Introduction to Complexity Theory*
Lecture 15: Derandomization of BPP

Notes taken by Erez Waisbard and Gera Weiss

March 16, 1999

In this lecture we present an efficient deterministic simulation of randomized algorithms. This process, called derandomization, introduce new notions of pseudorandom generators. We extend the definition of pseudorandom generators and show how to construct a generator that can be used for derandomization. The new construction differ from the generator that we constructed in the previous lecture in it's running time (it will run slower, but fast enough for the simulation), but most importantly in it's assumptions. We are not assuming the existence of one-way function but we make another assumption which may be weaker than that.

1 Introduction

Randomness plays a key role in many algorithms. However random sources are not always available and we would like to minimize the usage of randomness. A naive way to remove the "randomness element" from algorithm is simply to go over all possible coin tosses it uses and act according to the majority. This however we can't do for BPP in polynomial time with respect to the size of the input. If we could shrink the amount of randomness the algorithm consumes (to logarithmic) then we could imply the naive idea in polynomial time. A way to use small random source to create much more randomness was introduced in the previous lecture - the pseudorandom generator. Pseudorandom generator G stretch out random seed into a polynomial-long pseudorandom sequence that can't be efficiently (in polynomial time) distinguished from a truly random sequence.

$$G : \{0,1\}^k \to \{0,1\}^{poly(k)}$$

For convenience we reproduce here the formal definition we gave in the previous lecture for pseudorandom generator.

A deterministic polynomial-time algorithm G is called a pseudorandom generator if there exist a stretching function $l : N \to N$, so that for any probabilistic polynomial-time algorithm D, for any positive polynomial p, and for all sufficiently large k's

$$|Prob[D(G(U_k)) = 1] - Prob[D(U_{l(k)}) = 1]| < \frac{1}{p(k)}$$

Where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp., $U_{l(k)}$) as well as over the coin tosses of D.

Suppose we have such a pseudorandom generator then for every $\epsilon > 0$ we can shrink the amount of randomness used by an algorithm A, deciding a language in BPP, from $poly(n)$ to n^ϵ (where n

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

is the length of the input). The shrinking of randomness will not cause significance change in the behavior of the algorithm, meaning that it is infeasible to find a long enough input on which the algorithm which uses less randomness will decide differently than the original one. The problem is that the above doesn't indicate that there are no inputs for which A will act differently when using the pseudorandom source, but that they are hard to find. Thus in order to derandomize BPP we will need stronger notion of pseudorandom generator. We will need a pseudorandom generator
$\boxed{1}$ $\boxed{15.1}$ $\Rightarrow$ which can fool any small (polynomial size) circuit (and hence any polynomial time algorithm).

A deterministic polynomial-time algorithm G is called a non-uniformly strong pseudorandom generator if there exist a stretching function $l : N \to N$, so that for any family $\{C_k\}_{k \in L}$ of polynomial-size circuits, for any positive polynomial p, and for all sufficiently large k's

$$|Prob[C_k(G(U_k)) = 1] - Prob[C_k(U_{l(k)}) = 1]| < \frac{1}{p(k)}$$

$\boxed{1}$ $\boxed{15.2}$ $\Rightarrow$

If such G exist which is robust against polynomial-size circuit then

$$\forall \epsilon > 0 \;\; BPP \subseteq Dtime(2^{n^\epsilon})$$

$L \in BPP$ implies that the algorithm A for deciding L doesn't only take an input x with size n but also uses randomness R with size l (when we write for short $A(x)$ we really mean $A(x, R)$). The relation between the size of the input and the size of the randomness is $l = poly(n)$.
Let us construct a new algorithm A' which will use less randomness than A but will act similar to A' for almost all inputs.

$$A'(x, s)A(x, G(s))$$

Where $s \in \{0,1\}^{n^\epsilon}$.
A' uses little randomness and we claim that A' only differ from A for finitely many x's.

For all but finitely many x's

$$|Prob[A(x) = 1] - Prob[A'(x) = 1]| \; < \; \frac{1}{6}$$

The idea of the proof is that if there were infinitely many x's for which A and A' differ, then we could distinguish G's output from a random string, in contradiction to the assumption that G is a pseudorandom generator.

In order to contradict Definition **??** it suffices to present a family $\{C_k\}$ of small circuits for which

$$|Prob[C_k(G(U_k)) = 1] - Prob[C_k(U_{l(k)}) = 1]| \geq \frac{1}{6}$$

Suppose towards contradiction that for infinitely many x's A and A' behave differently, i.e

$$Prob[A(x) = 1] - Prob[A'(x) = 1] \geq \frac{1}{6}$$

Then we incorporate these inputs and A into a family of small circuits as follows

$$x \to C_x(\alpha)A(x, \alpha)$$

This will enable us to distinguish the output of the generator from a uniformly distributed source. The circuit C_k will be one of $\{C_x : A(x) \text{ uses } k \text{ coin tosses}\}$. Note that if there are infinitely many x's on which A and A' differ then there are infinitely many sizes of x's on which they differ. The amount of randomness used by the algorithm is polynomial with respect to the size of the input.

The idea behind this construction is that

$$C_k(G(U_k)) \equiv A'(x) \quad \textit{and} \quad C_k(U_{l(k)})) \equiv A(x)$$

Hence we have a family of circuits such that

$$|Prob[C_k(G(U_k)) = 1] - Prob[C_k(U_{l(k)}) = 1]| \geq \frac{1}{6}$$

Which is a contradiction to the definition of pseudorandom generator.

Saying that algorithm A decides BPP means that if $x \in L$ the probability that A will say $'YES'$ is greater than $\frac{2}{3}$ and if $x \notin L$ the probability that A will say $'YES'$ is smaller than $\frac{1}{3}$. By the above proposition, for all but finitely many x's $|Prob[A(x) = 1] - Prob[A'(x) = 1]| < 1/6$. Thus, for all but finitely many x's

$$x \in L \Rightarrow prob\ [A(x, U_n) = 1] \geq \frac{2}{3} \Rightarrow prob\ [A'(x, s) = 1] > \frac{1}{2}$$

$$x \notin L \Rightarrow prob\ [A(x, U_n) = 1] \leq \frac{1}{3} \Rightarrow prob\ [A'(x, s) = 1] < \frac{1}{2}$$

Now we define the algorithm A'' which incorporate these finitely many inputs, and for all other inputs it loops over all possible $s \in \{0,1\}^{n^\epsilon}$ (seeds of G) and decides by majority.

Algorithm A'': On input x proceed as follows.

```
if x is one of those finitely x's
    return the known answer
else
    for all s ∈ {0,1}^{n^ε}
        run A'(x, s)
    return the majority of A' answers
```

Clearly this A'' deterministicly decides L and run in time $2^{n^\epsilon} \cdot poly(n)$ as required.

2 New notion of Pseudorandom generator

The time needed for A'' to decide if an input x is in L or not was exponential in the length of it's seed s. For simulation purposes if the random seed is logarithmic in the size of the input, running the pseudorandom generator exponential time in the length of the seed is really running it polynomial time with respect to the length of the input x. Thus the time needed for simulating a randomized algorithm which run in polynomial time remains polynomial even if we allow the pseudorandom generator to run exponential time (with logarithmic size seed). In general, for the purpose of derandomizing BPP we may allow the generator to use exponential time. There seems to be no point insisting that the generator will use only polynomial time in the length of the seed when the running time of the simulating algorithm has an exponential factor in the size of the seed.

Motivated as above, we now introduce a new notion of pseudorandom generator with the following properties.

1. Indistinguishable by a polynomial-size circuit.
2. Can run in exponential time ($2^{O(k)}$ on k-bit seed).

The efficiency of the generator which was one of the key aspects is relaxed in the new notion. This new notion may seem a little unfair at first since we only give polynomial time to the distinguisher and allow the generator to run exponential time. It even suggests that if we give the seed as an extra input, polynomial size circuit wouldn't be able to take advantage of that because it wouldn't be able to evaluate the generator on the given seed. The relaxation allow us to construct pseudorandom generators under seemingly weaker conditions than the ones required for polynomial time pseudorandom generators (the assumption about existence of one-way functions).

3 Construction of non-iterative pseudorandom generator

The difference between the definition of pseudorandom generator that we introduced in this lecture and the definition of pseudorandom that we had before (the one usually used for cryptographic purposes) is that we allow the generator to run in time exponential in it's input size.

This difference enables us to construct a random generator under possibly weaker conditions without damaging the derandomization process. In this section we will demonstrate how to construct such a generator using "unpredictable predicate" and a structure called "design" (we will give a precise definition later). In the construction we use two main tools which we assume to exist. We assume the existence of such a predicate and the existence of a design, but later we will show how to construct such a design hence the only real assumption that we make is the existence of a predicate.

In the previous class we proved that pseudorandom generators exist if and only if one-way permutation exist.
We will show (in section **??** below) that this assumption is not stronger than the previous assumption, i.e the existence of one-way permutation implies the existence of a predicate (but not necessarily the opposite way). So it seems better to use the new assumption which may be true even if there exist no one-way function.

The previous construction uses a one-way permutation $f : \{0,1\}^l \to \{0,1\}^l$ in the following way:

> From a random string $S_0 = S$ (the seed), compute a sequence $\{S_i\}$ by $S_{i+1} = f(S_i)$. The random bits are then extracted from this sequence using a hard-core predicate. We proved that a small circuit that is not fooled by this bit sequence can be used to demonstrate that f is not a one-way permutation because it can be used to compute f^{-1}.

The construction that we will give here has a different nature. Instead of creating the bits sequentially we will generate them in parallel. Unpredictable predicates supply an easy way to construct one additional random looking bit from the seed. The problem is to generate more bits. We will do it by invoking the predicate on nearly disjoint subsets of the bits in the seed. This will give us bits which are nearly independent (such that no polynomial-size circuit will notice that they has some dependence).

3.1 Parameters

- k - Length of seed.
- m - Length of output (a polynomial in k).
- We want the generator to produce an output that is indistinguishable by any polynomial-size circuit (in k, or equivalently in m).

3.2 Tool 1: An unpredictable predicate

The first tool that we will need in order to construct a pseudorandom generator is a predicate that can not be approximated by polynomial sized circuits.

We say that an $exp(l)$-computable predicate $b : \{0,1\}^l \to \{0,1\}$ is unpredictable by small circuits (or simply unpredictable) if for every polynomial $p(\cdot)$, for all sufficiently large l's and for every circuit C of size $p(l)$:

$$Prob\left[C(U_l) = b(U_l)\right] < \frac{1}{2} + \frac{1}{p(l)}$$

This definition require that small circuits attempting to compute b have only negligible fraction of advantage over unbiased coin toss. This is a real assumption, because we don't know of a way to construct such a predicate (unlike the next tool that we will show how to construct later).

To evaluate the strength of our construction we prove, in the next claim, that the existence of unpredictable predicate is guaranteed if one-way permutation exist. The other way is not necessarily true, so it is possible that our construction can be applied (if someone will find a provable unpredictable predicate) while other constructions that depend on the existence of one-way permutations are not applicable (if someone will prove that such functions do not exist).

If f_0 is a one-way permutation and b_0 is a hard-core of f_0, then $b(x)b_0\left(f_0^{-1}(x)\right)$ is an unpredictable predicate.

We begin by noting that b is computable in exponential time because it takes exponential time to invert f and together with the computation of b_0, the total time is no more than exponential in the size of x.

The second property that we need to show is that it is impossible to predict b. In order to prove this property we use the variable $yf_0^{-1}(x)$ to get:

$$b\left(f_0(y)\right) = b_0(y)$$

Assume towards contradiction that b is predictable. This means that there exist an algorithm A and a polynomial $p(\cdot)$ such that for infinite number of n's :

$$Prob\left[A(U_n) = b(U_n)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

A is a polynomial-size algorithm which can predict b with a noticeable bias. But f is a permutation so we may write:

$$Prob\left[A\left(f_0(U_n)\right) = b\left(f_0(U_n)\right)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

Hence, from the definition of b we get:

$$Prob\left[A\left(f_0(U_n)\right) = b_0(U_n)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

which is a contradiction to the definition of b_0 as a hard core.

Recall that we demonstrated a generic hard-core predicate (the inner product *mod* 2) that is applicable to an arbitrary one-way permutation, hence essentially the last claim is only assuming the existence of one-way permutation (because the hard-core predicate can always be found). And we succeeded to show that the existence of unpredictable predicate may only be a weaker assumption than the existence of a one-way permutation. We did not prove that it is really a weaker assumption but we did show that it is not stronger. It may be the case that both assumptions are the same but we don't know of any proof for such a claim.

The assumption that we use to construct a generator is the existence of a "hard" predicate. The word "hard" means that the predicate can not be approximated by small circuits. The hardness of a predicate is measured by two parameters:

- The size of the circuit.
- The closeness of approximation.

In this notes we use polynomial-size circuits and polynomial approximation. Other papers demonstrated that similar process can be carried out with different conditions on this hardness parameters. In particular, the closeness of approximation parameter is greatly relaxed.

3.3 Tool 2: A design

The task of generating a single bit from a random bits seems easy if you have the predicate that we assumed to exist in the previous section because the output of the predicate must look random to every polynomial-size circuit. The problem is how can we generate more than one bit. We will do this using a collection of nearly disjoint sets to get random bits that are almost mutually independent (almost means indistinguishable by a polynomial-size circuits). To formalize this idea we introduce the notion of a design:

A collection of m subsets $\{I_1, I_2, ..., I_m\}$ of $\{1, ..., k\}$ is a (k,m,l)-design (or simply design) if the following three properties exist:

For every $i \in \{1, ..., m\}$,

$$|I_i| = l$$

For every $i \neq j \in \{1, ..., m\}$,

$$|I_i \cap I_j| = o(\log\ k)$$

The collection is constructible in $exp(k)$-time.

In our application the set $\{1, ..., k\}$ is all bit locations in the seed, and the subsets $\{I_1, I_2, ..., I_m\}$ correspond to different subsequences extracted from the seed. For example, one may look at a ten bit seed

$$S = \langle 1010010110 \rangle$$

The subset $I = \{1, 5, 7\} \subset \{1, ..., 10\}$ correspond to the first, the fifth and the seventh bits in the seeds which are, in this example :

$$S[I] = \langle 100 \rangle$$

In general, for $S = \langle \sigma_1 \sigma_2 \cdots \sigma_k \rangle$, and $I = \{i_1, ..., i_l\} \subset \{1, ..., k\}$ we use the notation:

$$S[I]\langle \sigma_{i_1} \sigma_{i_2} \cdots \sigma_{i_k} \rangle$$

3.4 The construction itself

We now want to construct a pseudorandom generator with a polynomial stretching function. The size of the seed will be k and the size of the output will be $m = m(k)$ which is a polynomial in k. Suppose we have a design $\{I_1, ..., I_m\}$ which consist of m subsets of the set $\{1, ..., k\}$. If these subset where completely disjoint then it is obvious (from Definition **??**) that for every unpredictable predicate b, the sequence $\langle b(S[I_1])\ b(S[I_2]) \cdots b(S[I_m])\rangle$ is unpredictable. Since unpredictability implies pseudorandomness we get pseudorandom generator. We will show that this is also true when the intersection of any two subsets is logarithmic in k and m (i.e, this is a design). Our generator will blow up seeds by applying the unpredictable predicate to every subsequence corresponding to
$\boxed{1}$ $\boxed{15.3}$ $\Rightarrow$ subsets in the design.

Let $b : \{0,1\}^{\sqrt{k}} \to \{0,1\}$ be unpredictable predicate, and $\{I_1, ..., I_m\}$ be a $(k, m, \sqrt{k})$ - design, then the following function is a pseudorandom generator (as defined in section **??**):

$$G(S)\langle b(S[I_1])\ b(S[I_2]) \cdots b(S[I_m])\rangle$$

Based on the definition of unpredictable predicate and the definition of a design it follows that it take no more than exponentially many steps to evaluate G.

We will show now that no small circuit can distinguish the output of G from a random sequence. Suppose towards contradiction that there exist a family of polynomial-size circuits $\{C_k\}_{k\in}$ and a polynomial $p(\cdot)$ such that for infinitely many k's

$$|Prob[C_k(G(U_k)) = 1] - Prob[C_k(U_m) = 1]| > \frac{1}{p(k)}$$

(This is the negation of G being a pseudorandom generator because the definition of pseudorandom generator demands that for every sufficiently large k's we have $|\cdots| < \frac{1}{p(k)}$ which imply that there are only finitely many k's with $|\cdots| > \frac{1}{p(k)}$)

We will assume that this expression is positive and remove the absolute value, because if we have infinitely many k's such that the above is true, we have that half of them have the same sign. We may take the sign as we want since we can always take other sequence of circuits with the reversed sign. So without loss of generality we assume that for infinitely many k's

$$Prob[C_k(G(U_k)) = 1] - Prob[C_k(U_m) = 1] > \frac{1}{p(k)}$$

For any $0 \le i \le m$ we define a "hybrid" distribution H_k^i on $\{0,1\}^m$ as follows: the first i bits are chosen to be the first i bits of $G(U_k)$ and the other $m - i$ bits are chosen uniformly at random :

$$H_k^i G(U_k)_{[1,...,i]} \circ U_{m-i}$$

In this notation we get $H_k^0 = U_m$ and $H_k^m = G(U_k)$. If we look at the function :

$$f_k(i) Prob[C_k(H_k^i) = 1]$$

We get that, since $f_k(m) - f_k(0) > \frac{1}{p(k)}$, there must be some $0 \le i_k \le m$ such that:

$$f_k(i_k + 1) - f_k(i_k) > \frac{1}{m} \cdot \frac{1}{p(k)}$$

So we know that there exist a circuit which behaves significantly different if the i_k'th bit is taken randomly or from the generator (the difference is greater than one over the polynomial $p'(k)m \cdot k$). That is:

$$Prob[C_k(H_k^{i_k+1}) = 1] - Prob[C_k(H_k^{i_k}) = 1] > \frac{1}{p'(k)}$$

We will use this circuit to construct another circuit which will be able to guess the next bit with some bias. On i_k bits of input and $m - i_k$ bits of random, the new circuit C_k' will behave as follows:

$$C_k'(\langle y_1, ..., y_{i_k}\rangle, \langle R_{i_k+1}, ..., R_m\rangle) \begin{cases} R_{i_k+1} & if\ C_k(\langle y_1, ..., y_{i_k}, R_{i_k+1}, ..., R_m\rangle) = 1 \\ 1 - R_{i_k+1} & otherwise \end{cases}$$

where $\langle y_1, ..., y_m\rangle G(U_k)$.

The idea behind this construction is that we know that the the probability that C_k will return 1 is significantly different whether R_{i_k+1} equals y_{i_k+1} or not. This is true because when R_{i_k+1} equals y_{i_k+1} we see that C_k is getting $H_k^{i_k+1}$ as input and otherwise it get $H_k^{i_k}$. The consequence of this fact is that we can use the answer of C_k to distinguish these two cases.

To analyze the behavior of C_k' more formally we look at two events (or Boolean variables):

$$\begin{array}{l} A\,(C_k(\langle y_1, ..., y_{i_k}, R_{i_k+1}, ..., R_m\rangle) = 1) \\ B\,(R_{i_k+1} = y_{i_k+1}) \end{array}$$

Notice that C_k' will return y_{i_k+1} in two distinct scenarios. The first scenario is when $R_{i_k+1} = y_{i_k+1}$ and C_k returns 1, and the second scenario is when $R_{i_k+1} \neq y_{i_k+1}$ and C_k returns 0. Using the above notation we get that:

$$Prob[C_k' = y_{i_k+1}] = Prob[B] \cdot Prob[A|B] + Prob[B^c] \cdot Prob[A^c|B^c]$$

Since

- $Prob[A] = f(i_k) = Prob[C_k(H_k^{i_k}) = 1]$
- $Prob[A|B] = f(i_k + 1) = Prob[C_k(H_k^{i_k+1}) = 1]$
- $Prob[B] = Prob[B^c] = \frac{1}{2}$
- $Prob[A] = Prob[B] \cdot Prob[A|B] + Prob[B^c] \cdot Prob[A|B^c]$

we get that:

$$\begin{aligned} Prob[C_k' = y_{i_k+1}] &= Prob[B] \cdot Prob[A|B] + Prob[B^c] \cdot Prob[A^c|B^c] \\ &= Prob[B] \cdot Prob[A|B] + Prob[B^c] - Prob[B^c] \cdot Prob[A|B^c] \\ &= Prob[B] \cdot Prob[A|B] + Prob[B^c] - (Prob[A] - Prob[B] \cdot Prob[A|B]) \\ &= \frac{1}{2} + Prob[A|B] - Prob[A] \\ &= \frac{1}{2} + f(i_k + 1) - f(i_k) \\ &> \frac{1}{2} + \frac{1}{p'(k)} \end{aligned}$$

Hence the conclusion is:

$$[Prob[C'_k(G(U_k)_{[1,\dots,i_k]}) = G(U_k)_{i_k+1}] > \frac{1}{2} + \frac{1}{p'(k)}$$

(We use subscript notation to take partial bits of a given bit sequence. In this particular case $G(U_k)_{[1,\dots,i_k]})$ is the first i_k bits of $G(U_k)$ and $G(U_k)_{i_k+1}$ is the i_k+1's bit of $G(U_k)$.)

That is, C'_k can guess the i_k'th bit of $G(U_k)$ with $\epsilon\frac{1}{p'(k)}$ advantage over unbiased coin toss.

Let us now extend C' to also get complete seed as input (in addition to $S[I_1], S[I_2], \dots, S[I_{i_k}]$)

$$C''_k(S \circ G(S)_{[1,\dots,i_k]})C'_k(G(S)_{[1,\dots,i_k]})$$

Since $G(U_k)_{i_k+1}$ is defined to be $b(S[I_{i_k+1}])$, we have:

$$Prob_s[C''_k(S \circ G(S)_{[1,\dots,i_k]}) = b(S[I_{i_k+1}])] > \frac{1}{2} + \epsilon$$

We claim that there exist $\alpha \in \{0,1\}^{k-|I_{i_k+1}|}$ such that:

$$Prob_s[C''_k(S \circ G(S)_{[1,\dots,i_k]}) = b(S[I_{i_k+1}]) \mid S[\overline{I_{i_k+1}}] = \alpha] > \frac{1}{2} + \epsilon$$

This claim is true because if we look at a random selection of S as two independent selections of $S[I_{i+1}]$ and $S[\overline{I_{i+1}}]$ we see that the average over the second selection is greater than $\frac{1}{2}+\epsilon$ so there must be an element with weight greater than that.

Now we come to the key argument in this proof. By incorporating C''_k with α to a new circuit, we get a circuit that can approximate the value of $b(S[I_{i_k+1}])$ but it need the "help" of $b(S[I_1]), b(S[I_2]), \dots, b(S[I_{i_k}])$. We will show now that we can do without this "help" when $\{I_1, \dots, I_m\}$ is a design (note that we didn't use the fact that this is a design until now).

To prove that it is possible to build a circuit that doesn't use the "help", we need to show that there exist a polynomial-size circuit that get only $S[I_{i_k+1}]$ and can approximate $b(S[I_{i_k+1}])$. To do this we use the fact that all the bits in $S[I_1], S[I_2], \dots, S[I_{i_k}]$ depend only on small fraction of $S[I_{i_k+1}]$ so circuits for computing these bits are relatively small and we can incorporated them in a circuit that include all possible values of these bits. Details follows.

To elaborate the last paragraph, recall that the intersection of any two subsets in a design is at most $O(log\ k)$, hence given $S[\overline{I_{i_k+1}}] = \alpha$ we know that for every $i \le i_k$ the bits in I_i are fixed except for $O(log\ k)$ bits that may be in $S[I_{i_k+1}]$, and are given as part of the input. Since there are at most $O(log\ k)$ such bits, there exist a polynomial-size circuit that can "precompute" the value of b for every combination of these bits.

The first part of this circuit is a collection of tables, one for every I_i. Each table is indexed by all possible values of the "free bits" of an I_i (i.e., these in I_{i_k+1}). The entry for every such value (of $S[I_{i_k+1} \cap I_i]$) contains the corresponding $b(S[I_i])$ (under $S[\overline{I_{i_k+1}}] = \alpha$).

The second part of this circuit just implements a "selector"; that is, it uses the bits of $S[I_{i_k+1}$ in order to obtain the appropriate values of $b(S[I_1]), b(S[I_2]), \dots, b(S[I_{i_k}])$ from the corresponding tables mentioned above.

Since we have polynomially many entries in every table and polynomially many tables we get that this is a polynomial-size circuit.

The conclusion is that we got a circuit that can approximate b with a non negligible advantage over unbiased coin toss. There exist infinitely many such circuits for infinitely many k's, which
$\boxed{1}$ $\boxed{15.4}$ $\Rightarrow$ contradict the assumption of b being unpredictable predicate.

4 Constructions of a design

In this section we describe how to construct a design that can be used for the generator that we introduced in the preceding section. We need to construct m different subsets of the set $\{1, ..., k\}$ each has size l with small intersections.

4.1 First construction: using $GF(l)$ arithmetic

Assume without loss of generality that l is a prime factor and let $k = l^2$ (if l is not a prime factor pick the smallest power of 2 which is greater than l).

For the field $FGF(l)$, we get that the Cartesian product $F \times F$ contains $k = l^2$ elements which we identify, with the k elements of $\{1, ..., k\}$. Every number in $\{1, ..., k\}$ is assigned to a pair in $F \times F$ (in a one-to-one correspondence). In the forgoing discussion we will interchange the pair and it's representative in $\{1, ..., k\}$ freely.

For every polynomial $p(\cdot)$ of degree d over F introduce the subset:

$$I_p\{\langle e, p(e)\rangle : e \in F\} \subseteq F \times F$$

We get that:

The size of each set is $|I_p| = |F| = l$.

For every two polynomials $p \neq q$ the sets I_p and I_q intersects in at most d points (that is, $|I_p \cap I_q| \leq d$). This is true since:

$$|\{\langle e, p(e)\rangle : e \in F\} \cap \{\langle e, q(e)\rangle : e \in F\}| = |\{\langle e : p(e) = q(e)\}|$$

But the polynomial $p(e) - q(e)$ has degree smaller or equal to d so it can only have d zeroes (due to the Fundamental Theorem of Algebra).

There are $|F|^{d+1} = l^{d+1}$ polynomials over $GF(l)$ so for every polynomial $P(\cdot)$ we can find d such that the number of sets is greater than $P(l)$.

This structure is constructible in exponential (actually polynomial) in k time because all we need to do is simple arithmetic in $GF(l)$.

The conclusion is that we have a design (see Definition **??**) that can be applied in the construction of pseudorandom generator that we gave above. This design remove the second assumption that we made about the existence of a design, so we get (as promised) that the only assumption needed in order to allow derandomization of BPP is that there exist an unpredictable predicate.

4.2 Second construction: greedy algorithm

In this subsection we introduce another construction of a design. We call this algorithm greedy because it just scans all the subsets of the needed size until it find one that doesn't overlap all previously selected subsets too much. This algorithm is "simpler" than the one we gave in the previous subsection but we need to show that it work correctly.

Consider the following parameters:

- $k = l^2$
- $m = poly(k)$
- We want that for all i to have $|I_i| = l$ and for all $i \neq j$, $|I_i \cap I_j| = O(log\ k)$

For these parameters we give a simple algorithm that scans all subsets one by one to find the next set to include in the design:

for $i = 1$ to m

for all $I \subset [k]$, $|I| = l$ do

$flag \leftarrow FALSE$

for $j = 1$ to $i - 1$

if $|I_i \cap I_j| > log\ k$ then $flag \leftarrow TRUE$

if $flag = TRUE$ then $I_i \leftarrow I$

This algorithm run in an exponential time because there are 2^l subsets of size l and we scan them m times. Since $m \cdot 2^l < 2^k$ we get that even if we had to scan all the subsets in every round we could finish the process in time exponential in k.

To prove that the algorithm works it is enough to show that if we have $I_1, I_2, ..., I_{i-1}$ such that

$i \leq m$

For every $j < i : |I_j| = l$

For every $j_1, j_2 < i : |I_{j_1} \cap I_{j_2}| \leq log\ m + 2$ Then there exist another set $I_i \subset [k]$ such that $|I_i| = l$ and for every $j < i : |I_j \cap I_i| < 2 + log\ m$. We prove this claim using the Probabilistic Method. That is, we will show that most choices of I_i will do. In fact we show the following

Let $I_1, I_2, ..., I_{i-1} \subset [k]$ each of size l. Then there exists an l-set, I, such that for all j's, it is the case that $|I_j \cap I| \leq \log\ m + 2$.

We first consider the probability that a uniformly chosen l-set has a large intersection with a fixed l-set, denoted S. Actually, it will be easier to analyze the intersection of S with a set R selected at random so that for every $i \in [k]$

$$Prob[i \in R] = \frac{2}{l}$$

That is, the size of R is a random variable, with expectation $2k/l$ (which equals $2l$). We will show that with very high probability the intersection of R with S is not too big, and that very high probability $|R| \geq l$ (and so we can find an appropriate l-set). Details follow.

Let s_i be the i'th element in S sorted in any order (e.g., the natural increasing order). Consider the sequence $\{X_i\}_{i=1}^{l}$ of random variables defined as

$$X_i \begin{cases} 1 & if\ s_i \in R \\ 0 & otherwise \end{cases}$$

Since these are independent Boolean random variables with $Prob[X_i = 1] = \frac{2}{l}$ for each i, we can use Chernoff's bound to get:

$$
\begin{aligned}
Prob\left[|S \cap R| > 2 + log\ m\right] &= Prob\left[\sum_{i=1}^{l} X_i > 2 + log\ m\right] \\
&= Prob\left[\frac{\sum_{i=1}^{l} X_i}{l} > \frac{2}{l} + \frac{log\ m}{l}\right] \\
&< Prob\left[\left|\frac{\sum_{i=1}^{l} X_i}{l} - \frac{2}{l}\right| > \frac{log\ m}{l}\right] \\
&< 2 \cdot e^{-log^2\ m} \ll 1/2m
\end{aligned}
$$

It follows that for R selected as above, the probability that there exists an I_j so that $|R \cap I_j| > 2 + \log m$ is bounded above by $\frac{i-1}{2m} < \frac{1}{2}$. The only problem is that such an R is not necessarily of size l. We shall show that with high probability the size of R is at least l, and so it contains a subset which will do (as I_i).
Consider the sequence $\{Y_i\}_{i=1}^{k}$ defined as

$$
Y_i \begin{cases} 1 & if\ i \in R \\ 0 & otherwise \end{cases}
$$

Then, applying Chernoff's Bound we get:

$$
\begin{aligned}
Prob[|R| < l] &\leq Prob[|\tfrac{1}{k} \textstyle\sum_{i=1}^{k} Y_i - \tfrac{2}{l}| < \tfrac{1}{l}] \\
&< 2 \cdot e^{-2} \ll \tfrac{1}{2}
\end{aligned}
$$

Thus, for R selected as above, the probability that either $|R| < l$ or there exists an I_j so that $|R \cap I_j| > 2 + \log m$ is *strictly smaller than 1.* Therefore , there exists a set R such that $|R| \geq l$ and yet, for every $j < i$, we have $|R \cap I_j| \leq 2 + \log m$. Any l-subset of R qualifies as the set asserted by the claim.

We stress that this discussion about a randomly selected set is not part of the algorithm. The algorithm itself is totally deterministic. The randomness is just in our discussion – it serves as a tool to show that the algorithm will always find what it is looking for in every step.