbf{E}(X)| \geq \lambda) &= \Pr((X - \mathbf{E}(X))^2 \geq \lambda^2) \\ &\leq \frac{\mathbf{E}((X - \mathbf{E}(X))^2)}{\lambda^2} \qquad \text{(Markov inequality)}\end{aligned}$$

□

The expectation of any real valued random variables is additive, i.e., if $Z_1, Z_2, \ldots, Z_n$ are real valued random variables, then the random variable $X = Z_1 + Z_2 + \ldots + Z_n$ has expectation $\mathbf{E}(Z_1) + \mathbf{E}(Z_2) + \ldots + \mathbf{E}(Z_n)$. If the Z_i are pairwise independent then variance is also additive: $\sigma^2(X) = \sigma^2(Z_1) + \ldots + \sigma^2(Z_n)$. Thus Chebyshev's inequality applies to the random variable X. In particular, if $Z_1, Z_2, \ldots, Z_n$ are pairwise independent random variables, such that $\Pr(Z_i = 1) = \Pr(Z_i = 0) = 1/2$ then:

$$\Pr(|X - \frac{n}{2}| \geq \epsilon n) \leq \frac{1}{4\epsilon^2 n}$$

In the case that $Z_1, Z_2, \ldots, Z_n$ are mutually independent we have:

Theorem 2.5 *(**Chernoff Bound**) $Z_1, \ldots, Z_n$ mutually independent. $X = \Sigma Z_i$.*

$$\Pr(|X - \mathbf{E}(X)| \geq \epsilon n) \leq 2^{-\frac{\epsilon^2 n}{3}}$$

For a proof, see, for example, Alon, Spencer [1].

3 Universal Hashing

Let S and T be two finite sets ($|T| = t$) and let $F(S,T)$ denote the set of functions from S to T. If H is any subset of $F(S,T)$, we define a probability space on H which assigns probability $1/|H|$ to each function in H.

Definition 3.1 $U \subseteq F(S,T)$ is a *universal hashing class* if for all $x \neq y \in S$, and $i, j \in T$, and for h chosen uniformly at random from U:

$$\Pr_U(h(x) = i, h(y) = j) = \frac{1}{t^2}$$

As a direct concequence we have if $x \neq y$: $\Pr(h(x) = h(y)) = \Pr(h(x) = i) = \frac{1}{t}$. An important fact about universal hashing classes is the following.

Lemma 3.2 *The set of random variables* $\{h(x) \mid x \in S\}$ *defined by a random* $h \in U$ *are pairwise independent.*

Note that $F(S,T)$ is itself a universal hashing class. We will consider a variety of applications of universal hashing classes to complexity theory. In these applications, what is needed is a very small universal hashing class. One standard construction is as follows:

Assume $|S| = q = p^k$ where p is a prime, and identify the elements of S with the finite field F_q. For $a, b \in S$, define the function $h_{a,b} : S \longrightarrow S$ by $h_{a,b}(x) = a + bx$, and define $H(S,S)$ to be the set $\{h_{a,b} | a, b \in F_q\}$. Then for fixed $x \neq y$ in S, and $h_{a,b}$ chosen uniformly at random from $H(S,S)$,

$$\begin{aligned} \Pr(h_{a,b}(x) = i, h_{a,b}(y) = j) &= \Pr(a + bx = i, a + by = j) \\ &= \Pr(a = \frac{jx - iy}{x - y}, b = \frac{i - j}{x - y}) = \frac{1}{t^2} \\ &= \Pr(h_{a,b}(x) = i) \Pr(h_{a,b}(y) = j) \end{aligned}$$

So, $H(S,S)$ is a universal family of hash functions of size $|S|^2$. If T is any set such that $|T|$ divides $|S|$, we can use this to obtain a universal hash family from S to T. Fix an arbitary mapping g from S to T in such a way that the inverse image of any $z \in T$ has size $|S|/|T|$. It is customary to use the notation s mod T to denote $g(s)$. If h is a function mapping S to S, then define h^T to be the function from S to T given by $h^T(s) = h(s)$ mod T. Then it is easy to show that if U is a universal class of hash functions from S to S, then $\{f^T | f \in U\}$ is a universal class of hash functions from S to T. Thus $H(S,T) = \{h^T_{a,b} | a, b \in S\}$. is a universal hash family from S to T.

Remark. In applications, $|S|$ may not be equal to a prime power and $|T|$ may not be a divisor of $|S|$. If $|S|$ is not a prime power it is usually sufficient to embed S in a larger set whose size is a prime power (such a set can clearly be chosen to have size less than $2|S|$). If $|T|$ is not a divisor of S, then one can use a map g for which all of the inverse images of elements have size within one of each other. This leads to a family of functions that is not a universal hash family but is "close enough" so that the applications go through. These details will not be discussed further.

3.1 Derandomization by universal hashing: a simple example

Definition 3.3 Let $G = (V, E)$ be any graph and A, B be two subsets of the vertex. $E(A, B)$ is the number of edges between A and B.

Suppose we define a random map f from the vertex set V to $\{0, 1\}$, where each vertex is randomly assigned to each set with probability $1/2$. Then for each edge (a, b) there is a $1/2$ chance that the endpoints of the edge get mapped to different values. Thus, by linearity of expectation, the expected number of edges in $E(f^{-1}(0), f^{-1}(1))$ is at least $|E|/2$. This shows:

$$\exists A \subseteq V \text{ s.t. } E(A, \bar{A}) \geq \frac{|E|}{2}$$

Now, consider the algorithmic problem of finding such an A. Of course, exhaustive search would take exponential time. Using universal hashing, such an A can be found deterministically in NC^1. The above proof that the expectation of $E(A, \bar{A})$ is at least $|E|/2$ only relies on pairwise independence, i.e., for any two vertices a and b, $f(a)$ and $f(b)$ are pairwise independent. Thus, the same fact would hold if we choose f uniformly at random from a universal hash family. Thus the family contains an f such that the set $A = f^{-1}(0)$ will satisfy the required property. Since there are universal hash families of size $|V|^2$, we can exhaustively search them in polynomial time (and, in fact, NC^1).

4 Dictionary Problem

In the static dictionary problem, we are given a subset S (of size n) out of a large universe V (where V is, say, the set of r bit integers for some r), and we need to store the elements of S in a table T (of size $t \geq n$) in such a way that we can quickly test, for an arbitary $x \in V$, whether $x \in S$. The most straightforward solution is to sort S and store the elements of S in T in sorted order. This takes $n \log n$ time in worst case to build the table, and then $\log n$ time in worst case to test whether $x \in S$. (Here we assume a model in which a memory register can store a r bit integer and arithmetic operations and comparisons on such integers can be done in unit time).

An alternative approach is to use one of the standard textbook hashing algorithms. These typically can have very bad worst case performance, but, if S is a randomly chosen subset of V, then the expected time to store S in T is only $O(n)$ and the expected search time to determine whether $x \in S$ is $O(1)$.

This suggests the problem: is it possible to find a scheme that works *for any dictionary* S, with a table of size not much bigger than $|n|$ that can be constructed in time $O(n)$, and has worst case search time $O(1)$. The following construction of Fredman, Komlös and Szemerédi [6] essentially achieves this.

A function h from V to T is *perfect* for the set $S \subseteq V$ if the restriction of h to S is 1-1. A family H of functions from V to T is a *n-perfect hash family* if for every $S \subset V$ of size n, there is an $h \in H$ that is perfect for S.

Suppose now that we have a perfect family H of hash functions. Then given S, there is an $h \in H$ that maps S in a 1-1 fashion to T. Choose such an h and use it to map S into the table. Now, to do a search for an element x, we need only compute $h(x)$ and check that entry of the table to see if it is x.

This approach requires some auxiliary storage to store the particular function h. Furthermore, while we only have to look once in the table for x, we also have to compute $h(x)$. Thus to be useful, we need a the perfect hash family H to be such that any function h can be stored in small space, and also that $h(x)$ can be computed very quickly, i.e., in constant time.

Suppose that h is a random function from V to T. Consider a fixed S of size n, and let C be the random variable which is the number of collisions, i.e., the number of pairs $x, y \in S$ with $x \neq y$ such that $h(x) = h(y)$. Then h is not perfect for S if and only if $C \geq 1$, and by Markov's inequality the probability that $C \geq 1$ is at most $\mathbf{E}(C)$. C is equal to $\sum_{x \neq y} C_{x,y}$ where $C_{x,y}$ is the random variable that is 1 if $h(x) = h(y)$ and 0 otherwise. Now for $x, y \in S$ with $x \neq y$, the probability that $h(x) = h(y)$ is $1/t$ and thus $\mathbf{E}(C_{x,y}) = 1/t$. By linearity of expectation $\mathbf{E}(C) = n(n-1)/2t$.

Thus, if we choose the table size t to be at least n^2 then for each S, the probability that a random h is 1-1 on S is at least $1/2$.

This analysis only depends on the fact that for each $x \neq y$, $h(x)$ and $h(y)$ are pairwise independent. Thus the same analysis goes through if we choose h at random from a universal hash family from V to T. What this means is that if $t \geq n^2$ and U is any universal hash family from V to T, then for each $S \subseteq V$ of size n, at least half the functions $h \in U$ are perfect on S.

Now if we take U to be the universal hash family $H(V,T)$ defined earlier, then each $h \in H(V,T)$ is represented by a pair a, b of r bit integers (interpreted as elements of the field $GF(2^r)$, and computing h can be done in a constant number of arithmetic operations. Thus, we have a scheme that achieves $O(1)$ look up time with a table of size n^2, and a constant amount of auxiliary storage to store h. Given S, to store S we need to find an $h \in H(V,T)$ that is perfect for S. This can be done by testing each h one at a time, which takes $|V|^2 n$ time in worst case. However, since at least half of all $h \in H(V,T)$ are perfect for S, if we consider the $h \in H(V,T)$ in random order the expected number of functions we need examine is only constant.

The next problem is to improve this scheme so that the total storage requirements (table size plus auxiliary storage) is at most linear in n. The FKS strategy for doing this is a two-stage hashing scheme. The storage scheme will consist of two tables, T and T' each of size $t = 2n$ and a constant amount of additional space, where T stores the elements of S and T' stores some auxiliary information.

For the first stage, we will find a hash function that maps S into the integers $\{1, 2, \ldots 2n\}$. in such a way that the total number of collisions is small, at most $n/2$. Indeed if we take f at random from a universal hash family then the expected number of collisions, as calculated previously, is $n(n-1)/2t$, which for $t = 2n$ is less than $n/4$. By Markov's inequality, the number of collisions is at most $n/2$ with probability $1/2$. So we can find such an f by searching randomly through the hash family, looking only at an expected number of functions that is constant.

Suppose we have such a function f. Let S_i be the elements of S that are mapped to i by f, and let $n_i = |S_i|$. Now for each i, we find a function h_i that maps S_i perfectly into the set $\{1, 2, \ldots, n_i^2\}$. For this, we can select h_i from a universal class of hash functions from V to $\{1, 2, \ldots, n_i^2\}$, since the "table size" is the square of n_i.

Define s_i to be the sum $n_1^2 + n_2^2 + \ldots + n_i^2$. Note that, since the number of collisions is $\sum_i n_i(n_i - 1)/2$, s_n is at most twice the number of collisions plus n, so is at most $2n$. Now, we build the tables T and T' as follows. Location i of T' will contain the pair s_{i-1}, h_i. T will contain the elements of S, where for each $i \leq 2n$ each $s \in S_i$ is stored in location $s_{i-1} + h_i(s)$. It is easy to see that this scheme maps elements of s into T in a 1-1 manner. To search for an arbitary $s \in V$, we first evaluate $j = h(s)$, retrieve s_{j-1} and h_j from location j of T' and then look in location $s_{j-1} + h_j(s)$ of T.

Some additional tricks in the FKS paper allow the total space to be reduced to $n + o(n)$ locations. Extensions of the above ideas can also be extended to handle the dynamic dictionary problem, where insertions and deletions are allowed [5].

5 Interactive Proofs

The model of interactive proofs was introduced by Goldwasser, Micali and Rackoff [9]. We content ourselves with an informal description of the model; for more details see the above paper.

We have a language L, and two parties, called the *prover* and the *verifier* (who are each modeled by probabilistic Turing machines). The parties are given a string x as input and the prover attempts to convince the verifier (possibly falsely) that $x \in L$ and the verifier is attempting to check the prover. Such a proof should have the property that if $x \in L$ then with high probability, the prover can convince the verifier that this is so. On the other hand, if $x \notin L$ then it should be unlikely that the prover convinces the verifier.

We consider protocols that work as follows. Both parties get the input x. The verifier then does some computation and then sends a message to the prover. The prover does some computation and then sends a message back to the verifier. Such an exchange is called a *round* of communication. The protocol continues in this way until at some point the verifier decides to "accept" that $x \in L$ or "reject".

A protocol is specified by a pair of Turing machines, one for the prover and one for the verifier that behave according to the above general rules, and a threshold p_1 with $0 < p_1 \leq 1$. The language accepted by the protocol is the set of strings x such that when the protocol is run on input x, the verifier accepts x with probability at least p_1.

A language L is in the complexity class IP if there is a protocol that accepts L, such that the verifier is a probabilistic poly-time machine and the total amount of communication is polynomial in $|x|$, which satisifies the additional property of *soundness*: there is an $\epsilon > 0$ such that if $x \notin L$ then for any (possibly dishonest) strategy of the prover (i.e., any Turing machine) the probability that the verifier accepts x is at most $p_2 - \epsilon$. If there is such a protocol that operates in at most k rounds then L is said to be in $IP(k)$.

One of the simplest examples of such a protocol is the protocol for graph nonisomorphism [7]. Here the input x consists of (the adjacency matrix of) a pair of graphs G_0 and G_1 on the same vertex set and L is the set of such pairs that are not isomorphic. The following is an interactive proof for L. The verifier generates a random bit b (0 or 1) and also generates a random permutation σ of the vertex set of the graph. He computes the graph $H = \sigma(G_b)$ obtained by permuting the vertices according to σ and sends H to the prover. The prover then is supposed to determines whether H is isomorphic to G_0; if so he sends back the bit 0, and otherwise he sends back the bit 1. The verifier then accepts if and only if the bit he receives is equal to b.

It is easy to see that

- if $G_0 \not\equiv G_1$ then Pr(verifier accepts) = 1,
- if $G_0 \equiv G_1$ then for *any* prover Pr(verifier accepts) $\leq \frac{1}{2}$.

Notice that it is essential for this protocol that the random coins that the verifier flips are private, since otherwise a dishonest prover could always just send back the bit b and cause the verifier to accept. Remarkably, it is possible to modify the protocol so that all coins that are flipped can be flipped publicly. Indeed, this is a special case of the following general result of Goldwasser and Sipser [10]:

Theorem 5.1 *If L has a k round interactive proof using private coins then L has a $k+2$ round interactive proof using public coins, i.e.,*

$$\textit{Public coins } \mathbf{IP}(k+2) = \textit{Private coins } \mathbf{IP}(k)$$

(The model of public coin interactive proofs was studied as Arthur-Merlin games by Babai and Moran [2].) To get the flavor of the proof we follow the presentation of Schöning that shows how to eliminate the private coins for the special case of graph non-isomorphism. For simplicity, we assume that G_0 and G_1 have no nontrivial automorphisms. Let S be the set of all possible graphs that the verifier might send.

$$S = \bigcup_{\Pi \in S_n} \{\Pi(G_0), \Pi(G_1)\}$$

Then we have:

- If $G_0 \equiv G_1$ then $|S| = n!$,
- if $G_0 \not\equiv G_1$ then $|S| = 2n!$.

So we can recast the prover's job to be: prove that $|S| = 2n!$. For technical reasons that will become apparent, we define $S' = S \times S \times S \times S$ so that:

- If $G_0 \equiv G_1$ then $|S'| = n!^4$,
- if $G_0 \not\equiv G_1$ then $|S'| = 16n!^4$.

Set m to be the size of S'. So it suffices for the prover to convince the verifier that $m \geq 16(n!)^4$. Let V be the set of all possible $n \times n$ adjacency matrices of graphs and consider the universal hashing class $H(V^4, T)$ where $|T| = t = 8(n!)^4$.

Define the following public coin protocol. First the verifier picks a random h from $H(V^4, T)$ and a random $i \in T$, and sends both to the prover. The prover then looks for an $x \in S'$ (i.e., a sequence of four graphs each of which is either isomorphic to G_0 or G_1) such that $h(x) = i$. He then sends such an x, together with the four vertex permutations of each of the graphs that show that they are each isomorphic to G_0 or G_1. The verifier checks that this is indeed the case and accepts if it is.

Notice that the prover succeeds in convincing the verifier precisely when there is an $x \in S'$ such that $h(x) = i$. We now show that if G_0 and G_1 are not isomorphic, the probability that the there is such an $x \in S'$ is at least $1/6$, and if G_0 and G_1 are isomorphic then the probability that there is such an $x \in S'$ is at most $1/8$. Thus the above protocol is an IP proof.

Suppose first that G_0 and G_1 are not isomorphic. Say that the hash function h is good if h restricted to S' maps onto at least $t/2$ elements of t. If h is bad then all of the elements of S' are mapped to a set of size at most $t/2$. We want to upper bound the probability that a random h is bad. Defining (as in the analysis of perfect hashing), n_i to be the number of elements of S' mapped by h into i, we have that for bad h the number C of collisions that occur satisfies:

$$C = \Sigma_{i \in T} \binom{n_i}{2} \geq t/2 \binom{\frac{2m}{t}}{2} = \frac{m(2m-t)}{2t},$$

where the inequality comes from the convexity of $\binom{n_i}{2}$, and the assumption that h maps elements to at most $t/2$ elements.

Thus the probability that h is bad is bounded above by the probability that $C \geq m(2m-t)/2t$. Now, by Markov's inequality this probability is bounded above by $E(C)/(m(2m-t)/t)$, which is equal to $(m-1)/(2m-t)$ since $E(C) = m(m-1)/2t$ (as computed previously). Since $t = m/2$, this probability is at most $2/3$. Thus the probability that h is good is at least $1/3$. Now if h is good, then with probability at least $1/2$, the index $i \in T$ that is chosen by the verifier is one for which there is an $x \in S'$ with $h(x) = i$. Thus the probability that the prover can make the verifier accept is at least $1/6$.

On the other hand, if G_0 and G_1 are isomorphic, then $S' = (n!)^4$=t/8, and for any h, the fraction of i for which there is an $x \in S'$ with $h(x) = i$ is at most $1/8$, so this upper bounds the probability that the verifier will accept.

6 BPP $\subset \Sigma^2$

A probabilistic polynomial time Turing machine M is a poly-time Turing machine that takes as input a pair x and y where $|y| = r(|x|)$ for some fixed polynomial r and returns "accept" or "reject". For such a Turing machine, and for each string x, W_x is the set of y of length $r(|x|)$ such that M accepts x, y. The language L accepted by M is the set of x for which W_x contains at least half of all strings of length $r(|x|)$. The Turing machine is said to run with "bounded error" if there is a constant $\epsilon > 0$ such that for $x \notin L$, W_x contains at most a $(1/2 - \epsilon)$ fraction of all strings of length $r(|x|)$. The complexity class BPP consists of those languages that are accepted by a bounded error probabilistic time Turing machine.

Sipser [14] used universal hashing to show that BPP is contained in the polynomial time hierarchy. His original proof showed that $BPP \subseteq \Sigma_4$; Gac observed that the proof actually shows $BPP \subseteq \Sigma_2$. The proof is sketched here.

Recall that Σ_2 is the set of languages L for which there exists a polynomial time Turing machine that takes as input triples of strings x, y, z where $|y| = p(|x|)$ and $|z| = q(|x|)$ and outputs accept or reject, such that $x \in L$ if and only if there is a y such that for all z: M accepts x, y, z.

Suppose that L is a language that is accepted by some BPP machine M. Set $n = r(|x|)$. Using standard amplification techniques, we can assume that if $x \notin L$ then the fraction of y of length n that are in W_x is at most

$2^n/q(n)$ for any fixed polynomial q; it will be convenient to chose $q(n) = 4n$. Define the parameter $m = n - 1 - \log_2 n$ (and assume m is an integer) so that $2^n/4n = 2^{m-1}$. The problem of determining whether $x \in L$ can then be reformulated as determining whether W_x is "large" (of size at least $n2^m$ or "small" (of size at most 2^{m-1}). Note that the problem of determining whether a given y is in W_x is poly-time computable, using the machine M.

Let H be a universal class of hash functions from $\{0,1\}^n$ to $\{0,1\}^m$ (having size 2^{2n}).

Claim $x \notin L \longleftrightarrow$ there is a set $h_1, h_2, \ldots, h_m$ of functions in H such that for any $y \in W_x$, there is an $i \in \{1, 2, \ldots, n\}$ such that for all $z \in W_x - \{y\}$, $h_i(y) \neq h_i(z)$.

For a set $W \subset \{0,1\}^n$, and $y \in W$, a function h isolates y in W if for all $z \in W - \{y\}$, $h(y) \neq h(z)$. Then the predicate on the right says that there is a family of m functions in H such that each element of W_x is isolated by some member of the family.

Notice that if the claim is true, then it gives a Σ_4 algorithm for L. In fact the algorithm is in Σ_2. We can replace a predicate of the form: there exists $i \in \{1, 2 \ldots, m\}$ such that for all $z \in W_x - \{y\}$, $R(i, z)$, by the predicate: for all sequences $(z_1, z_2, \ldots, z_m) \in (W_x - \{y\})^m$, $R(1, z_1) \vee R(2, z_2) \vee \ldots \vee R(m, z_m)$, thereby eliminating the second existential quantifier. The resulting predicate is thus in Σ_2 because the eliminated quantifier quantifies only over polynomially many elements.

So it remains to prove the claim. Consider first the case that $x \in L$. Each $h \in H$ can isolate no more than 2^m elements of W_x, and since W_x has at least $n2^m > m2^m$ no set of m functions can isolate all of them.

Next consider the case that $x \notin L$, so that $|W_x| \leq 2^{m-1}$. We will show that if we choose $h_1, h_2, \ldots, h_m$ at random from H then with positive probability, every element of W_x is isolated by at least one of them. This implies that the predicate is satisfied. If we fix $z, y \in W_x$ with $z \neq y$, and choose h at random from H, then the probability that $h(y) = h(z)$ is at most 2^{-m} by the property of universal hash classes. Thus the probability that h does not isolate y is at most $2^{-m}|W_x| \leq 1/2$. Now if we choose $h_1, h_2, \ldots, h_m$ at random, then the probability that y is not isolated by any of them is 2^{-m}. Thus the probability that there exists a $y \in W_x$ that is not isolated by any of them is at most $|W_x|2^{-m} \leq 1/2$, as required.

References

[1] N. Alon and J.H. Spencer. *The Probabilistic Method.* Wiley, 1991.

[2] L. Babai and S. Moran Arthur-Merlin Games: A randomized proof system and a hierarchy of complexity classes. *J. Comp.Sys. Sci.*, 36:254-276, 1988.

[3] B. Chor and O. Goldreich. On the Power of Two-Point Based Sampling. *Journal of Complexity*, 5:96–106, 1989.

[4] A. Cohen and A. Wigderson. "Dispersers, deterministic amplification, and weak random sources". *30th FOCS*, pages 14-19.

[5] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F.M. Heide, H. Rohnert, and R.E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. *29th FOCS*, pages 524–531, 1988.

[6] M.L. Fredman, J. Komlós and E. Szemerédi Storing a sparse table with O(1) worst case access time. *J. ACM* 31:538-544, 1984.

[7] O. Goldreich, S. Micali and A. Wigderson Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems. *J. ACM* 38:691-729, 1991.

[8] Oded Goldreich and Avi Wigderson. Tiny Families of Functions with Random Properties : A Quality-size Trade-off for Hashing. *STOC*, 1994.

[9] S. Goldwasser, S. Micali and C. Rackoff. The knowledge complexity of interactive proof systems. *Siam Journal of Computing* 18:186-208, 1989.

[10] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *Advances in Computing Research 5: Randomness and Computation*, S. Micali, ed., JAI Press, Greenwich CT, 1989.

[11] R. Impagliazzo and L.A. Zuckerman. How to Recycle Random Bits. *30th FOCS*, pages 248–253, 1989.

[12] R.M. Karp, N. Pippenger, and M. Sipser. A Time-Randomness Tradeoff. *AMS Conference on Probabilistic Computational Complexity*, 1985.

[13] Noam Nisan. Pseudorandom Generators for Space Bounded Machines. *22nd STOC*, 1990.

[14] M. Sipser A complexity theoretic approach to randomness. *15th STOC*, 1990, 330-335.