Applications of Universal Hashing in Complexity Theory

V. Arvind and M. Mahajan
Institute of Mathematical Sciences, C.I.T Campus
Madras 600113, India
email: {arvind,meena}@imsc.ernet.in

Abstract

The aim of this tutorial article is to provide an introduction to methods based on universal hashing in the area of randomized computation. We discuss two applications each from randomized algorithms and interactive proof systems.

1 Introduction

Hashing as a useful data structuring technique for maintaining search tables is a familiar idea for most of us in the computing community. This technique is used to efficiently implement *dynamic dictionaries* (a dictionary is a data structure that supports find-key queries and delete-key and insert-key operations). The theoretical study of hashing has led to the discovery of the concept of universal hashing [8, 37] and the construction of classes of universal hash functions. Apart from data structuring, universal hashing has found applications in various other areas. These applications range from derandomization of probabilistic algorithms, simulating randomized algorithms using weak random sources, interactive proof systems, learning theory, and various other applications in complexity theory. Over the last decade universal hashing has turned out to be a very useful tool in the theoretician's tool kit.

1.1 Defining Universal Hash Functions

We introduce universal hashing via hashing as a data structuring solution to the dynamic dictionary problem. The usual data structures, that are based on the 'comparison model' for maintaining a search table of size n, require a worst-case time of $\Omega(\log n)$ for find-key queries and update operations. Hashing is an entirely different approach that achieves $O(1)$ find-key time. This is simply because hashing is not tied down to the comparison model but it allows 'algebraic' operations on keys. We will assume that the set of keys S is chosen from the *key space* M of size m, which is a totally ordered universe, without loss of generality, $M = \{0, 1, \cdots, m-1\}$. A *hash table* for S consists of a table T of n locations indexed by the *address space* $N = \{0, \cdots, n-1\}$ and a *hash function* h which is a mapping from M into N. Clearly, the solution to the dictionary problem using hash tables is trivial if $n = m$. This solution is practically useless because the key space size m is usually very large compared to $|S|$. The main issue is that of reducing the address space size n to a value close to $|S|$ while maintaining the objective of having $O(1)$ search time. Ideally we would want h to map different keys in S to different locations in the address space N. A *collision* is said to occur between keys $x \neq y$ if $h(x) = h(y)$.

A hash function $h : M \mapsto N$ is said to be *perfect* for a set $S \subseteq M$ if h causes no collisions on S. Given a perfect hash function for a static set of keys S, it processes find-key queries in $O(1)$ time. The problem is that if S is dynamic, then perfect hash functions will lose their property after some insert-key operations. An approach to designing good hash functions for the dynamic dictionary problem is to appropriately relax the requirements of perfect hashing for dynamic dictionaries. The idea is to choose a family of hash functions $\mathcal{H} : M \mapsto N$ with the property that for every set S of small size, a large fraction of the hash functions in $\mathcal{H}$ have very few collisions on the set S.

Definition 1.1 A family of functions $h : M \mapsto N$ is said to be *almost 2-universal* if for all $x, y \in M$ such that $x \neq y$, and for $h \in_R \mathcal{H}$, (This denotes that h is chosen uniformly at random from $\mathcal{H}$.)

$$Prob[h(x) = h(y)] \leq 1/n.$$

Exercise *Show that the set of all functions from $M \mapsto N$ is almost 2-universal.*

The above exercise shows that with respect to pairs of elements, a randomly picked hash function from any almost 2-universal class of hash functions $\mathcal{H}$ behaves like a randomly picked hash function from the set of all $M \mapsto N$ functions. This is the property that will be exploited in all applications of universal hashing.

The following exercise summarizes the application of almost 2-universal hash functions to dynamic dictionaries.

Exercise *Let $\mathcal{H} : M \mapsto N$ be an almost 2-universal hash family.*

1. *For all $x \in M$ and $S \subseteq M$ and $h \in_R \mathcal{H}$,*

$$E[|\{y \in S \mid h(x) = h(y)\}|] \leq |S|/n.$$

2. *Let $R = R_1 R_2 \cdots R_r$ be a sequence of find-key, insert-key, and delete-key operations in a dynamic dictionary of keys from M. Let s be the total number of insert-key operations in R. Let $\rho(h, R)$ denote the total cost of processing the sequence R using hash function h with address space N and the linked list scheme for collisions. Show that if the hash table is built by choosing $h \in_R \mathcal{H}$ then,*

$$E[\rho(h, R)] \leq r(1 + s/n).$$

   assuming that find-key, insert-key, and delete-key are unit cost operations.

3. *Based on the above two parts, design a dynamic hash table that takes $O(1)$ expected time for its operations and has a space utilization of 0.5.*

Our first goal is to construct small almost 2-universal classes of functions which are easy to represent in small space and easy to evaluate. As a first example, fix m and n, and choose a prime $p \geq m$ (By Bertrand's postulate there is a prime between m and $2m$). We will work over the field $Z_p = \{0, 1, \cdots, p-1\}$. Let $g : Z_p \mapsto N$ be the function given by $g(x) = x \pmod{n}$. For all $a, b \in Z_p$ define the hash function $h_{a,b} : Z_p \mapsto N$ as $h_{a,b}(x) = g(ax + b \pmod{p})$.

Consider the family of hash functions $\mathcal{H} = \{h_{a,b} \mid a, b \in Z_p, a \neq 0\}$. We prove that it is almost 2-universal.

Lemma 1.2 *$\mathcal{H} = \{h_{a,b} \mid a, b \in Z_p, a \neq 0\}$ is an almost 2-universal hash family from $M \mapsto N$.*

Proof. Let $x \neq y$ be two elements in Z_p. We wish to compute the probability that $h_{a,b}(x) = h_{a,b}(y)$ for a random hash function $h_{a,b}$. Let $ax + b \pmod{p} = r$ and $ay + b \pmod{p} = s$. Observe that since $a \neq 0$ it holds that $r \neq s$. Furthermore having fixed x and y, a and b are uniquely determined for each such choice of r and s. Thus $h_{a,b}(x) = h_{a,b}(y)$ iff $g(r) = g(s)$, i.e. $r - s = 0 \pmod{n}$.

For each $r \in Z_p$ the number of s such that $r \neq s$ and $r - s = 0 \pmod{n}$ is clearly bounded by $(p-1)/n$. There are p values that r could take and so the total number of such pairs r, s is bounded by $p(p-1)/n$. Thus the total number of hash function $h_{a,b}$ such that $h_{a,b}(x) = h_{a,b}(y)$ is bounded by $p(p-1)/n$. But $|\mathcal{H}| = p(p-1)$. Therefore it follows that $Prob[h_{a,b}(x) = h_{a,b}(y)] \leq 1/n$ and hence $\mathcal{H}$ is almost 2-universal. ∎

Exercise *How many bits are needed to specify a hash function $h_{a,b}$ from the above class? Contrast with the use of a totally random function as a hash function.*

We now give another example of a 2-universal class. Let $M = \{0,1\}^m$ and $N = \{0,1\}^n$ be the key and address spaces respectively. We shall view M and N as m-dimensional and n-dimensional vector spaces over $GF(2)$ respectively. Let the class of hash functions $\mathcal{H}$ be the set of all $m \times n$ matrices over $GF(2)$, where for each $A \in \mathcal{H}$, $A(x) = x \cdot A$ for $x \in \{0,1\}^m$.

Lemma 1.3 *The class $\mathcal{H} : M \mapsto N$ defined above is an almost 2-universal class of hash functions.*

Proof. Let $x \neq y$ be two m-vectors over $GF(2)$. Clearly, for any matrix $A \in \mathcal{H}$, $A(x) = A(y)$ iff $A(x-y) = 0$ (where 0 denotes, by abuse of notation, the null m-vector). Note that the set $\{A \mid A(x-y) = 0\}$ is a subgroup of $\mathcal{H}$ (under matrix addition over $GF(2)$). Furthermore, it is easy to see that for any $z \in \{0,1\}^n$ there is some $A \in \mathcal{H}$ such that $A(x-y) = z$, and the set $\{A \mid A(x-y) = z\}$ is a right coset of the subgroup $\{A \mid A(x-y) = 0\}$ of $\mathcal{H}$. Since all cosets are of the same size as the subgroup (by Lagrange's theorem) it follows that $|\{A \mid A(x-y) = 0\}| = |\mathcal{H}|/2^n$, since there are precisely 2^n different right cosets of $\{A \mid A(x-y) = 0\}$. Therefore it follows that

$$Prob[A(x) = A(y)] = 1/2^n$$

where $A \in \mathcal{H}$ is picked uniformly at random. This shows that $\mathcal{H}$ is almost 2-universal. ∎

1.2 Review of complexity classes

Before we discuss universal hashing further, we will very briefly review the definitions and relevant properties of some basic complexity classes.

We are interested in solving computational problems within prescribed computational resource-bounds (usually time and space). For the purpose of formal definitions we consider only decision problems in this subsection. The instances of these decision problems are encoded as strings over a fixed finite alphabet, say Σ. Thus the decision problems are essentially languages contained in Σ^*. Loosely speaking, a *complexity class* is a collection of decision problems which can be solved under prescribed bounds on computational resources.

In this article we restrict ourselves to different kinds of time-efficient algorithms, where efficient in time usually stands for *polynomial time*. An algorithm has polynomial running time if it halts within $n^{O(1)}$ steps for inputs of length n.

Definition 1.4 The class P consists of the set of all languages which have polynomial time algorithms.

The class NP consists of all languages L such that each $x \in L$ has a short (polynomial sized) proof of membership in L which can be verified in polynomial time.

Definition 1.5 For every language $L \in$ NP there is a polynomial-time algorithm A and a polynomial p such that

- $x \in L$ implies $\exists y \in \Sigma^{p(|x|)} : A(x, y)$ accepts.
- $x \notin L$ implies $\forall y \in \Sigma^{p(|x|)} : A(x, y)$ rejects.

Definition 1.6
- A *polynomial time many-one reduction* from a language L_1 to a language L_2 is a function f, computable in polynomial time, such that $x \in L_1$ iff $f(x) \in L_2$.
- A language L is NP-hard if every $L' \in$ NP is polynomial-time many-one reducible to L. L is NP-complete if $L \in$ NP and L is NP-hard.

We generalize polynomial-time computability to randomized algorithms.

Definition 1.7
1. The class RP (for randomized polynomial time) is the set of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x
   - $x \in L$ implies $Prob[A(x) \text{ accepts}] \geq 1/2$.
   - $x \notin L$ implies $Prob[A(x) \text{ accepts}] = 0$.
2. The class BPP (for bounded error probabilistic polynomial time) is the set of all languages L that have a randomized algorithm A running in worst-case polynomial time such that for any input x
   - $x \in L$ implies $Prob[A(x) \text{ accepts}] \geq 3/4$.
   - $x \notin L$ implies $Prob[A(x) \text{ accepts}] \leq 1/4$.

Both RP and BPP algorithms are known as Monte Carlo algorithms because they can err on inputs (although with only small probability). Notice that while BPP algorithms can err on both positive and negative inputs, RP algorithms may err only on positive inputs. If we reverse the acceptance and rejection condition for RP algorithms we get the complement class co-RP. co-RP algorithms may err only on negative inputs.

Randomized algorithms that always output the correct answer when they halt and have expected polynomial running time are termed *Las Vegas* algorithms.

Exercise *Show that a language that has both* RP *as well as* co-RP *algorithms has a Las Vegas algorithm.*

The polynomial hierarchy is a hierarchy of complexity classes built on top of NP by the following generalization. Recall that NP consists of problems whose YES instances have short proofs of membership, which can be easily verified. However, the task of finding a proof of membership, even if one knows that a proof exists, is not easy. Now consider a scenario where there is one all-knowing all-powerful person called the Prover, and one real-world resource-bounded person called the Verifier. (Typically the resource bound is polynomial time.) The Prover's task is to supply a membership proof; the Verifier's task is to verify the proof. If L is in NP, then there is a Prover and Verifier pair such that for every $w \in L$, the Prover can send a polynomial size message to the Verifier which will convince the Verifier about membership of w in L. On the other hand, for $w \notin L$, no message sent by any Prover can convince the Verifier that $w \in L$.

The Prover-Verifier interaction in NP can be generalized as follows: We add a third agent, a Challenger. Like the Prover, the Challenger is also all-knowing and all-powerful. The Prover and the Challenger send polynomial-sized messages to each other. The Verifier keeps track of the exchange, and at the end, declares one of the two to be the winner. When $x \in L$, the Prover has a *winning strategy*; no matter what messages the Challenger sends, the Prover has responses which lead the Verifier to declare the Prover the winner. On the other hand, when $x \notin L$, the Challenger has a winning strategy.

Informally, what is happening is that the Prover and the Challenger have a debate. The Prover asserts that $x \in L$, the Challenger asserts that $x \notin L$. The Verifier acts as a Referee in deciding which of the debators has more convincing arguments.

An important resource in this model is the number of rounds of message-passing. Technically, a round is a polynomially sized message in one direction. We can assume that rounds alternate in direction. Thus NP requires one round, beginning with the Prover. (In fact, in this case, the Challenger has no role to play.) Similarly, it is easy to see that all co-NP problems can be solved in one round, beginning with the Challenger. As we allow more rounds, we can capture increasingly complex problems above NP. We consider classes P-C[k] and C-P[k]: prover-challenger interaction, beginning with prover or challenger respectively, and requiring k rounds. (Of course, each round must not need more than polynomial time.) The union of all such classes, for any constant k, is precisely the polynomial hierarchy, denoted by PH. It has P at the base (0 alternation), NP and co-NP at the next level (P-C[1] and C-P[1] respectively), and problems of increasing complexity towering up in subsequent levels. Traditionally, the class P-C[k] is referred to as Σ_k^p and the class C-P[k] as Π_k^p. (This notation stems from a similar notation used for the arithmetic hierarchy in recursion theory). The formal definitions of these classes are given below.

Definition 1.8 1. A language L is in Σ_k^p if there is a polynomial-time algorithm A and a polynomial p such that

- $x \in L$ implies $\exists y_1 \in \Sigma^{p(|x|)} \forall y_2 \in \Sigma^{p(|x|)} \dots Q y_k \in \Sigma^{p(|x|)} : A(x, y_1, \dots, y_k)$ accepts.
- $x \notin L$ implies $\forall y_1 \in \Sigma^{p(|x|)} \exists y_2 \in \Sigma^{p(|x|)} \dots Q y_k \in \Sigma^{p(|x|)} : A(x, y_1, \dots, y_k)$ rejects.

(Here Q is the quantifier $\exists$ is k is odd and the quantifier $\forall$ if k is even.)

2. A language is in Π_k^p if its complement is in Σ_k^p.

3. $PH = \bigcup_{k>0} \Sigma_k^p$.

Exercise *Show that PH has the following nice structure:* $\Sigma_k^p \cup \Pi_k^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

We could equally well allow polynomial alternations; i.e. the number of alternations is not a constant, but is polynomially related to the size of the input. This gives us the class called Alternating Polynomial time APTIME (in the above notation, this is the class P-C[poly]).

Definition 1.9 A language L is in APTIME if there is a polynomial-time algorithm A and polynomials p, q such that

- $x \in L$ implies $\exists y_1 \in \Sigma^{p(|x|)} \forall y_2 \in \Sigma^{p(|x|)} \dots Q y_{q(|x|)} \in \Sigma^{p(|x|)} : A(x, y_1, \dots, y_{q(|x|)})$ accepts.
- $x \notin L$ implies $\forall y_1 \in \Sigma^{p(|x|)} \exists y_2 \in \Sigma^{p(|x|)} \dots Q y_{q(|x|)} \in \Sigma^{p(|x|)} : A(x, y_1, \dots, y_{q(|x|)})$ rejects.

APTIME is known to coincide with PSPACE, the class of problems which can be solved using space polynomial in the input size.

Exercise 1. *Show that* APTIME *is contained in* PSPACE.

2. *Define the* computation *graph of a* PSPACE *machine* M *for input* x *as the directed graph with nodes as configurations and arcs as transitions between configurations. Let* s *denote the start configuration and* t *denote the accepting configuration. Notice that the* PSPACE *machine* M *accepts* x *iff there is a directed* s*-*t *path in the computation graph.*

3. *In order to simulate* M *with an* APTIME *machine, design an* APTIME *machine that, on input* x*, checks for a directed* s*-*t *path in the computation graph of* M *for input* x*. (Hint: use a divide-and-conquer strategy to check for an* s*-*t *path.)*

1.3 Pairwise independence and universal hashing

In this subsection we discuss pariwise independece of random variables and a related application of universal hashing.

Definition 1.10 • A set of discrete random variables $X_1, X_2, \cdots X_n$ is said to be *indepedent* if

$$Prob[X_1 = x_1 \wedge X_2 = x_2 \wedge \cdots X_n = x_n] = \Pi_{i=1}^{n} Prob[X_i = x_i]$$

for all tuples $(x_1, x_2, \cdots, x_n)$ in the range of the n-tuple of random variables.

• A set of discrete random variables $X_1, X_2, \cdots X_n$ is said to be k*-wise windepedent* if

$$Prob[X_{i_1} = x_{i_1} \wedge X_{i_2} = x_{i_2} \wedge \cdots X_{i_k} = x_{i_k}] = \Pi_{l=1}^{k} Prob[X_{i_l} = x_{i_l}]$$

for all index subsets $i_1, \cdots, i_k$ and all tuples $(x_{i_1}, x_{i_2}, \cdots, x_{i_k})$ in the range of the corresponding k-tuple of random variables.

Recall from Section 1.1 that the condition $Prob[h(x) = h(y)] \leq 1/n$ intuitively means that for pairs $x \neq y$ in the key space M, the values $h(x)$ and $h(y)$ behave as if they were the images of an arbitrary function picked at random. In other words, it is as if $h(x)$ and $h(y)$ are pairwise independent. We now define a slightly stronger notion of universal hashing that captures this pairwise independence property.

Definition 1.11 Let M and N be the key and address spaces of sizes m and n respectively. A family of hash functions $\mathcal{H} : M \mapsto N$ is *2-universal* if for all $x \neq y \in M$ and any $u, v \in N$, and $h \in_R \mathcal{H}$

$$Prob[h(x) = u \wedge h(y) = v] = 1/n^2.$$

An important property of 2-universality is the following.

Exercise *Let* $\mathcal{H} : M \mapsto N$ *be a 2-universal family of hash functions. The set* $\{h(x) \mid x \in M\}$ *of* $|M|$*, defined by a random* $h \in_R \mathcal{H}$*, is a pairwise independent set of random variables.*

Here is an example of a 2-universal family of hash functions.

Exercise *Let* Z_p *be both the key space and the address space for the following hash function class* $\{h_{a,b} : a, b \in Z_p \mid \forall x \in Z_p \; h_{a,b}(x) = ax + b \pmod{p}\}$*. Show that this class is 2-universal.*

To choose the key and address spaces as Z_p is unrealistic, since computing a large prime p is not easy. We define some more 2-universal hash families from a key space M to an address space N, when the key space is $M = \{0,1\}^s$ and the address space is $N = \{0,1\}^t$, for $t < s$. Recall from finite field theory that we can impose a field structure on M so that it is the unique (upto isomorphism) finite field of size 2^s. Then, as in the above exercise, $\{h_{a,b} : a, b \in M \mid \forall x \in Z_p \ h_{a,b}(x) = ax + b\}$ (where $ax + b$ is computed in the field M) is easily seen to be a 2-universal family hashing M to M (prove it!). The following exercise shows how we can get a 2-universal $M \mapsto N$ family.

Exercise

- *Let $g : M \mapsto N$ be any* fixed *function that maps M onto N such that for every $y \in N$, $|g^{-1}(y)| = |M|/|N|$. Prove that the class $\mathcal{H} = \{g \circ h_{a,b} : a, b \in M \mid \forall x \in Z_p \ h_{a,b}(x) = ax + b\}$ is a 2-universal family from $M \mapsto N$*
- *Show how to define a (nearly) 2-universal hash family from an arbitrary key space M to an arbitrary sized address space N. Quantify what 'nearly' means by analysis.*

We now discuss an example where, using the pairwise independence property of 2-universal hash families, a specific randomized algorithm can be derandomized.

Given an undirected graph $G = (V, E)$, the *maxcut* problem is to find a subset A of V such that $E(A, \overline{A})$ has maximum possible cardinality (where $E(A, \overline{A})$ is the set of edges between A and $\overline{A}$). The maxcut problem is known to be NP-hard.

Consider the following randomized algorithm for the maxcut problem.

Step 1. For each $v \in V$, with probability $1/2$, include v in A;
Step 2. Output A;

Notice that for any edge $(u, v) \in E$ the probability that exactly one end point gets into A is $1/2$.

Exercise *Prove that the expected size of $E(A, \overline{A})$ for the subset A computed by the above algorithm is at least $|E|/2$. (Hint: the expectation of a sum of random variables is the sum of the expectations of each of the random variables.)*

It follows from the above exercise that for any graph G there exists $A \subseteq V$ such that $|E(A, \overline{A})| \geq |E|/2$. Consider the algorithmic problem of deterministically finding such an A. Exhaustive search would of course take exponential time. But notice that there are $|V|$ 0-1 random variables in the randomized algorithm defined above, namely the random variable X_u for each $u \in V$, where X_u takes the value 1 if u is put in A and otherwise it takes the value 0. Notice that for proving the claim in the above exercise we need only pairwise independence of the set of random variables $\{X_u \mid u \in V\}$, because that suffices to guarantee that the probability of an edge being part of the cut induced by A is $1/2$.

Assume without loss of generality that $|V| = 2^s$ and consider the 2-universal hash family (defined in an exercise above) $\mathcal{H} : V \mapsto \{0,1\}$, where we choose the fixed function $g : V \mapsto \{0,1\}$ to be the (mod 2) function. The size of the entire class $\mathcal{H}$ is $|V|^2$. Thus a random $h \in_R \mathcal{H}$ requires $2 \log |V|$ random bits to define it. By the previous Exercise, it follows that $\{h(u) \mid u \in V\}$ is a pairwise independent set of 0-1 random variables, which can be used as the set $\{X_u \mid u \in V\}$ in the above randomized algorithm without making any difference to the expected value of the cut induced by A.

Now, it is easy to derandomize the algorithm. Since $|\mathcal{H}| = |V|^2$ we can actually cycle through all $h \in \mathcal{H}$ and pick the corresponding A which yields the largest cut. The resulting deterministic algorithm is clearly polynomial time.

Exercise *Show how to carry out the derandomization when $|V|$ is not an exact power of 2.*

Pointers for further reading

The design of the first randomized algorithms is attributed to Berlekamp [5] and Rabin [26]. The concept of randomized algorithms was formalized (using probabilisitic Turing machines) and studied in [27, 12]. For a comprehensive treatment of randomized algorithms (including detailed bibliographic references) we refer the reader to the textbook of Motwani and Raghavan [23].

Carter and Wegman introduced universal hash functions in [8, 37] where they also construct the classes of universal hash functions described in this section.

We refer the reader to the section on hashing in chapter 8 of the textbook [23] for related material and exercises. Also, we suggest for further reading the notes of Wigderson [38].

For definitions and results on various complexity classes we refer the interested reader to standard text books [25, 4, 6]. The game-theoretic characterization of PH and PSPACE is from [9].

The first concrete application of pairwise independence to derandomizing randomized algorithms was due to Luby [21] where he designed an NC algorithm for the maximal independent set problem (see Kozen's textbook [19] for an excellent treatment). The example of the maxcut problem is from Wigderson's notes [38].

2 Probabilistic algorithms and the polynomial hierarchy

In this section, we demonstrate another application of 2-universal hash functions to derandomizing probabilistic algorithms. The method described is general enough to derandomize the entire class BPP (unfortunately, at the expense of moving up the polynomial hierarchy!). Specifically, we show that BPP is located in the second level of the polynomial hierarchy, i.e. BPP $\subseteq \Sigma_2^p$. In order to prove this we first need to amplify the success probability of a given BPP algorithm $\mathcal{A}$. Let $\mathcal{A}$ be a BPP algorithm with success probability $p > 3/4$.

1. Input x;
2. $s := 0$;
3. **for** $i := 1$ **to** $2t + 1$ **do**
4.   Simulate $\mathcal{A}$ on x;
5.   **if** $\mathcal{A}(x) = 1$ **then** $s := s + 1$;
6. **endfor**;
7. **if** $s > t$ **then** ACCEPT **else** REJECT;

The failure probability of the above algorithm is clearly bounded by

$$\sum_{i=0}^{t} \binom{2t+1}{i} p^i (1-p)^{2t+1-i}$$

The following lemma [29] can be easily used to compute the amplification achieved by the above algorithm.

Lemma 2.1 *Let E be some event that occurs with probability $1/2 + a$, with $0 < a \leq 1/2$. Then E occurs in $2t + 1$ independent trials at least $t + 1$ times with probability at least $1 - 1/2(1 - 4a^2)^t$.*

Proof. Let q_i be the probability that E occurs exactly i times in $2t + 1$ independent trials. By the binomial distribution we have $q_i = \binom{2t+1}{i} \cdot (1/2 + a)^i (1/2 - a)^{2t+1-i}$. Since $a > 0$, the above expression is clearly bounded by $\binom{2t+1}{i} \cdot (1/2+a)^i (1/2-a)^{2t+1-i}((1/2+a)/(1/2-a))^{t-i}$. Regrouping the terms in this we get $\binom{2t+1}{i} \cdot (1/2 - a) \cdot (1/4 - a^2)^t$.

Now, the probability that E occurs more than t times is $1 - \sum_{i=0}^{t} q_i$ which is at least as large as $1 - \sum_{i=0}^{t} \binom{2t+1}{i} \cdot (1/4 - a^2)^t$. This expression is exactly $1 - 2^{2t} \cdot (1/4 - a^2)^t = 1 - (1 - 4a^2)^t$. ■

In order to compute the amplified success probability of the simulation we put $a = 1/4$ in the above lemma. The success probability is $1 - (3/4)^t$.

It is easy to see that for inputs of length n, if we wish our simulation to have a success probability of $1 - 2^{-q(n)}$, for a polynomial q, we can pick $t = p(n)$ for an appropriate polynomial $p(n)$. The overall running time of the simulation is still bounded by a polynomial: it contributes only a polynomial factor of $p(n)$ to the running time of $\mathcal{A}$ on length n inputs.

Now we proceed with showing that BPP is contained in Σ_2^p.

Let L be a set in BPP accepted by a BPP algorithm $\mathcal{A}$, with running time bounded by $p(n)$ and error probability bounded by $2^{-q(n)}$ on inputs of length n. More precisely,

$$x \in L \Rightarrow Prob_{w \in \Sigma^{p(|x|)}}[\mathcal{A}(w, x) = 1] \geq 1 - 2^{-q(n)}.$$

$$x \notin L \Rightarrow Prob_{w \in \Sigma^{p(|x|)}}[\mathcal{A}(w, x) = 1] \leq 2^{-q(n)}.$$

Let $W_x \subseteq \Sigma^{p(|x|)}$ denote the subset of rejecting random strings w for input x. Clearly, $x \in L$ iff $|W_x| \leq 2^{p(n)-q(n)}$ and $x \notin L$ iff $|W_x| \geq 2^{p(n)}(1 - 2^{-q(n)})$. Thus $|W_x|$ is either large or small. The key point in designing the Σ_2^p predicate for L is that it must somehow distinguish between the case when $|W_x|$ is large and the case when it is small.

Fix an input x of length n and let $r = p(n)$. We will choose m and consider a standard class of universal hash functions $\mathcal{H} : \Sigma^r \mapsto \Sigma^m$. Intuitively, by the pigeon-hole principle, if $|W_x|$ is larger than $|\Sigma^m| = 2^m$ then no function (in particular none in $\mathcal{H}$) can map W_x into Σ^m without collisions. On the other hand, if $|W_x|$ is small compared to 2^m then, as we shall see, there are functions in $\mathcal{H}$ (in fact many) that map W_x into Σ^m without collisions.

The class of hash functions $\mathcal{H} : \Sigma^r \mapsto \Sigma^m$ that we will use is the collection of all $r \times m$ matrices over $GF(2)$, which can be seen as linear transformations from $\Sigma^r \mapsto \Sigma^m$ (where Σ^r and Σ^m are viewed as r and m dimensional vector spaces respectively over $GF(2)$). We have seen in Section 1 that this is an almost 2-universal class (Lemma 1.3). We concentrate on the following specific properties of $\mathcal{H}$.

Definition 2.2 1. A *hash family* H of size s from $\mathcal{H}$ is a subset $H \subseteq \mathcal{H}$ with $|H| = s$.

2. A *hash family* $H = \{h_1, \cdots, h_s\} \subseteq \mathcal{H}$ hashes a set $W \subseteq \Sigma^r$ if for every $w \in W$ there is an $h_i \in H$ such that for each $w' \in W - \{w\}$ it holds that $h_i(w) \neq h_i(w')$.

Lemma 2.3 *A* hash family *H of size s picked from $\mathcal{H}$ can hash a subset W of Σ^r only if $|W| \leq s2^m$.*

Proof. Each $h_i \in H$ can hash at most 2^m different strings in Σ^r without collision. Thus H can hash W only if $|W| \leq s2^m$. ■

We now turn to a sufficient condition for the hashability of W.

Lemma 2.4 *Let $W \subseteq \Sigma^r$ such that $|W| \leq 2^{m-1}$. Then a hash family H of size m, picked randomly from $\mathcal{H}$, hashes W with probability at least $1/2$.*

Proof. Let $x, y \in \Sigma^r$ with $x \neq y$. Consider a single hash function h randomly picked from $\mathcal{H}$ (h is an $r \times m$ matrix over $GF(2)$). Clearly, $Prob_h[h(x) = h(y)] = Prob_h[h(x - y) = 0^m]$. Notice that $\{h \in \mathcal{H} \mid h(x - y) = 0^m\}$ is a subgroup of $\mathcal{H}$ and, furthermore, for each $z \in \Sigma^m$ there is an $h_z \in \mathcal{H}$ such that $h_z(x - y) = z$. Thus it follows that for each $z \in \Sigma^m$ the set $\{h \in \mathcal{H} \mid h(x - y) = z\}$ is a right coset of the subgroup $\{h \in \mathcal{H} \mid h(x-y) = 0^m\}$. Consequently, $\{h \in \mathcal{H} \mid h(x-y) = z\}$ and $\{h \in \mathcal{H} \mid h(x-y) = 0^m\}$ have the same size. Putting it together, it means that a random hash function $h \in \mathcal{H}$ is equally likely to map $x - y$ to any element $z \in \Sigma^m$. Therefore, it follows that $Prob_h[h(x) = h(y)] = 2^{-m}$.

Call h bad for x if there is $y \in W - \{x\}$ such that $h(x) = h(y)$. Call H bad for x if every $h \in H$ is bad for x.

Given x, $Prob_h[h \text{ is bad for } x] = Prob_h[\exists y \in W : h(x) = h(y)]$ which is bounded by $|W|/2^m$ (by the union law of probabilities). Since $|W| \leq 2^{m-1}$, $Prob_h[h \text{ is bad for } x] \leq 1/2$. Therefore, $Prob_H[H \text{ is bad for } x]$ is bounded by $1/2^{|H|}$ which is $1/2^m$. By definition, H fails to hash W if H is bad for some $x \in W$. Therefore, again using the union law for probabilities, we finally have,

$Prob[H \text{ fails to hash } W] \leq |W|/2^m \leq 1/2$.

Therefore a random hash family H of size m hashes *any* subset $W \subseteq \Sigma^r$ such that $|W| \leq 2^{m-1}$ with probability at least $1/2$. ■

Turning back to the BPP set L, notice that if $x \in L$ then the set W_x is such that $|W_x| \leq 2^{r-q(n)}$. Thus, if we choose the value of the parameter m in Lemma 2.4 as $m = 1 + r - q(n)$, there will exist hash families H of size m from the class $\mathcal{H}$ (consisting of $r \times m$ matrices over $GF(2)$) that can hash W_x.

On the other hand, if $x \notin L$, then $|W_x| \geq (1 - 2^{-q(n)})2^r > 2^{r-1}$. Since $m2^m < 2^{r-1}$ for a suitable polynomial $q(n)$, by Lemma 2.3 W_x cannot be hashed by any hash family of size m from $\mathcal{H}$.

It follows that $x \in L$ iff the following Σ_4^p predicate is true: $(\exists H)(\forall u \in W_x)(\exists h_i \in H)(\forall v \in W_x - \{u\})[h_i(u) \neq h_i(v)]$.

Observe that the third quantifier $\exists h_i \in H$ has a polynomial range since $|H| = m$ is polynomially bounded. This can be absorbed into the next universal quantifier giving us a Σ_2^p predicate for the BPP set L.

Thus BPP $\subseteq \Sigma_2^p$.

Exercise *Work out the details of converting the Σ_4^p predicate into a Σ_2^p predicate, following the hint given in the above proof.*

Pointers for further reading

The containment of BPP in Σ_2^p was shown by Sipser in [33]. The proof in these notes is that of [33]. An alternative, simpler proof (not based on universal hashing) of the same containment appears in [20]. More recently, as a corollary of an entirely different gènre of results in [24], this containment has been improved to show BPP $\subseteq$ ZPP$^{\text{NP}}$.

Lemma 2.1 is due to Schöning [29]. The usual method to prove that the majority vote amplifies the success probability of a BPP algorithm is to use Chernoff bounds to estimate the tail probability of the sum of independent Bernoulli trials (see e.g. [23] for a detailed treatment). These bounds are tighter than the one given by Lemma 2.1 but their derivations are harder.

3 Recycling random bits

We saw in the last section how to amplify the success probability of a probabilistic algorithm $\mathcal{A}$ from a constant (say $3/4$) to $1 - 2^{-\Omega(k)}$. Given an input, the algorithm $\mathcal{A}$ was run $O(k)$ times, using *independent* random strings for each run, and finally the outcome was decided with the majority vote. If the algorithm $\mathcal{A}$ requires r random bits, then the above simulation with amplified success probability requires $O(rk)$ random bits. This simulation makes the error probability exponentially small in k, using $O(rk)$ random bits.

The *probability amplification* problem is that of achieving this error probability while using the minimum possible number of random bits. This problem is not only theoretically interesting but it also makes practical sense because it is difficult to design computational random sources that can be used by randomized algorithms. To produce 'random' bits, a computer might, for example, use a pseudo-random generator or resort to another source like using the last digits of a real-time clock. In any case, it is not clear how random these generated bits will actually be. How do we take these issues into consideration in the design of randomized algorithms? Clearly, one aspect is to consider the number of random bits also as a computational resource for randomized algorithms, apart from the usual resources of time and space.

Another issue of interest is to see if weak or imperfect sources of randomness can be used in randomized algorithms. This is now an active area of research. Theoretical models of imperfect computational random sources have been developed and studied [39, 34].[1]

We will discuss one solution to the amplification problem. The following intuitive argument indicates that such a saving of random bits is possible. A single execution of the randomized decision procedure $\mathcal{A}$ uses r random bits and delivers one bit of information (accepting or rejecting the input); it is plausible that somehow $r-1$ of the random bits are still available for further executions of the algorithm. Following this argument, one could expect that k repeated executions of $\mathcal{A}$ can be done with no more than $r+k-1$ random bits. We describe a method based on universal hashing, due to [17], which achieves something reasonably close; namely, it achieves the amplification to $1-2^{-\Omega(k)}$ with $O(r+k^2)$ random bits.

The main difficulty in any method of 'recycling' already used random bits is to make sure that the statistical behavior of the majority vote random variable undergoes no significant change. This will guarantee that the analysis for the majority vote's correctness still goes through (with only minor changes).

The solution of Impagliazzo and Zuckerman can be intuitively described as follows: in the first execution, algorithm $\mathcal{A}$ uses the string w_1 of r random bits. Then a suitable randomly picked hash function h is applied to w_1 to output $h(w_1)$ which is $s<r$ bits. A string of $r-s$ 'new' random bits is appended to $h(w_1)$ to yield the string w_2, which is used in the next execution of $\mathcal{A}$. This process is then repeated. At the ith execution, a string $w_i = h(w_{i-1})z$ is used, where $h(w_{i-1})$ is s bits long and z is a string of $r-s$ new random bits. Thus, the k executions of $\mathcal{A}$ to be used for the amplification use the 'pseudo-random' sequence $w_1, w_2, \cdots, w_k$.

Let us first bound the number of new random bits used by the above procedure. For the first execution algorithm $\mathcal{A}$ uses r random bits. For each subsequent iteration, $\mathcal{A}$ uses $r-s$ new random bits. In addition, some random bits are required to pick h from a suitable universal hash family. We shall see in the construction that in all only $O(r)$ random bits are required for picking the hash functions. Thus totally $r+O(r)+(r-s)\cdot(k-1)$ random bits are required for the k executions. It will turn out in the analysis that follows that we can choose $r-s$ to be $O(k)$, thus implying that $O(r+k^2)$ random bits suffice for amplifying the success probability of $\mathcal{A}$ to the desired level. This is a big improvement over the naïve $O(rk)$ random bit implementation we discussed in the previous section.

We now proceed to define the class of hash functions that we will use. In fact, we will use the hash family defined in Lemma 1.2 of Section 1.

Let the key and address spaces be $\{0,1\}^r$ and $\{0,1\}^s$ respectively. By Bertrand's postulate we can choose a prime p such that $2^r \le p \le 2^{r+1}$ bits (so that $p \ge 2^r$). We will work over the field

[1] At the end of this section we give more pointers to related work.

$Z_p = \{0, 1, \cdots, p-1\}$. Fix $g : Z_p \mapsto N$ to be the function given by $g(x) = x \pmod{2^s}$. For all $a, b \in Z_p$ define the hash function $h_{a,b} : Z_p \mapsto N$ as $h_{a,b}(x) = g(ax + b \pmod{p})$.

The hash family we use is $\mathcal{H} = \{h_{a,b} \mid a, b \in Z_p, a \neq 0\}$ which we proved in Lemma 1.2 to be almost 2-universal. [2]

Definition 3.1 Let D, D' be two probability distributions on a finite set S. We say that D and D' are *ϵ-close* if for every subset $X \subseteq S$ it holds that $|D(X) - D'(X)| \leq \epsilon$.

Here $D(X)$ denote the probability of X under the distribution D, i.e. $D(X) = \sum_{x \in X} Pr_D(x)$.

Exercise *Given probability distributions D, D' on S, which event X maximizes $|D(X) - D'(X)|$?*

A distribution D on a finite set S is called *ϵ-uniform* if D and the uniform distribution S are ϵ-close.

Definition 3.2 The collision probability of a distribution D on a finite set S is the probability of getting the same element twice in two independent draws from S according to distribution D.

Lemma 3.3 *("Leftover Hash Lemma") Let X be any subset of $\{0,1\}^r$ with $|X| \geq 2^l$. Let $\mathcal{H}$ be an almost 2-universal class of hash functions from $\{0,1\}^r$ to $\{0,1\}^s$. Suppose $r \geq l > s$, and let $e = (l - s)/2$. Then the distribution of $\langle h, h(x) \rangle$ in $\mathcal{H} \times \{0,1\}^s$ is 2^{-e}-uniform, where $h \in_R \mathcal{H}$ and $x \in_R X$.*

Proof. We prove the lemma in two parts.

Claim 3.3.1 *The collision probability of the distribution $\langle h, h(x) \rangle$ in $\mathcal{H} \times \{0,1\}^s$ is bounded by $(1 + 1/2^{2e})/(|\mathcal{H}|.2^s)$.*

Proof of Claim 3.3.1. Let $h_1, h_2 \in_R \mathcal{H}$ and $x_1, x_2 \in_R X$ be picked uniformly and independently at random from $\mathcal{H}$ and X respectively. We have to bound $Prob[h_1 = h_2 \wedge h_1(x_1) = h_2(x_2)]$ from above. This is precisely $(1/|\mathcal{H}|) \cdot Prob[h(x_1) = h(x_2)]$, where $h \in_R \mathcal{H}$ and $x_1, x_2 \in X$ are picked uniformly, independently at random. If $x_1 \neq x_2$, then it follows from the fact that $\mathcal{H}$ is almost universal that $Prob[h(x_1) = h(x_2)]$ is bounded by $1/2^s$. But the probability that $x_1 = x_2$ is $1/|X|$ and since $|X| \geq 2^l$, this probability is bounded above by $1/2^l$. Putting it all together

$$\begin{aligned} Prob[h_1 = h_2 \wedge h_1(x_1) = h_2(x_2)] &= 1/|\mathcal{H}| \cdot Prob[h(x_1) = h(x_2)] \\ &\leq 1/|\mathcal{H}| \cdot (1/2^s + 1/2^l) \\ &\leq (1 + 1/2^{2e}) \cdot (1/(|\mathcal{H}|.2^s)). \end{aligned}$$

□

Claim 3.3.2 *Let D be a probability distribution on a finite set S. If the collision probability of D is bounded by $(1 + 2\delta^2)/|S|$ then D is a δ-uniform distribution on the set S.*

Proof of Claim 3.3.2. Suppose D is not δ-uniform on the set S. Then there is a subset Y of S such that $D(Y) > |Y|/|S| + \delta$. Let $\beta > \delta$ such that $D(Y) = |Y|/|S| + \beta$. Consider the collision probability $Prob[d_1 = d_2]$ for two independent draws d_1, d_2 from the set S according to distribution D. Let us compute with respect to the subset Y as follows. The probability that $d_1 = d_2$ under the condition that $d_1, d_2 \in Y$ is at least $1/|Y|$ (*Exercise: use simple calculus to prove this*). Similarly,

[2] Strictly speaking, since choosing a prime in the required range is not easy, we should be using the 2-universal hash family defined in Section 1.3. Work out the details using this particular family.

the probability that $d_1 = d_2$ under the condition that $d_1, d_2 \in S - Y$ is at least $1/(|S| - |Y|)$. Thus the collision probability $Prob[d_1 = d_2]$ is at least

$$D(Y)^2/|Y| + (1 - D(Y))^2/(|S| - |Y|)$$

We get $1/|S| + \beta^2/|Y| + \beta^2/(|S| - |Y|)$ after substituting $D(Y) = |Y|/|S| + \beta$ and simplifying. This expression takes its minimum value for $|Y| = |S|/2$ and thus the collision probability is at least $(1 + 4\beta^2)/|S| > (1 + 4\delta^2)/|S|$. This contradicts the assumed bound on the collision probability. □

The lemma follows directly from the two claims. ■

Now, consider the algorithm $\mathcal{A}$ whose success probability we wish to amplify. For technical reasons we impose upper and lower bounds on both the success and failure probability for $\mathcal{A}$.

- $x \in L \Rightarrow 2/3 \leq Pr[\mathcal{A}(x) = 1] \leq 3/4$.
- $x \in L \Rightarrow 1/4 \leq Pr[\mathcal{A}(x) = 0] \leq 1/3$.
- $x \notin L \Rightarrow 1/4 \leq Pr[\mathcal{A}(x) = 1] \leq 1/3$.
- $x \notin L \Rightarrow 2/3 \leq Pr[\mathcal{A}(x) = 0] \leq 3/4$.

Exercise *Prove that any BPP algorithm $\mathcal{A}$ can be modified so as to satisfy the above bounds on its failure and success probabilities.*

Let $w_1, \cdots, w_k$ be the sequence of pseudo-random strings generated using the method described above. We wish to estimate accurately the amplified success probability when $\mathcal{A}$ is run k times using the sequence $w_1, \cdots, w_k$ and the majority vote is used for deciding membership of the input. Let us compare this with the naïve simulation $\mathcal{A}$ with fresh random strings $y_1, \cdots, y_k$ for k runs followed by majority vote at the end. The distributions of $w_1, \cdots, w_k$ and $y_1, \cdots, y_k$ in $\{0,1\}^{rk}$ are very different. $y_1, \cdots, y_k$ is uniformly distributed in $\{0,1\}^{rk}$ whereas $w_1, \cdots, w_k$ is certainly not uniformly distributed. Indeed, $w_1, \cdots, w_k$ does not seem to be even close to uniformly distributed. The reason is that each pseudo-random sequence $w_1, \cdots, w_k$ can be described[3] using $O(r + k^2)$ bits, whereas more than half the strings in $\{0,1\}^{rk}$ require at least $rk - 1$ bits to describe them!

But then we do not need to compare the distributions of $w_1, \cdots, w_k$ and $y_1, \cdots, y_k$. What is important is to compare $\mathcal{A}$'s outputs in the two simulations. Let $b(w_1)b(w_2)\cdots, b(w_k)$ be the k-bit sequence of $\mathcal{A}$'s decisions for the k runs using $w_1, \cdots, w_k$, and let $b_1 b_2 \cdots b_k$ be $\mathcal{A}$'s decisions for the k runs using $y_1, \cdots, y_k$ for the k runs. Let us do the analysis for a particular input $x \in L$ (for $x \notin L$ the analysis is symmetric). Let $Prob[b_i = 1] = q$ for each $i = 1, 2, \cdots, k$. It holds from our assumption that $2/3 \leq q \leq 3/4$.

Consider the distributions $b_1 b_2 \cdots b_k$ and $b(w_1)b(w_2)\cdots, b(w_k)$ in $\{0,1\}^k$. We know that $b_1 b_2 \cdots b_k$ as $\mathcal{A}$'s output sequence followed by majority vote amplifies $\mathcal{A}$'s success probability to $1 - 2^{-\Omega(k)}$. We would like to estimate how small ϵ should be so that if $b(w_1)b(w_2)\cdots, b(w_k)$ and $b_1 b_2 \cdots b_k$ are ϵ-close distributions, then a majority vote on $b(w_1)b(w_2)\cdots, b(w_k)$ also has success probability $1 - 2^{-\Omega(k)}$.

Let $X = \{z \in \{0,1\}^k \mid$ there are at most $k/2$ 1's in $z\}$. The set X is the set of all the bad outcome sequences for the majority vote decision for algorithm $\mathcal{A}$. Let D and E be the distributions defined by $b(w_1)b(w_2)\cdots, b(w_k)$ and $b_1 b_2 \cdots b_k$ on $\{0,1\}^k$ respectively. If D and E are ϵ-close then $Prob_D(X) \leq Prob_E(X) + \epsilon = \epsilon + \sum_{i=o}^{k/2} p^i (1-p)^{k-i} \leq \epsilon + 2^{-\Omega(k)}$.

[3] We're talking here of descriptions in the sense of Kolmogorov complexity.

Thus, ϵ should be $2^{-\Omega(k)}$ in order that $b(w_1)b(w_2)\cdots, b(w_k)$ bounds the failure probability of $\mathcal{A}$ to $2^{-\Omega(k)}$.

Finally, we proceed to show that the distribution D is indeed $2^{-\Omega(k)}$-close to E. We need the following inductive claim to establish that.

Claim 3.3.3 *$\forall i \geq 0$ the distribution D_i defined by the string $b(w_1)b(w_2)\cdots b(w_i)hw_{i+1}$ is ϵ_i-close to the distribution E_i defined by $b_1 b_2 \cdots b_i hv$, where $h \in_R \mathcal{H}$ and $v \in_R \{0,1\}^r$, and $b(w_1)b(w_2)\cdots b(w_i)$ and $b_1 b_2 \cdots b_i$ are as defined earlier. The constant ϵ_i will be appropriately chosen in the proof.*

Proof of Claim 3.3.3. We prove the claim by induction. For the case $i = 0$ notice that hw_1 is identically distributed as the distribution E_0. Thus we can choose $\epsilon_0 = 0$ in this case. For the case $i = 1$, the distribution D_1 is defined by $b(w_1)hw_2$. Recall that the last $r - s$ bits of w_2 are uniformly picked from $\{0,1\}^{r-s}$. Thus we only need to compare the distributions defined by $b(w_1)hh(w_1)$ and $b_1 hu$ where $h \in_R H$ and $u \in_R \{0,1\}^s$. Since w_1 is uniformly distributed in $\{0,1\}^r$ it follows that $b(w_1)$ and b_1 are identically distributed.

Now, given a specific value $a \in \{0,1\}$ taken by $b(w_1)$, we have to show that $hh(w_1)$ is ϵ_1-close to the uniform distribution hu in $H \times \{0,1\}^s$. But this is precisely what the leftover hash lemma asserts. Given $b(w_1) = a$, the set $b^{-1}(a)$ is of size $q2^r$ or $(1-q)2^r$ (depending on whether $a = 1$ or 0). In any case $|b^{-1}(a)| \geq 2^{r-2}$. Choosing $l = r - 2$ in the leftover hash lemma we get $\epsilon_1 = 2^{-(r-s-2)/2}$.

Now for the induction step. Assume that D_{i-1} is ϵ_{i-1}-close to E_{i-1}. We need to show that D_i is ϵ_i-close to E_i. Since the last $r - s$ bits in the strings describing both D_i and E_i are uniformly distributed, it suffices to show that $b(w_1)b(w_2)\cdots b(w_i)hh(w_i)$ is ϵ_i-close to $b_1 b_2 \cdots b_i hu$, where $h \in_R H$ and $u \in_R \{0,1\}^s$.

The following sub-claim is useful.

Subclaim *Let V_1, V_2, V_3 be random variables with their ranges being $\{0,1\}^*$. Suppose the distributions of V_1 and V_2 are δ_1-close. Further, let t be a function so that $t(V_1)$ and V_3 are δ_2-closely distributed. Then $t(V_2)$ and V_3 are $\delta_1 + \delta_2$-closely distributed.*

To see this let $X \subseteq \{0,1\}^*$. Then

$$
\begin{aligned}
|Prob[t(V_2) \in X] - Prob[V_3 \in X]| \\
&\leq |Prob[t(V_2) \in X] - Prob[t(V_1) \in X]| + |Prob[t(V_1) \in X] - Prob[V_3 \in X]| \\
&\leq |Prob[V_2 \in t^{-1}(X)] - Prob[V_1 \in t^{-1}(X)]| + |Prob[t(V_1) \in X] - Prob[V_3 \in X]| \\
&\leq \delta_1 + \delta_2.
\end{aligned}
$$

For the induction step set $V_1 = b_1 b_2 \cdots b_{i-1} hz$, $V_2 = b(w_1)\cdots b(w_{i-1})hw_i$, and $V_3 = b_1 b_2 \cdots b_{i-1} b_i hv$, where $z \in_R \Sigma^r$.

Let the transformation t be defined as $t(a_1 a_2 \cdots a_{i-1} hy) = a_1 a_2 \cdots a_{i-1} b(y) hh(y)$.

By induction assumption V_1 and V_2 are ϵ_{i-1}-closely distributed. $t(V_1)$ and V_3 are ϵ_1-closely distributed by the case for $i = 1$. Consequently we have $t(V_2)$ and V_3 are $\epsilon_i = \epsilon_{i-1} + \epsilon_1$ closely distributed. The claim follows. □

The claim asserts that for $i = k$, $b(w_1)\cdots b(w_k)$ and $b_1 \cdots b_k$ are $k\epsilon_1$-closely distributed. Notice that $k\epsilon_1 = k.2^{-(r-s-2)/2}$. In order to make $k\epsilon_1$ equal to $2^{-\Omega(k)}$, we can set $s = r - 2k - 2$.

This completes the proof.

Pointers for further reading

The problem of saving random bits while achieving amplification in randomized algorithms has been studied in several papers. Chor and Goldreich [10] show how to achieve inverse polynomial amplification using pairwise independence with just doubling the number of random bits used. The amplification result which we have discussed in these notes is due to Impagliazzo and Zuckerman [17]. Our description is based on expositions in Wigderson's notes [38] and Schöning's book [31]. Using expander graphs a similar result is proved in [11].

The problem of simulating BPP algorithms with weak random sources was initiated and studied in [28, 35, 36]. More recently, the results of these papers have been substantially improved in [39, 34].

4 Randomization and Interaction

Recall that the polynomial hierarchy is built up by setting up a Prover-Challenger game, with a polynomial time verifier acting as referee. There is another way in which NP can be generalized. Suppose we dispense with the Challenger altogether, and view NP as a setup where the Prover talks directly to the Verifier. What happens if we allow interaction between these two agents? The Prover sends a message, the Verifier asks a question (possibly based on the previous message), the Prover sends an answer and so on. If the input is in the language, then no matter what questions the Verifier asks, the Prover's replies are always convincing. And if the input is not in the language, then no matter what replies the Prover sends, the Verifier remains unconvinced. Has this generalization achieved anything?

It is easy to see that as long as the Verfier is a deterministic polynomial time algorithm, this makes no difference to the language accepted. The Prover is all-powerful; the moment he sends the first message, he can figure out what the next question will be, and answer that too in advance. So the whole interaction can be brought down to one round, which is NP.

However, increasingly the notion has found favour that not just polynomial time, but even probabilistic polynomial time (BPP, RP) may be viewed as efficient computation. So now it makes sense to look at how interaction adds to the power of a BPP verifier. What does this achieve? Consider interaction between a Prover and a Randomized Verifier. Define rounds as we did in the Prover-Challenger game, except that here there is no Challenger; the Verifier himself participates in the interaction. What's more, the Verifier accepts statistically overwhelming evidence, and does not insist on absolutely correct proofs.

So now we can talk of classes P-RV[k] and RV-P[k]. Clearly, P-RV[1] includes NP = P-V[1], since the verifier can always choose to ignore random bits. But does it include more? And is it true that for every k, P-RV[k] contains P-V[k]?

Actually, surprising things happen once we include randomization. Let us first specify a little more clearly what we mean by a P-RV interaction scheme. These schemes are usually called protocols, or games. Traditionally, complexity theorists call the Prover Merlin and the Randomized Verifier Arthur; Merlin was the wizard in King Arthur's court. (Following our own history, we could settle for an exceedingly clever if not all-powerful Prover Birbal, and a diligent Verifier Akbar.) So they are also called AM games, or interactive proofs (IP), rather than P-RV interaction schemes. However, we will stick to our P-RV notation for a while; the reason will become clear later.

A language L is in P-RV[k] if there is an interaction scheme, consisting of k messages being passed between the Verifier and the Prover, with the following properties:

- The first message is from the Verifier to the Prover.
- The Verifier performs polynomial time randomized computation between messages.

- Whenever $x \in L$, the Prover has a strategy for answering the Verifier's questions so that no matter what questions the Verifier asks, he ends up accepting x with probability at least 3/4.
- Whenever $x \notin L$, no matter what strategy the Prover uses to answer the Verifier's questions, the Verifier ends up rejecting x with probability at least 3/4.

Exercise *Show that the probability of producing a correct answer can be amplified, so we can safely assume that the error is exponentially small ($2^{-q(|x|)}$ for some polynomial q).*
(Repeat the entire game some number of times. But this increases the number of rounds. Avoid this by doing the repetition in parallel.)

As an example, consider Graph Non-Isomorphism GNI, the complement of the GI problem we defined earlier. GNI is in co-NP, but we do not know of any method of providing short witnesses to non-isomorphism. Since co-NP is Π_1^p, GNI has a C-P[1] scheme. We will now show that it also has a RV-P[2] scheme.

Input Graphs G_1 and G_2 over n vertices.

Verifier's algorithm Choose a permutation $\sigma \in S_n$ randomly. Choose a bit $b \in \{1, 2\}$ randomly. Construct the graph $H = \sigma(G_b)$. Send Merlin the following message:

Question Here is a graph H. Did I produce it by transforming G_1, or G_2?

Answer from Prover A bit b'.

Verifier's result If $b = b'$, then output YES, else output NO.

Let us see why this works. Suppose the graphs are not isomorphic. Then, if H was produced from G_1, it cannot be also produced from G_2, since otherwise we would have demonstrated a transformation from G_1 via H to G_2. Similarly if H was produced from G_2, it cannot possibly be produced from G_1. In other words, H comes from a unique graph. So the Prover can figure out which of G_1 and G_2 is the unique graph, and always return the correct answer b.

On the other hand, suppose the graphs are isomorphic. Then an H produced from any one of the input graphs can equally well be produced from the other input graph. Even if the Prover is exceedingly clever, he cannot possibly tell, merely by looking at H and the input graphs, where H came from. So any answer he gives will be correct on no more than half the instances.

So the game defined above has this property: the game accepts YES instances with probability 1, and accepts NO instances with probability at most half.

This is almost what we require. The error on NO instances is not yet bounded away from half, but that is easily corrected; the Verifier repeats the whole process with a new permutation, and asks one more question. He outputs YES only if both of the Prover's answers were correct. It is easy to see that on NO instances, the error probability is now at most 1/4. For YES instances, there is still no error. So we have a one-sided error version of an RV-P game. By repeating the process $p(n)$ times, we can even bring down the error probability to an inverse exponential.

But now we seem to have used $2 \times p(n)$ rounds, 2 per question. Can the number of rounds be reduced? Easily, as the following exercise shows.

Exercise *Show that GNI is in RV-P[2], with inverse exponential error probability. (Hint: Note that the questions in different rounds are independent. Instead of posing them in successive rounds, the Verifier could pose them all together.)*

What are the important resources in such a game? Intuitively, a problem that can be solved in two rounds is at least as easy as one which seems to need four rounds. So number of rounds is a resource. Other obvious resources are the running time and the space used by verifier. But there is a different type of resource, less obvious: randomization. It raises two issues. Firstly, how much randomization (or how many random bits) is required to solve the problem? In the example of GNI, we needed enough random bits to be able to choose $p(n)$ permutations uniformly at random. The previous section has already touched upon this issue. Secondly, what kind of random bits? Publicly visible ones, or bits hidden from the prover? Does it make any difference? Well, going back to our game for GNI, we can see that it is crucial! If the Prover could see the random bits used by the Verifier, then he will send back those bits as answers, and will never have made a mistake. The Verifier will be unable to distinguish between YES and NO instances. So, the random bits used must be *private*.

Games using private random bits are usually called interactive proof (IP) systems. Only games using public random bits are called Arthur-Merlin games. (Random bits in this context are often referred to as coins or coin tosses.) Now, at last, we can abandon the P-RV and RV-P notation! Henceforth, RV-P[k] and P-RV[k] with private coins will be referred to as IP[k], RV-P[k] with public coins will be referred to as AM[k], and P-RV[k] with public coins will be referred to as MA[k]. (A word of caution: this notation is not completely standardized. Some authors prefer to call a back-and-forth exchange of messages one round. Then RV-P[$2k$] will be either IP[k] or AM[k], and P-RV[$2k$] will be either IP[k] or MA[k]. Fortunately, as we shall see, this will not matter, as long as $k \geq 2$.) Obviously, AM[k] $\subseteq$ IP[k], for every k. Formally, these classes are defined as follows.

Definition 4.1 A language L is in IP[$2g(n)$] if there are functions P and V such that:

- V maps tuples of strings to strings or to a {YES,NO} outcome. A tuple $\langle w, r, x_1, y_1, \ldots, x_i, y_i \rangle$, for $i < g(|w|)$, is mapped by V to the next question x_{i+1}, and tuples $\langle w, r, x_1, y_1, \ldots, x_{g(|w|)}, y_{g(|w|)} \rangle$ are mapped to YES or NO. The function, on any such tuple $\langle w, r, x_1, y_1, \ldots, x_i, y_i \rangle$, can be computed in time polynomial in $|w|$.

- P maps tuples of strings to strings. A tuple $\langle w, x_1, y_1, \ldots, x_i \rangle$ is mapped by P to the answer y_i. The function need not even be computable. Note that the function P does not have the string r as an argument.

- For every $w \in \Sigma^n$:

$$w \in L \implies Prob_r[\mathrm{V}(w, r, x_1, y_1, \ldots, x_{g(n)}, y_{g(n)}) = \mathrm{YES}] > 3/4,$$
$$w \notin L \implies \forall \mathrm{P}' : Prob_r[\mathrm{V}(w, r, x_1, y_1', \ldots, x_{g(n)}, y_{g(n)}') = \mathrm{YES}] < 1/4$$

Definition 4.2 A language L is in AM[$2g(n)$] if there are functions P and V such that:

- P maps tuples of strings to strings. A tuple $\langle w, x_1, \ldots, x_i \rangle$ is mapped by P to y_i. (Again, the function need not even be computable.)

- V is a probabilistic machine with running time bounded by a polynomial in $|w|$.

- For every $w \in \Sigma^n$:

$$w \in L \implies Prob_{\langle x_1, \ldots, x_{g(n)}, r \rangle}[V(w, x_1, y_1, \ldots, x_{g(n)}, y_{g(n)}, r) = \mathrm{YES}] > 3/4,$$
$$w \notin L \implies \forall P' : Prob_{\langle x_1, \ldots, x_{g(n)}, r \rangle}[V(w, x_1, y_1', \ldots, x_{g(n)}, y_{g(n)}', r) = \mathrm{YES}] < 1/4$$

(We may view the strings $x_1, \ldots, x_{g(n)}$ as random strings chosen by the Verifier. Since all coin tosses are supposed to be public, the Verifier then sends these strings to the prover. The only non-trivial computation V does is at the end of $g(n)$ rounds.)

Our scheme above showed that GNI is in IP[2]. The question is, is it also in AM[2]?

It turns out that there is a public-coin two round protocol for GNI, so it is in AM[2] after all. This protocol uses linear hash functions in a manner that exploits the nature of the problem. But there is also a more general result, of which we shall see a weaker version in the next section, stating that any problem in IP[k] is also in AM[k]. Counter-intuitive though it seems, evidently the Verifier does not gain any additional advantage by performing computations in secret from the Prover. The ability (or inability) of the Prover to fool him into making wrong answers remains unchanged.

Here is how we design an AM[2] game for GNI. First, we must count automorphisms in a graph. An automorphism of a graph G is a permutation $\pi \in S_n$ such that permuting the vertices of G according to π yields the graph G itself; $\pi(G) = G$. Let AUT(G) denote the set of automorphisms of G, and let #AUT(G) denote the number of distinct automorphisms of the graph G. Now define the following sets, for $b \in \{1, 2\}$

$$\begin{array}{rcl} S_b & = & \{\langle X, \pi\rangle \mid X \equiv G_b \wedge \pi \in \mathrm{AUT}(G_b)\} \\ S & = & S_1 \cup S_2 \\ S^q & = & S \times S \times \ldots \times S \quad (q \text{ times }) \end{array}$$

It is easy to see that the number of distinct X's in S_b is is precisely $n!/\#\mathrm{AUT}(G_b)$. Hence, $|S_b| = n!$. Also, if G_1 and G_2 are not isomorphic, then S_1 and S_2 are disjoint, so $|S| = 2 \times n!$. But if G_1 is isomorphic to G_2, then $S_1 = S_2$, so $|S| = n!$. So $|S^q|$ is $(n!)^q$ if $G_1 \equiv G_2$, and $2^q(n!)^q$ otherwise. This enormous difference in the set size (for polynomial q, the difference is an exponential factor) is what our public coin verifier exploits.

We also use the fact that S_b, and hence S and S^q, are in NP; strings in S_b have short proofs of membership. For $\langle X, \pi\rangle$, a proof of membership would be a permutation τ such that $X = \tau(G_b)$. A polynomial time verifier can check this, given τ, and can also check that $\pi(G_b) = G_b$.

We use the 2-universal hash functions defined in Lemma 1.3. The tuples of S^q can be encoded as strings over $\{0, 1\}$, of length r. We will decide our choice of q later; once it is fixed, let r be the corresponding length required of a string to encode an element of S^q. We also defer our choice of m. Consider the class $\mathcal{H}$ of linear hash functions from Σ^r to Σ^m. We use the results of Lemma 2.3 and Lemma 2.4.

Here is how the protocol proceeds.

Input Graphs G_1 and G_2 over n vertices.

Arthur's algorithm Randomly choose a hash family H of m hash functions $h_1, \ldots, h_m$ from $\{0, 1\}^r$ to $\{0, 1\}^m$, and send it to Merlin with the following question.

Question Supply a counter-example showing that H fails to hash S^q.

Answer from Merlin Strings u, $v_1, \ldots v_m$ in Σ^r. Also, for u and for each v_i, string w, $z_i, \ldots, z_m$ respectively, which is a proof of membership in S^q.

Arthur's result If $h_i(u) = h_i(v_i)$ for $1 \leq i \leq m$, and if w witnesses $u \in S^q$, and if for each $1 \leq k \leq m$, z_i witnesses $v_i \in S^q$, then accept. Otherwise reject.

Arthur chooses any polynomial valued q, m satisfying $1 + q\lceil\log(n!)\rceil \le m$ and $m + \log m \le q + q\lfloor\log(n!)\rfloor$. If $G_1 \not\equiv G_2$, then $|S^q| > m2^m$, so with probability 1, H fails to hash S^q. So Merlin can find an element $u \in S^q$, and elements $u_1, \ldots, u_m$, such that for each $h_i \in H$, $h_i(u) = h_i(u_i)$. (Each h_i is bad for u, because of a collision with u_i. So H is bad for u.) If $G_1 \equiv G_2$, then $|S^q| \le 2^{m-2}$. So the probability that H fails to hash S^q is at most half, and if H hashes S^q, Merlin cannot find any u for which H is bad.

By repeating this whole process twice, in parallel, we can make the error probability less then 1/4.

This protocol can be further simplified. Instead of choosing a family of hash functions H, suppose Arthur chooses a single function h. (The advantage is that fewer random bits are required.) For $u \neq 0^r$, we say that h destroys u if $h(u) = 0^m$. From Lemma 1.3, a randomly chosen h destroys u with probability $1/2^m$. Now Arthur asks Merlin for an element of S^q that is destroyed by h. We can show that if S^q is large, such an element exists with high probability, and if S^q is small, such an element is unlikely to exist.

To formalize these quantities, we need the notion of expectation and variance of a random variable, and the well-known Chebyshev's inequality.

Definition 4.3 If X is a discrete random variable taking values from a set S, then the expected value of X, denoted $E[X]$ or μ_X, is $\sum_{i\in S} i Prob[X = i]$, and the variance of X, denoted var(X) or σ_X^2, is $E[(X - \mu_X)^2]$.

Proposition 4.4 [Chebyshev's inequality] *For $k > 0$, $Prob\left[|X - \mu_X| \ge k|\sigma_X|\right] \le 1/k^2$*

We also need the following property of our class of hash functions.

Exercise *If $A \subseteq \Sigma^r$, and $A_h = \{u \in A \mid h(u) = 0^m\}$, show that*

1. *$E[|A_h|] = |A|/2^m$, and*
2. *$var[|A_h|] = 2^{-m}(1 - 2^{-m})|A|$.*

Exercise *Show that if Arthur chooses any polynomial valued q, m satisfying $2 + q\lceil\log(n!)\rceil \le m \le q + q\lfloor\log(n!)\rfloor$, the protocol described below for GNI will work.*

Input *Graphs G_1 and G_2 over n vertices.*

Arthur's algorithm *Randomly choose a hash function h from $\{0,1\}^r$ to $\{0,1\}^m$, and send it to Merlin.*

Question *Is there an element in S^q that h destroys?*

Answer from Merlin *A string u in Σ^r, along with w, a proof of membership of u in S^q.*

Arthur's result *If $h(u) = 0^k$, and if w witnesses $u \in S^q$, then accept. Otherwise reject.*

Exercise *Show that $MA \subseteq AM$. (Use success probability amplification as described in Section 2.)*

Exercise *Strengthen the $BPP \subseteq \Sigma_2^p$ result to show that MA is contained in Σ_2^p.*
Strengthen the $BPP \subseteq \Pi_2^p$ result to show that AM is in Π_2^p.

We introduce more notation, but fortunately, no significant new classes; the new notation just allows a different description of the classes we have already seen. Let $\mathcal{C}$ be a complexity class. Then

$L \in \exists.\mathcal{C}$ if there is a polynomial p and a language $A \in \mathcal{C}$ such that

$x \in L \Leftrightarrow \exists y : |y| \leq p(|x|), \langle x, y\rangle \in A$

$L \in \forall.\mathcal{C}$ if there is a polynomial p and a language $A \in \mathcal{C}$ such that

$x \in L \Leftrightarrow \forall y : |y| \leq p(|x|), \langle x, y\rangle \in A$

$L \in BP.\mathcal{C}$ if there is a polynomial p and a language $A \in \mathcal{C}$ such that

$Prob_y[x \in L \Leftrightarrow \langle x, y\rangle \in A]$

$L \in R.\mathcal{C}$ if there is a polynomial p and a language $A \in \mathcal{C}$ such that

$x \in L \Longrightarrow Prob_y[\langle x, y\rangle \in A] \geq 3/4$

$x \notin L \Longrightarrow Prob[\langle x, y\rangle \in A] = 0$

where y is chosen uniformly and at random from strings of length upto $p(|x|)$.

Now it is easy to see that NP = $\exists$.P, $\Sigma^p_{k+1} = \exists.\Pi^p_k$, $\Pi^p_{k+1} = \forall.\Sigma^p_k$, and so on. Also, BPP = BP.P, $\exists$.BPP $\subseteq$ MA, AM = BP.NP.

Exercise *Show that AM is contained in co-R.NP.*

This means that for languages in AM, the error can always be made one-sided. When $x \in L$, Merlin always succeeds in convincing Arthur; when $x \notin L$, he may still succed but with small probability. Notice that our AM protocol for GNI is already in this form.

Another general result shows that MA$[k+1]$ = AM$[k]$ = AM$[2]$ for $k \geq 2$. That is, as long as the number of rounds is not related to the input size (k is a constant), two rounds always suffice. This is in stark contrast to what happens with alternation; the levels of the polynomial hierarchy are believed to be distinct, while the levels of the interactive proof hierarchy coincide. And these coincided levels lie in Π^p_2. Does this mean that against a powerful Prover, a Randomized Verifier is no substitute for a powerful Challenger and a deterministic Verifier?

A surprising result proved a few years ago shows that in general this is not the case. Private and public coin protocols which used polynomially many rounds were constructed to accept problems in PSPACE (recall that this is precisely the class APTIME, where there are polynomially many rounds between the Prover and the Challenger). What this means is that with polynomial rounds of interaction, a Randomized Verifier can take on the role of an all-powerful Challenger. The containment in the reverse direction is easy to see; AM[poly] and IP[poly] (usually denoted just IP, the polynomial bound on number of rounds is assumed) can be simulated in PSPACE by a machine which tries out all strategies that the Prover could possibly use. So we have the result IP = PSPACE.

Pointers for further reading

Interactive proof systems were first introduced by Goldwasser, Micali and Rackoff [14]. Their motivation was studying zero-knowledge proofs in the context of cryptography. At the same time, Babai [1] independently introduced the seemingly weaker formulation of Arthur-Merlin games, in order to capture the complexity of some group-theoretic problems. Goldwasser and Sipser [16] then showed that these public and private coin models are actually equivalent in computational power.

Babai [1] showed that MA is in AM and that the MA and AM hierarchies collapse. Babai and Moran [2] showed that MA is in Σ^p_2 and AM is in Π^p_2. Goldreich, Mansour and Sipser [13] showed that AM can be modified to have one-sided error; AM is in co-R.NP. The simulation of PSPACE in IP is due to Shamir [32], building on techniques developed by Lund, Fortnow, Karloff and Nisan [22] to show that PH, and more, is in IP.

The private coin protocol for graph non-isomorphism was first proposed by Goldwasser, Micali and Wigderson [15]. The existence of a public coin protocol follows from the results of [1] and [16]. Our public coin protocol for GNI is presented in [30]. The simplifed form in the Exercise, using a single hash function rather than a hash family, follows the proof presented in the book by Köbler, Schöning, and Torán [18]. Another simplified version is shown by Boppana, Hastad and Zachos [7].

For the proof of Chebyshev's inequality and related results concerning random variables, the reader is referred to [23] or any standard text on probability theory.

5 Private vs Public coins: IP and AM

In this section, we generalize the idea of the previous section; using hash functions, we show how to convert private coins to public coins. To keep notation clearer, we will refer to a round here as an exchange of messages rather than a single message. We show how to simulate a g-round private-coin protocol IP$[g]$ by a $(g+2)$-round public-coin protocol AM$[g+2]$. We will refer to the prover and verifier of the IP system as P and V, and of the AM system as M and A.

On input w, V reads a random string r. It then generates a message x_1, gets a reply y_1, generates another message x_2, gets a reply y_2 and so on. After g rounds, V decides whether to accpet or reject, based on $w, r, x_1, y_1, \ldots, x_g, y_g$. Since the behaviour of the Verifier is determined by w, r and the Prover's responses, we will talk of input $\langle w, r\rangle$ to a deterministic Verifier.

Without loss of generality we can assume that all messages exchanged are of the same length, say m. Let the length of the random string used by V be l. We assume that the IP protocol has already been modified to have exponentially small error probability.

We use the class $\mathcal{H}$ of linear functions over $GF(2)$; these form an almost 2-universal class of hash functions, as described in Lemma 1.3. Let the key and address spaces be Σ^k and Σ^b respectively; then a hash function $h \in \mathcal{H}$ is a $k \times b$ Boolean matrix D, and $h_D(x) = xD$ for $x \in \Sigma^k$. If $H = \{h_1, \ldots, h_l\} \subseteq \mathcal{H}$, $C \subseteq \Sigma^k$, and $D \subseteq \Sigma^b$, then $h_i(C) = \{y \in \Sigma^b \mid \exists x \in C, h_i(x) = y\}$, $H(C) = \cup_{i=1}^{l} h_i(C)$, and $h_i^{-1}(D) = \{x \in \Sigma^k \mid \exists y \in D, h_i(x) = y\}$, and $H^{-1}(D) = \cup_{i=1}^{l} h_i^{-1}(D)$.

We need the following properties of this class.

Lemma 5.1 [Approximate Lower Bound Lemma] *Given $b, k, l > 0$, $l > max(b, 8)$, and $C \subseteq \Sigma^k$. Randomly select a hash family of size l from the almost 2-universal class of linear functions; $H = \{h_1, \ldots, h_l\} \subseteq \mathcal{H}$. Randomly select a subset of l^2 strings from the address space; $Z = \{z_1, \ldots, z_{l^2}\} \subseteq \Sigma^b$. Then*

1. *If $4|C| \le 2^b \le 8|C|$ then $Pr[H(C) \cap Z \ne \emptyset] \ge 1 - 2^{-l/8}$*
2. *If for $d > 0$, $|C| \le 2^b/d$, then $Pr[H(C) \cap Z \ne \emptyset] \le l^3/d$*

Proof.

1. Let us first interpret this statement. The restriction on b tells us that C is of size comparable to the address space. So if a set Z of some l^2 strings is chosen randomly from the address space, then it is very unlikely that l randomly chosen hash functions all avoid Z while mapping C.

   Let us prove that this is indeed unlikely. Firstly, note that since $\mathcal{H}$ is almost 2-universal, for any x, y, $Prob_h[h(x) = h(y)] \le 1/2^b$. Adapting the argument in Lemma 2.4, for a fixed x,
$$\begin{array}{rlcl} Prob[h_i \text{ is bad for } x] = & Prob[\exists y \in C\ x \ne y \wedge h_i(x) = h_i(y)] & \le & |C|/2^b \\ & & \le & 1/4 \\ Prob[H \text{ is bad for } x] = & Prob[\forall i \le l, \exists y \in C\ x \ne y \wedge h_i(x) = h_i(y)] & \le & 1/4^l \\ Prob[H \text{ is bad for some } x] = & Prob[\exists x \forall i \le l, \exists y \in C\ x \ne y \wedge h_i(x) = h_i(y)] & \le & |C|/4^l \\ & \text{(since } l > b) & \le & 1/2^l. \end{array}$$

So with probability at least $1-1/2^l$, H successfully hashes C. But in this case, $H(C)$ cannot be very small; in fact it must have at least $|C|/l$ elements. (Why? Each $h \in H$ accounts for at most $|h(C)| \leq |H(C)|$ elements. So H can account for at most $l|H(C)|$ elements. Since H hashes every element in C, $|C| \leq l|H(C)|$. This argument refines that in Lemma 2.3.)

Now, assuming that H hashes C, $|H(C)| \geq |C|/l \geq 2^b/(8l)$. A randomly drawn $z \in \Sigma^b$ thus lies outside $H(C)$ with probability at most $1 - |H(C)|/2^b \leq 1 - 1/(8l)$. So all strings of Z avoid $H(C)$ with probability at most $(1-1/(8l))^{l^2}$. Finally,

$$\begin{aligned}
Prob[H(C) \cap Z = \emptyset] &= Prob[H(C) \cap Z = \emptyset \mid H \text{ hashes } C]Prob[H \text{ hashes } C] + \\
&\quad Prob[H(C) \cap Z = \emptyset \mid H \text{ fails to hash } C]Prob[H \text{ fails to hash } C] \\
&\leq Prob[H(C) \cap Z = \emptyset \mid H \text{ hashes } C] + Prob[H \text{ fails to hash } C] \\
&\leq (1-1/(8l))^{l^2} + 2^{-l} \\
&< 2^{-l/8}
\end{aligned}$$

2. Again, let us first interpret this statement. If C is smaller than the address space by a factor of d, then $H(C)$ will also be small. Hence a hash family of size l, while mapping C, is very likely (probability at least $1 - l^3/d$) to completely avoid a random set of l^2 keys.

   Formally, since $|H(C)| \leq l|C|$ and $2^b \geq d|C|$, a randomly drawn $z \in \Sigma^b$ is in $H(C)$ with probability at most $|H(C)|/2^b \leq (l|C|)/(d|C|) = l/d$. So some string in Z lies in $H(C)$ with probability at most $(l/d)|Z| = l^3/d$.

■

The New Protocol

Round 0: M sends A a number b_1.

Round i ($1 \leq i \leq g$): A sends M a set H_i of l randomly chosen hash functions from Σ^m to Σ^{b_i}. A also sends a randomly chosen subset Z_i of Σ^{b_i} of size l^2. M responds with strings x_i, y_i in Σ^m and a number b_{i+1}. If $x \notin H_i^{-1}(Z_i)$, A rejects immediately.

Round $g+1$: A sends M a set H_{g+1} of l randomly chosen hash functions from Σ^l to $\Sigma^{b_{g+1}}$. A also sends a randomly chosen subset Z_{g+1} of $\Sigma^{b_{g+1}}$ of size l^2. M responds with string r in Σ^l. If $r \notin H_{g+1}^{-1}(Z_{g+1})$, A rejects immediately.

Final Verification. 1. A verifies that for $1 \leq i \leq g$, $V(w, r, x_1, y_i, \ldots, x_{i-1}, y_{i-1}) = x_i$.

2. A verifies that $V(w, r, x_1, y_i, \ldots, x_g, y_g) =$YES.
3. A also verifies that $b_1 + \ldots + b_{g+1} \geq l - g\log l$.

If all these conditions are satisfied, A accepts.

End of Protocol

Merlin's Winning Strategy when $w \in L$:

Let B_0 be the set of random strings on which the protocol, with the best strategy of P, leads to acceptance. We can assume that $|B_0| \geq (2/3)2^l$.

Merlin's strategy is as follows:

Round 0: Put the random strings in B_0 into boxes, each box labelled by the first message x_1 that V generates, given any random string in that box. Now put these boxes into l sacks, so that boxes of roughly the same size go into the same sack. Specifically, sack $S_i^{(0)}$ has boxes of sizes

$(2^{i-1}, 2^i]$. (Exception: $S_1^{(0)}$ has boxes of sizes $[1, 2]$.) Now choose the heaviest sack; call it S^0. Viewing this sack as a subset of B_0, clearly $|S^0| \geq |B_0|/l$. Let S^0 have k_0 boxes in it. k_0 is thus the number of distinct labels on boxes in the chosen sack. Set $b_1 = 2 + \lceil \log k_0 \rceil$.

Round i $(1 \leq i < g)$**:** A sends H_i, Z_i. H_i maps message strings (from Σ^m) to strings of length b_i, and Z_i is a set of l^2 strings of length b_i. By the Approximate Lower Bound Lemma, with high probability (at least $1 - 2^{-l/8}$), some box label from the sack S^{i-1} is in $H_i^{-1}(Z_i)$. Choose any such label, say x_i. Send x_i, and $y_i = \mathrm{P}(w, x_1, y_1, \ldots, x_i)$.

x_i fixes a particular box $B_i = B_{x_1,y_1,\ldots,x_{i-1},y_{i-1}}(x_i)$ in the sack S^{i-1}. Sack S^{i-1} contains all random strings r for which, on input $\langle w, r \rangle$, V generates x_1, and if the response is y_1, generates x_2, and so on upto x_{i-1}. Box B_i is that subset of S^{i-1} for which, with the next response y_{i-1}, V generates message x_i.

Repeat the partitioning on strings in box B_i: put these strings into smaller boxes labelled by the $(i+1)$th message V generates on these strings, given the message stream $x_1, y_1, \ldots, x_i, y_i$. Put these boxes into l sacks $S_1^{(i)}, \ldots S_l^{(i)}$ based on size, and choose the heaviest sack, call it S^i. If S^i has k_i boxes, set $b_{i+1} = 2 + \lceil \log k_i \rceil$.

Round g**:** Select x_g, y_g as described for the previous rounds. This now fixes a particular box $B_g = B_{x_1,y_1,\ldots,x_{g-1},y_{g-1}}(x_g)$ in sack S^{g-1}. All strings in this box are consistent with message stream $x_1, y_1, \ldots, x_g, y_g$ and lead V to accept w. Let k_g be the size of this box. Set $b_{g+1} = 2 + \lceil \log k_g \rceil$.

Round $g+1$**:** A sends H_{g+1}, Z_{g+1}. H_{g+1} maps random strings (from Σ^l) to strings of length b_{g+1}, and Z_{g+1} is a set of l^2 strings of length b_{g+1}. By the Approximate Lower Bound Lemma, with high probability (at least $1 - 2^{-l/8}$), some string from the box B_g is in $H_{g+1}^{-1}(Z_{g+1})$. Choose any such string, say r, and send it to Arthur.

Analysis:

Merlin can fail in the following three ways:

1. At some stage i, Merlin cannot find a string in $H_i^{-1}(Z_i)$. Given the way Merlin chooses b_i, this happens with probability at most $(g+1)2^{-l/8}$ (probability at most $2^{-l/8}$ at any one stage), which is exponentially small.

2. The message stream $(x_1, y_1, \ldots, x_g, y_g)$ and random string r is not consistent with an accepting P-V interaction. Since Merlin chooses all responses based on the set of accepting random strings B_0, this situation will not arise.

3. The sum $b_1 + \ldots b_{g+1}$ is too small. We show below that this does not happen with the above strategy.

For each $0 \leq i < g$, consider sacks $S_1^{(i)}, \ldots, S_l^{(i)}$. These sacks partition the box B_i, so the heaviest sack S^i has weight at least $|B_i|/l$.

In sack S^i, there are k_i boxes, of roughly the same size (the smallest and the largest differ in size by a factor of at most 2), and one of these is the box B_{i+1} chosen at the next round. It is easy to show that $|S^i| \leq 2k_i|B_{i+1}|$. Also, $|B_g| = k_g$.

Since $b_{i+1} = 2 + \lceil \log k_i \rceil$, $2^{b_{i+1}} \geq 4k_i$.

Putting this together, we have $|B_i| \leq l|S^i| \leq lk_i|B_{i+1}| \leq l2^{b_{i+1}}|B_{i+1}|$.

So $|B_0| \leq l^g 2^{b_1+b_2+\ldots+b_g}|B_g| \leq l^g 2^{b_1+b_2+\ldots+b_g} 2^{b_{g+1}}$. But $|B_0|$ was a large set to begin with, of size at least $(2/3)2^l$. So $(2/3)2^l \leq l^g 2^{b_1+b_2+\ldots+b_{g+1}}$, and taking logarithms on both sides, we find that $b_1 + b_2 + \ldots + b_{g+1} \geq l - g \log l$, so Arthur's test succeeds.

What can a cheating Merlin do?

Consider now the case when w is not in L. The best that Merlin can do is to base his responses on the best possible response from any cheating Prover to V's queries. Note that for any prover, the set $|B_0|$ is small; we assume it is at most $2^l/l^{12g^2}$.

Let M send some b_1 in Round 0. For the purposes of analysis, we partition B_0, as the honest Merlin had done above, into boxes and sacks. We call a box *big* if it has size at least $|B_0|/(2^{b_1}/d)$, where d is a parameter we will fix later. Clearly, there can be at most $2^{b_1}/d$ big boxes. Consider the labels attached to these big boxes. By the Approximate Lower Bound Lemma, the probability that there is a big box label in $H_1^{-1}(Z_1)$ is at most l^3/d. Whatever x_1 Merlin sends in Round 1 implicitly defines a box, and the probability that this box is big is thus at most l^3/d.

At Round $1 < i \leq g$, let the message stream constructed so far be $x_1, y_1, \ldots, x_{i-1}, y_{i-1}$. So we (we doing the analysis, not Merlin) are looking at a box B_{i-1} of random strings consistent with this stream, and we can now partition it into smaller boxes and sacks. At this stage, we say that a box is big if it has size at least $|B_{i-1}|/(2^{b_i}/d)$. Using the same argument as before, the probability that the box B_i chosen at this round is big is at most l^3/d.

At Round $g+1$, we are looking at a box B_g. We call this box *good* if it is small, of size at most $2^{b_{g+1}}/d$, but still has some string in $H_{g+1}^{-1}(Z_{g+1})$; otherwise it is *bad*. Again, this can happen with probability at most l^3/d.

Thus the probability that Merlin chooses a big box at any stage or a good box at the last stage is at most $(g+1)l^3/d$. We choose d so that this is $1/3$; that is, $d = 3(g+1)l^3$.

Assume that no such thing happens. All boxes chosen (implicitly) by Merlin are small, and the last box is bad. It can be bad in two ways, either because it is big, or because it has no string in $H_{g+1}^{-1}(Z_{g+1})$. But in the latter case, Merlin definitely fails. So let us assume optimistically (for Merlin) that the last box is big. Stringing together all inequalities,

$$2^l/l^{12g^2} \geq |B_0| \geq |B_1|(2^{b_1}/d) \geq |B_2|(2^{b_1+b_2}/d^2) \geq \ldots \geq |B_g|(2^{b_1+\ldots+b_g}/d^g) \geq 2^{b_1+\ldots+b_{g+1}}/d^{g+1}.$$

Taking logarithms on both sides, we see that $b_1 + \ldots + b_g \leq (g+1)\log d + l - 12g^2 \log l \leq l - g\log l$, so Merlin fails Arthur's last test.

Key Idea: The crucial idea used in the above argument is that if a hash family H is to map some element of a small set C into a randomly chosen subset Z of the range, then the range itself must be small. Since there are very few accepting random strings when $w \notin L$, Merlin can find messages x_i which a random H_i maps into a random subset Z_i only if b_{i-1} were chosen to be sufficiently small. But this is precisely the situation which Arthur's last test is designed to catch.

On the other hand, if the input is to be accepted, then the set of accepting random strings is large. Since at each stage i Merlin chooses b_{i+1} based on the weight of the heaviest sack, most b_i's are reasonably large.

IP[1] to AM[3]: The idea can be made more explicit by studying how a 1-round IP protocol gets converted to a 3-round AM protocol. If $w \in L$, then Merlin's strategy is to consider the set fo accepting strings B_0, partition them into boxes and sacks based on V's first (and only) message x_1, and choose the heaviest sack S^0. If this sack has k_0 boxes in it, $b_1 = 2 + \lceil \log k_0 \rceil$. So if H_1 and Z_1 are chosen randomly, then with high probability some label from S^0 is mapped to Z_1. Choose this label x, send x and P's response y to x, and send $b_2 = 2 + \lceil \log k_1 \rceil$, where k_1 is the size of the box labelled x. Now, if H_2 and Z_2 are chosen randomly, then with high probability some string in $\mathrm{Box}(x)$ is mapped to Z_2. Choose such a string r. Arthur will pass the first two verification tests trivially, and the third too because

$$
\begin{aligned}
b_1 + b_2 &= 4 + \log(\text{number of boxes in chosen sack}) + \log(\text{size of chosen box}). \\
\text{So } 2^{b_1+b_2} &= 16\,(\text{number of boxes in chosen sack})\,(\text{size of chosen box}). \\
\text{But (size of chosen sack)} &\le 2(\text{number of boxes in chosen sack})(\text{size of chosen box}). \\
\text{So } 2^{b_1+b_2} &\ge 8(\text{size of chosen sack}) \\
&\ge 8|B_0|/l, \text{ since we chose the heaviest of } l \text{ sacks that partitioned } B_0.
\end{aligned}
$$

Since $|B_0| \geq (2/3)2^l$, $b_1 + b_2$ will satisfy the required inequality. Note that it is quite possible for one of b_1 and b_2 to be very small. For instance, if all boxes are of size 1 or 2 and hence go into sack $S_1^{(0)}$, then whatever box is chosen will be small and so k_1 and b_2 will be small. But to compensate, b_1 will be large, more than half the size of B_0.

On the other hand, if $w \notin L$, let M send some b_1. Once b_1 is chosen, we can partition B_0 into boxes and mark some boxes as big relative to b_1 (big means at least $|B_0|/(2^{b_1}/d)$). Since B_0 itself is small, there are very few big boxes. Once Arthur sends a randomly chosen H_1 and Z_1, we throw away all boxes whose labels are not mapped by H_1 to Z_1. Since there were few big boxes to begin with, very likely no big box survives this "throwing out", in which case any string x Merlin sends can only be the label of a small box.

Let M choose some x, y and b_2, and receive a randomly chosen H_2 and Z_2. Now we consider whether the box corresponding to x is big relative to b_2 (i.e. of size at least $2^{b_2}/d$). If it is not, then it is very unlikely that Merlin will find any string in this box that H_2 maps to Z_2. On the other hand, if Merlin sends a string not in this box, he loses at step (1) in the final verification. So to avoid this, once Merlin chooses x, he must choose a b_2 such that box(x) is big relative to it.

But, we have seen above that box(x) is very likely small relative to b_1. And now we see that it must be big relative to b_2. So $|B_0|/(2^{b_1}/d) \geq |B(x)| \geq 2^{b_2}/d$, and Merlin will lose at step (2) in the final verification.

Pointers for further reading

Goldwasser and Sipser [16] were the first to show that public and private coin models are actually equivalent in computational power. Our proof directly follows their presentation. Subsequently the result has been improved; Babai and Moran [3] show that IP$[q(n)]$ can be simulated in AM$[\lceil q(n)/2 \rceil]$.

The Approximate Lower Bound Lemma is due to Sipser [33]. We state and prove it here following the presentation from [16].

Acknowledgment

The first author would like to thank K. V. Subrahmanyam for enjoyable discussions on "recycling random bits".

References

[1] L. Babai. Trading group theory for randomness. *17th ACM Symp. Theory of Computing 1985*, 421–429.

[2] L. Babai, S. Moran. Arthur-Merlin games: a randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences* **36**:254–276, 1988.

[3] L. Babai, S. Moran. Proving properties of interactive proofs by a generalized counting technique. *Information and Computation* **82**:185–197, 1989.

[4] J. Balcázar, J. Díaz, J. Gabarró *Structural Complexity I and II* EATCS Monographs 11 and 22, Springer-Verlag, 1988 and 1990.

[5] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation,* 24:713–735, 1970.

[6] D. P. Bovet, P. Crescenzi. *Introduction to the Theory of Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[7] R. Boppana, J. Hastad, S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters,* 25:127–132, 1987.

[8] J. L. Carter, M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences,* 18(2): 143–154, 1979.

[9] A. Chandra, D. Kozen, L. Stockmeyer. Alternation. *Journal of the ACM,* 28:114–133, 1981.

[10] B. Chor, O. Goldreich. On the power of two-point sampling. *Journal of Complexity,* 5:96–106, 1989.

[11] A. Cohen, A. Wigderson. Dispersers, deterministic amplification, and weak random sources. *30th Annual Symposium on Foundations of Computer Science,* 14–19, 1989.

[12] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing,* 6(4):675–695, 1977.

[13] S. Goldwasser, Y. Mansour, M. Sipser. Interactive Proof Systems: Provers that never fail and random selection. *28th IEEE Symp. Foundations of Computer Science 1987, 449–461.*

[14] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing* **18**:186–208, 1989.

[15] S. Goldwasser, S. Micali, A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *27th IEEE Symp. Foundations of Computer Science 1986, 174–187.*

[16] S. Goldwasser, M. Sipser. Private coins versus public coins in interactive proof systems. In: S. Micali (ed.): *Randomness and Computation*, Vol. 5 of *Advances in Computing Research*, JAI Press, 1989. (Also in Proc. 18th ACM Symp. on Theory of Computing STOC 86, 59–68.)

[17] R. Impagliazzo, D. Zuckerman. How to recycle random bits. *30th Annual Symposium on Foundations of Computer Science,* 248–253, 1989.

[18] J. Köbler, U. Schöning, J. Torán. *The graph isomorphism problem: Its structural complexity.* Birkhauser, Berlin, 1993.

[19] D. Kozen. *Design and Analysis of Algorithms.* EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1991.

[20] C. Lautemann. BPP and the polynomial hierarchy. *Information Proc. Letters,* 17:215–218, 1983.

[21] M. Luby. A simple parallel algorithm for the maximal independent set. *SIAM Journal on Computing,* 15:1036–1053, 1986.

[22] C. Lund, L. Fortnow, H. Karloff, N. Nisan. Algebraic methods for interactive proof systems. Journal of ACM **39**(4):859–868, 1992.

[23] R. Motwani, P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[24] N. Nisan, A. Wigderson. Hardness versus randomness. *31st Annual Symposium on Foundations of Computer Science,* 2–12, 1988.

[25] C. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[26] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity, Recent Results and New Directions,* 21–39, Academic Press, NY, 1976.

[27] M. O. Rabin. Probabilistic automata. *Information and Control,* 6:230–245, 1963.

[28] M. Santha, U. Vazirani. Generating quasi-random sequences from slightly random sources. *Journal of Computer and System Sciences,* 33:75–87, 1986.

[29] U. Schöning. *Complexity and Structure.* Lecture Notes in Computer Science, vol. 211, Springer-Verlag, 1985.

[30] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences,* 37:312–323, 1988.

[31] U. Schöning. *Perlen der Theoretische Informatik.* B-I Wissenschaftsverlag, 1995.

[32] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. (Also in Proc. 31st IEEE Symp. Foundations of Computer Science FOCS 1990, pp 11–15.)

[33] M. Sipser. A complexity theory approach to randomness. *Proc. 15th ACM Symposium Theory of Computing*, 330–335, 1983.

[34] A. Srinivasan, D. Zuckerman. Computing with very weak random sources. *35th Annual Symposium on Foundations of Computer Science,* 264–275, 1994.

[35] U. Vazirani. Efficiency considerations in using semi-random sources. *19th ACM STOC,* 1987.

[36] U. Vazirani, V. Vazirani. Random polynomial time is equal to semirandom polynomial time. *26th Annual Symposium on Foundations of Computer Science,* 417–428, 1985.

[37] M. N. Wegman, J. L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences,* 22(3):265–279, 1981.

[38] A. Wigderson. Lecture notes on the fusion method and derandomization. *Technical Report SOCS-95-2,* School of Computer Science, McGill University.

[39] D. Zuckerman. Simulating BPP using a general weak random source. *32nd Annual Symposium on Foundations of Computer Science,* 79–89, 1991.