Simulating BPP Using a General Weak Random Source

David Zuckerman*

Abstract

We show how to simulate BPP and approximation algorithms in polynomial time using the output from a δ-source. A δ-source is a weak random source that is asked only once for R bits, and must output an R-bit string according to some distribution that places probability no more than $2^{-\delta R}$ on any particular string.

We also give two applications: one to showing the difficulty of approximating the size of the maximum clique, and the other to the problem of implicit $O(1)$ probe search.

1 Introduction

Randomness plays a vital role in almost all areas of computer science, both in theory and in practice. Randomized algorithms are often faster or simpler than the deterministic algorithms for the same problem (see e.g. [Rab]).

To produce "random" bits, a computer might consult a physical source of randomness, such as a Zener diode, or use the last digits of a real time clock. In either case, it is not clear how random these "random" bits will be. Moreover, it is impossible to verify the quality of a random source. It is therefore of interest to see if weak, or imperfect, sources of randomness can be used in randomized algorithms. Indeed, the fewer assumptions we make about the quality of our random source, the greater the chances are that the source satisfies these assumptions and hence that our randomized algorithms work correctly. This emphasis on reliability seems even more important as computers become faster, because users should be willing to pay a greater price for greater accuracy. This ties in with the recent interest in checking.

The history of weak random sources reflects this ideal of using as weak a source as possible. Von Neumann [vN] initiated the study of weak random sources by showing how to extract perfectly random bits from a source of independent coin flips of equal but unknown bias. Blum [Blu] generalized this by giving algorithms to convert the output of a Markovian source into truly random strings. Then Santha and Vazirani [SV] introduced the model of semi-random sources, where the probability of a given bit being a specific value, conditional on the value of previous bits, is not too large. From one semi-random source, they proved that it is impossible to extract even a single random bit (so Vazirani [Va1,Va3] used two independent sources).

In light of this result, one might give up hope for simulating randomized algorithms with one semi-random source. Nevertheless, [VV] and [Va2] showed how to simulate RP and BPP with one semi-random source. Chor and Goldreich [CG2] generalized these results by showing how to simulate BPP using a weaker model of a source, namely they assumed no sequence of $O(\log R)$ bits has too high a probability of being a particular sequence (here R denotes the total number of random bits used).

Various authors have also considered models where an adversary chooses the values of certain bits, but the others are random (see [CG+], [BL], [LLS], [CWi]).

In [Z], δ-sources were presented, which essentially generalize all of the above models[1], making no structural assumptions about dependencies:

Definition 1 *A δ-source is asked only once for R bits, and the source outputs an R-bit string such that no string has probability more than $2^{-\delta R}$ of being output, for some fixed $\delta > 0$.*

Compare this with the Chor-Goldreich PRB-source (we modify their definition slightly to make the comparison easier):

Definition 2 (CG2) *An (l, δ) PRB-source outputs R bits as R/l blocks $x_1, \ldots, x_{R/l}$, each of length l, such that for all l-bit strings $y_1, \ldots, y_{R/l}$,*

$$Pr[x_i = y_i | x_1 = y_1, \ldots, x_{i-1} = y_{i-1}] \leq 2^{-\delta l}.$$

The results of [CG2] hold only for $l = O(\log R)$.

In [Z], it was shown how to simulate RP using a string from a δ-source in time $n^{O(\log n)}$, or in polynomial time

*Current address: MIT Laboratory for Computer Science, Cambridge, MA 02139, diztheory.lcs.mit.edu. This research was done while the author was a student at U.C. Berkeley, and supported by an AT&T Graduate Fellowship, NSF PYI Grant No. CCR-8896202, and NSF Grant No. IRI-8902813.

[1] One of the bit-fixing sources in [CWi] has a weaker entropy bound than that which we impose. Our model can be modified to generalize this source, too, but then our simulations would fail.

CH3062-7/91/0000/0079$01.00 © 1991 IEEE

under the Generalized Paley Graph Conjecture. Actually, Sahay and Sudan [SS] pointed out a mistake in that paper, which we correct in this paper by introducing what we call the Modified Leftover Hash Lemma, a modification of the Leftover Hash Lemma appearing in [ILL].

In our main results, we improve the time of this algorithm to polynomial, and in addition give a polynomial simulation for BPP. In fact, our BPP simulation is more general, yielding simulations for approximation algorithms, e.g. those used to approximate the volume of a convex body [DFK]. We include separate algorithms for RP and BPP, because the RP algorithm has the advantage of using few random bits from the δ-source. Namely, if r truly random bits are used by an RP algorithm to achieve probability of success $1/2$, then $O(r \log r)$ bits from a δ-source are used by our RP simulation. Hence this is also a quasi-perfect pseudo-random generator (see [San], [Sip]). Our BPP algorithm requires $r^{O(1/\delta)}$ bits.

The difficulty in improving the RP simulation of [Z] to polynomial is that it requires too much time to get "good" blocks of different sizes. We get around this problem by introducing a new lemma about paths on expander graphs, which allows us to find the good blocks without using brute force.

The biggest obstacle in extending this simulation to work for BPP is that "good" blocks are needed at every stage of the algorithm. We solve this problem by considering several permutations, and arguing that most of the permutations will have all "good" blocks. The argument requires more care than at first appears, as we must argue first about initial segments being good, and then individual blocks as being good. We use few permutations by using pairwise independence.

These simulations are equivalent to the explicit construction of a certain type of expander graph, called a disperser (see [San], [Sip], [CWi]). We believe these disperser constructions will be useful in many different areas; to illustrate this belief we give two applications: one to showing the difficulty of approximating the size of the maximum clique, and the other to the problem of implicit $O(1)$ probe search.

Computing $\alpha = \alpha(G)$, the size of the maximum clique, is well known to be NP-complete [K]. Nothing was known about the difficulty of approximating α until Feige et.al. [FG+] showed that if approximating α to within a factor of $2^{(\log n)^{1-\epsilon}}$ is in $\tilde{P}$, then $N\tilde{P} = \tilde{P}$.

Given such a huge factor, one can ask the question: can we at least get a good estimate of the order of magnitude of α? More precisely, is there an algorithm that for some constant t outputs a number between $\alpha^{1/t}$ and α^t? One would expect this problem to be easier, because the algorithm has to decide among only $O(\log n)$ approximations.

Moreover, it seems more reasonable to ask for an approximation factor as a function of the clique size, rather than the graph size. It probably does a user little good to be able to distinguish between cliques of size $O(1)$ and $2^{(\log n)^{1-\epsilon}}$, whereas it might be useful to distinguish between cliques of size n^δ and $n^\delta 2^{(\log n)^{1-\epsilon}}$. The results of [FG+] show it is difficult to distinguish between cliques of size n^δ and $n^\delta 2^{(\log n)^{1-\epsilon}}$, and therefore give no answer to our question. By applying our disperser construction to the proof of [FG+] we show that the existence of such an approximation algorithm implies $N\tilde{P} = \tilde{P}$.

Implicit $O(1)$ probe search is the problem of arranging n elements from the domain $\{1, \ldots, m\}$ in a table of size n so that searching for an element can be done with a constant number of probes. Fiat, Naor, Schmidt, and Siegel [FNSS], building on [FKS], showed how to do this if $m = O(n)$. Fiat and Naor [FN] improved this to the case m is polynomial in n, and gave a nonconstructive solution if m is at most exponential in n. They did this by showing that constructing an implicit $O(1)$ probe search scheme is equivalent to constructing a certain combinatorial structure which they call a rainbow. In addition to their construction for m polynomial in n, they show how to construct rainbows for the case $m = n^{\log n}$, assuming a certain type of disperser can be explicitly constructed, as was conjectured by Sipser [Sip].

We cannot construct this disperser, but by constructing three other dispersers, two of which come directly from the RP construction above and the third of which relies heavily on the theory we've developed, we give the first constructive solution for m superpolynomial in n. The exact function is $m = n^{\frac{\sqrt{\log\log n}}{\log\log\log n}}$. Our solution is weaker than the earlier results, however, in that it does not allow search to be done in constant time, despite the constant number of probes.

2 Preliminaries

Most of this section is standard or included in [Z]. The main point to take away is the last lemma: the reduction from δ-sources to PRB-sources.

Definition 3 *RP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial time Turing machine $M_L(a,x)$ for which*

$$\begin{aligned} a \in L &\Rightarrow Pr[M_L(a,x) \text{ accepts}] \geq 1/2 \\ a \notin L &\Rightarrow Pr[M_L(a,x) \text{ accepts}] = 0 \end{aligned} \quad (1)$$

where the probabilities are for an x picked uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial p.

Definition 4 *BPP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial time Turing machine $M_L(a,x)$ for which*

$$a \in L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \geq 2/3 \quad (2)$$

$$a \notin L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \leq 1/3 \quad (3)$$

where the probabilities are for an x picked uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial p.

As is well known, by running M_L on independent random tapes, we can change the probabilities in (1), (2), and (3) to $1-2^{-poly(|a|)}$, $1-2^{-poly(|a|)}$, and $2^{-poly(|a|)}$, respectively. Furthermore, the results of Cohen and Wigderson [CWi] (see also [AKS], [IZ], and [Z]) imply that we can take the above probabilities to be $1-2^{-r/8}$, $1-2^{-r/8}$, and $2^{-r/8}$, respectively, where $r = |x|$.

We wish to simulate RP using a δ-source; say we wish to test whether a given element a is in L. If $a \notin L$, then all random strings cause M_L to reject, so there is nothing we can do. Suppose $a \in L$; then we wish to find with high probability a witness to this fact. Let W be the set of witnesses, i.e. $W = \{x | M_L(a,x) \text{ accepts}\}$, and N be the set of non-witnesses, i.e. the complement of W.

Definition 5 *A polynomial-time algorithm A simulates RP using a δ-source if it takes as input $R = poly(r)$ bits from the δ-source and outputs a polynomial number of r-bit strings, one of which lies in W with probability $1-2^{-\Omega(r)}$.*

For BPP, we have no "witnesses," but whether or not $a \in L$, we use W as the set of random strings producing the right answer, and N as the complement of W. Moreover, the simulation does not have to take the majority of the answers of the output strings, but can instead use any function of the answers. In fact, we will build several ternary trees, take the majority of majorities in the trees, and then the majority of the answers given by the trees.

Definition 6 *A polynomial-time algorithm B simulates BPP using a δ-source if it takes as input $R = poly(r)$ bits from the δ-source and outputs a polynomial number of r-bit strings, each of which answers $a \in L$ or $a \notin L$; B then takes some function of these answers to produce a single answer, which must be correct with probability $1-2^{-\Omega(r)}$.*

Note that such an algorithm B can be used to simulate approximation algorithms. This is because whenever a majority of numbers lie in a given range, their median also lies in that range. Thus, by taking medians instead of majorities, B can output a good approximation with probability $1-2^{-\Omega(r)}$.

We now define flat sources and show that we may assume without loss of generality that our δ-source is flat.

Definition 7 (CG2) *A flat δ-source is a source which places probability $2^{-\delta R}$ on $2^{\delta R}$ R-bit strings.*

Lemma 1 *[CG2] Suppose an algorithm simulates RP (BPP) from a flat δ-source. Then it also simulates RP (BPP) from a δ-source.*

Proof. The proof in our case is much simpler than in [CG2]. Fix an algorithm A. On certain input strings, A outputs the correct answer, and on others it doesn't. The δ-source may as well place as much probability as possible on the strings for which A is incorrect, i.e. the δ-source may as well be a flat δ-source. □

Remark: This observation, that some strings are good for A and others aren't, implies that an algorithm simulates RP (BPP) with probability $1-2^{-\Omega(R)}$ from a δ-source if and only if it simulates RP (BPP) with non-zero probability from a δ'-source, for some $\delta' < \delta$. This is because the probability of outputting the wrong answer is at most the number of bad strings divided by $2^{\delta R}$. Therefore, we need not worry about simulating RP or BPP with high probability, but only with non-zero probability.

Our algorithms to simulate RP divide the R-bit string X into r-bit strings $x_1, \ldots, x_{k'}$ (where $R = O(r \log r)$) and combine these strings in various ways. In order to prove that our algorithms work, we need the following two lemmas, which show that it suffices to prove the correctness of our algorithm on a (cr, δ) PRB source as long as we only use $O(r \log r)$ bits from the source. Note that cr is much larger than the block lengths used in [CG2], so our techniques are completely different.

Lemma 2 (Z) *Fix $\alpha > 0$, $\delta' < \delta$, and an integer $k > 0$. Let $k' = \lceil \frac{k(1-\delta')+\alpha}{\delta-\delta'-l-1} \rceil = O(k)$, and set $R = k'l$. For each initial string $X_i = x_1 \circ x_2 \circ \ldots \circ x_i$ (here $\circ$ denotes concatenation), label any $2^{\delta' l}$ of the x_{i+1}'s as "δ'-bad"; the others we call "δ'-good" (when it is not ambiguous, we will simply say "bad" and "good"). Then*

$$Pr[\text{for} \geq k \text{ values of } i,\ x_i \text{ is good}] > 1 - 2^{-\alpha l}.$$

Lemma 3 (Z) *If we have an algorithm that simulates RP using the output of a (cr, δ') PRB-source of length $O(r \log r)$, then we can construct an algorithm that simulates RP using the output of a δ-source, for any $\delta > \delta'$.*

3 A Hashing Lemma

Definition 8 *A probability distribution D on a set S is* quasi-random *within ϵ if for all $X \subseteq S$ $|D(X) - |X|/|S|| \leq \epsilon$. Here $D(X)$ denotes the probability of the set X according to distribution D.*

Definition 9 (CWe) *Let A and B be two sets, and H a family of functions from A to B. H is called a* universal family of hash functions *if for every $x_1 \neq x_2 \in A$ and $y_1, y_2 \in B$,*

$$Pr_{h \in H}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] = 1/|B|^2.$$

It will become important that there is a universal family of hash functions from $\{0,1\}^m$ to $\{0,1\}^n$ of size 2^{m+n}. To see this, let F be the finite field on 2^m elements. Then $H = \{(a,b) | a \in \{0,1\}^m, b \in \{0,1\}^n\}$ is universal, where

$$(a,b)(x) = (\text{last } n \text{ bits of } a \cdot x) \oplus b,$$

where the multiplication is in F and $\oplus$ denotes bitwise exclusive-or. Using the results of [Sho], we can efficiently construct F deterministically.

Leftover Hash Lemma [ILL]: Let $X \subset \{0,1\}^n, |X| \geq 2^l$. Let $e > 0$, and let H be a universal family of hash functions mapping n bits to $l - 2e$ bits. Then the distribution $(h, h(x))$ is quasi-random within $1/2^e$ (on the set $H \times \{0,1\}^{l-2e}$), where h is chosen uniformly at random from H, and x uniformly from X.

It is important that we have the ordered pair $(h, h(x))$, because it allows us to use the same hash function repeatedly, even after outputting $h(x)$. When we do apply it repeatedly, the following useful lemma will imply that the statistical error only grows additively.

Lemma 4 *Suppose f is a function such that if x is picked uniformly at random, then $f(x)$ is quasi-random within ϵ. Then if x is quasi-random within ϵ', then $f(x)$ is quasi-random within $\epsilon + \epsilon'$.*

We will need the Modified Leftover Hash Lemma to deal with the situation when the hash function is not close to uniformly random. For our purposes, it would suffice to derive a bound from the Leftover Hash Lemma; however, we can derive a stronger bound, which we include for the sake of completeness. Note that the bound is close to that obtained in [MNT].

Lemma 5 (Modified Leftover Hash Lemma) *Let $H = \{h : I \to O\}$ be a universal family of hash functions, and let $A \subset I$, $B \subset O$, and $C \subset H$. Then the distribution $(h, h(x))$, where h is chosen uniformly from C and x uniformly from A, is quasi-random within $\sqrt{\frac{|H||O|}{|C||A|}}$ on the set $C \times O$.*

Proof. We use ideas of the proofs in [MNT],[BBR], [ILL], and [IZ]. Let $p_{h,z} = Pr[h(x) = z]$ when x is chosen uniformly from A. Thus the probability that a particular (h,z) appear in the distribution above is $\frac{p_{h,z}}{|C|}$. The variation distance between the two distributions above on $(h, h(x))$ is

$$\begin{aligned} & \textstyle\sum_{h \in C, z \in O} \left| \frac{p_{h,z}}{|C|} - \frac{1}{|O||C|} \right| \\ \leq & \sqrt{|C||O| \textstyle\sum_{h \in C, z \in O} \left(\frac{p_{h,z}}{|C|} - \frac{1}{|C||O|} \right)^2} \\ \leq & \sqrt{|C||O| \textstyle\sum_{h \in H, z \in O} \left(\frac{p_{h,z}}{|C|} - \frac{1}{|C||O|} \right)^2} \\ = & \sqrt{|C||O| \textstyle\sum_{h \in H, z \in O} \left(\frac{p_{h,z}^2}{|C|^2} - \frac{2p_{h,z}}{|C|^2|O|} + \frac{1}{|C|^2|O|^2} \right)}. \end{aligned}$$

Now

$$\sum_{h \in H, z \in O} p_{h,z}^2 = |H| Pr[h(x) = h(y)]$$

if h is picked uniformly from H, and x, y independently and uniformly from A. This probability is therefore equal to

$$Pr[x = y] + Pr[x \neq y] \frac{1}{|O|} < \frac{1}{|A|} + \frac{1}{|O|}.$$

Using the above and $\sum_{z \in O} p_{h,z} = Pr[h(x) \in O] = 1$ yields the lemma. □

Note that the proof of the lemma only makes use of the hash family having the right collision probability, and not the full pairwise independence.

4 Simulating RP

We now present algorithms which simulate RP. Suppose $a \in L$ and $|W| \geq 2^r(1 - 2^{-r/8})$. Because of Lemma 3, we assume all blocks of $\Omega(r)$ bits are output from an $(\Omega(r), \delta)$ PRB-source. The starting point of our algorithm is due to [VV]: suppose that we had a function $f : \{0,1\}^r \times \{0,1\}^w \to \{0,1\}^r$, such that for a random w-bit string X from the source, the induced function $f(\cdot, X)$ (denoted $f_X(\cdot)$) maps many non-witnesses to witnesses. Then we could define a new witness set $W' = \{x | x \in W \text{ or } f(x, X) \in W\}$ which is much larger than W, with high probability.

We could then recurse on this idea, constructing W'' which is much larger than W', and eventually constructing $W^{(k)} = \{0,1\}^r$, where $k = O(\log r)$. Of course, if $a \notin L$, then $W^{(k)} = \emptyset$. Thus, in order to test whether $a \in L$, we need only test whether $0 \in W^{(k)}$. This is equivalent to $0 \in W^{(k-1)}$ or $f_{X_k}(0) \in W^{(k-1)}$, which is further equivalent to $0 \in W^{(k-2)}$ or $f_{X_k}(0) \in W^{(k-2)}$ or $f_{X_{k-1}}(0) \in W^{(k-2)}$ or $f_{X_{k-1}}(f_{X_k}(0)) \in W^{(k-2)}$.

Fleshing out the recursion completely, we are left with the algorithm that tests whether all combinations like $f_{X_1}(f_{X_5}(f_{X_6}(0))) \in W$. Observe that $k = O(\log r)$ implies a polynomial number of queries. Also notice that although our analysis works from W to W' and on to $W^{(k)}$, the algorithm works from $W^{(k)}$ to W.

The key idea for thinking about constructing such functions f was introduced in [Z]: imagine that if the inputs to f are chosen randomly, i.e. the r-bit string uniformly from the set of non-witnesses and the other bits according to a PRB-source, then the output is quasi-random within $2^{-3\delta r/16}$. Substituting in the string from our source, we obtain a function which maps random non-witnesses to quasi-random r-bit strings. Therefore, a large fraction of these non-witnesses will be mapped to witnesses, which is what we want.

Observe that this makes it easier to construct an f if N is large. We will therefore first show how to do this if N is large, and then how to make progress when N is small.

To get such a function f when N is large, we make use of the Modified Leftover Hash Lemma. Set $t = \lceil r/(1+\delta/3) \rceil$, and $u = r - t$, so that $u \leq \delta t/3$. We now view our r-bit non-witness as a hash function h mapping t-bit strings to u-bit strings. We then use $\lceil r/u \rceil$ t-bit blocks $y_1, \ldots, y_{\lceil r/t \rceil}$ from the source, and the output of f is the first r-bits of $h(y_1) \circ \cdots \circ h(y_{\lceil r/t \rceil})$ (here $\circ$ denotes concatenation). The Modified Leftover Hash Lemma implies that the output of f is quasi-random.

Thus we get a new set of non-witnesses N', where

$$N' = \{s \in N | f_{X_1}(s) \in N\}. \tag{4}$$

We now show

Lemma 6

$$E[|N'|] \leq |N| 2^{-r} + O(|N| 2^{-3\delta r/16}) \leq (1 + o(1)) 2^{(1-\delta/8)^2 r}. \tag{5}$$

Proof. Imagine that $s \in N$ is picked uniformly at random, and view it as a hash function h. Using the Modified Leftover Hash Lemma and Lemma 4 yields that the output of f is quasi-random within $2^{-3\delta/16}$. Multiplying the probability that this output lies in N by $|N|$ yields the result. □

Corollary 1

$$Pr[|N'| \leq 2^{(1-\delta/8)(1-\delta/10)r}] \geq 1 - 2^{-\delta r/50}.$$

Proof. Use Lemma 6 and Markov's Inequality. □

We can now prove:

Theorem 1 *For all $\delta > 0$, there is an algorithm that simulates RP using a δ-source.*

Proof. Since N' is too small to give f desireable properties, we make progress not by reducing N', but by reducing r. Let $r' = \lceil (1 - \delta/10) r \rceil$. We can map an r'-bit string to an r-bit string simply by padding with 0's on the front. This gives us new corresponding witness and non-witness sets W'' and N'' in $\{0,1\}^{r'}$, namely, an element is a witness in $\{0,1\}^{r'}$ iff it is mapped to a witness in $\{0,1\}^r$. Moreover,

$$|N''| \leq |N'| \leq 2^{(1-\delta/8)(1-\delta/10)} \leq 2^{(1-\delta/8)r'}.$$

Thus, replacing r by r' and N by N'', we are left with the same problem but the size has decreased by a factor of $(1 - \delta/10)$. Continuing in this manner, after $O(\log r/\delta)$ stages we will be working over logarithmic size sets, for which we can check every element (it will be more convenient to take the recursion only to logarithmic size sets, instead of to constant size sets as we had indicated earlier). Note that Corollary 1 ensures that all reductions in the size of N occur with high probability, as we get a geometric progression. (By our initial remark, it suffices to show that the statements hold with non-zero probability.)

We still have one problem left: each stage needs good blocks of different lengths. This was the main obstacle to improving the $n^{O(\log n)}$ algorithm of [Z] to polynomial.

It suffices to get $\delta/2$-good blocks, simply by replacing δ by $\delta/2$ in the above argument. Suppose at the ith stage we need a good block of length l_i. We will get this $\delta/2$-good block from the ith block of length r output by the PRB-source. Actually, the proof will be easier if we assume we have k δ-good blocks available to us, as implied by Lemmas 2 and 3. We may assume without loss of generality that $l_i | r$, for if it does not, we may pad the ith r-bit block with 0's until it does, and we will still have a $\delta/2$-good block of length r (so we will have to replace δ by $\delta/2$).

To get the good block of length l_i, let an adversary label $2^{\delta l_i/2}$ of the blocks bad, so the rest are good. Subdivide a block of length r into $b_i = r/l_i$ blocks of length l_i, and call a block of length r good if at least $\delta/4$ fraction of the sub-blocks are good. Assuming $l_i \geq 4/\delta$, Lemma 2 implies that less than $2^{\delta r}$ blocks of length r are not good. Thus, this is a legitimate labelling, so by the techniques of Lemmas 2 and 3 we may assume we can get k blocks, where for all i the ith block has at least $\delta/4$ fraction of sub-blocks of length l_i good.

To get one good block of each length, we could use brute force and try all possible combinations of sub-blocks, one of each length. This, however, yields an $r^{O(\log r)}$ algorithm. If there were only one good sub-

block of each length, then we would have to use brute force; however, we know that a constant fraction of the sub-blocks are good. We exploit this fact by using the following lemma:

Lemma 7 *Let $G = (V, E)$ be an expander graph (directed or undirected) on $n = |V|$ nodes, such that every set of size $n/2$ has at least $n/2 + \alpha n$ neighbors. For any k, let $S_1, \ldots, S_k$ be arbitrary subsets of V such that $|S_i| \geq (1-\alpha)n$. Then there exists a path $v_1, \ldots, v_k$ in G such that $v_i \in S_i$.*

Proof. By induction on k in the statement: there are at least $n/2$ endpoints v_k in such paths $v_1, \ldots, v_k$. This is true for $k = 1$; if it is true for a given k, then any neighbor of an endpoint that also lies in S_{k+1} is a possibility for v_{k+1}. But the neighbor set is of size at least $n/2 + \alpha n$, so its intersection with S_{k+1} is at least $n/2$. □

Viewing the degree 7 expanders in [GG] as directed graphs (instead of bipartite graphs), we see that these graphs have $\alpha = (2 - \sqrt{3})/4$, as well as being explicitly constructible.

We apply the lemma as follows: Using $n = r$, we'd like to set S_i equal to the set of good sub-blocks of the ith good block B_i. However, we would then have a fraction $\delta/4$ of the sub-blocks in S_i, whereas we want a fraction $1 - \alpha$. We therefore set $n = r^m$, and let S_i represent m-tuples of sub-blocks, of which at least one is good, where m is chosen so that $(1 - \delta/4)^m \leq \alpha$. We also must work with a modulus, so S_i denotes the following set:

$$\{(s_1, \ldots, s_m) | (\exists j) s_j (\mathrm{mod} r/l_i) \text{th sub-block of } B_i \text{ is good}\}.$$

We then don't have to run our algorithm on all combination of sub-blocks, but only on ones given by paths on the [GG] expanders. Since there are $7^{O(\log r)}$ paths of length $O(\log r)$ in a degree 7 expander, and each such path contains $m^{O(\log r)}$ combinations of sub-blocks, this yields a polynomial time algorithm.

□

Remark: The time dependence on δ is $r^{O(\frac{1}{\delta^2} \log \frac{1}{\delta})}$, and $R = O(r \log r/\delta^3)$. This is because there are $O(\log r/\delta)$ stages, so $2^{O(\log r/\delta)}$ strings are produced for every sequence of possibly good blocks. The dominant factor is the number of sequences of possibly good blocks: each stage requires $\lceil r/t \rceil = O(1/\delta)$ good blocks, so $O(\log r/\delta^2)$ good blocks are needed in total. In order to get this many good blocks, we must request $O(\log r/\delta^3)$ blocks from the source (i.e. $R = O(r \log r/\delta^3)$, and try all sequences of length $O(\log r/\delta^2)$. This gives $\binom{\log r/\delta^3}{\log r/\delta^2} = 2^{O(\frac{\log r}{\delta^2} \log \frac{1}{\delta})}$. For each sequence of possibly good blocks, we must try all paths of m-tuples in an expander, which adds another factor of $m^{O(\log r/\delta^2)} = 2^{O(\frac{\log r}{\delta^2} \log \frac{1}{\delta})}$. Multiplying these factors together gives the result.

5 Simulating BPP

Most of the difficulty in extending the above ideas to BPP is that for BPP, it is difficult to argue anything unless *all* the blocks are good. One bad block is enough to lose control over the majority of the output test strings.

We first give an algorithm, Algorithm B, to simulate BPP, assuming we could always get good blocks. Our algorithm will be essentially the same as the RP algorithm, except that instead of using a function $f : \{0,1\}^r \times \{0,1\}^w \to \{0,1\}^r$, we use the same construction to give an $f : \{0,1\}^r \times \{0,1\}^w \to \{0,1\}^{2r}$. There is now a way of associating $x \in \{0,1\}^r$ with 2 other r-bit strings: namely, compute $f(x, X)$, where X is a good block from the source, and split it into two r-bit strings x_1 and x_2. In our recursion, the answer we associate with x is the majority of the answers associated previously with x, x_1, and x_2.

We let N be the set of strings that give the wrong answer. Thus,

$$N' = \{x | \text{at least two of } x, x_1, x_2 \text{ are in } N\}.$$

Using the fact that $f(x, X)$ is quasi-random within $2^{-3\delta r/16}$ if x is picked uniformly from N, (and surely if x is picked uniformly from $\{0,1\}^r - N$), we get

Lemma 8

$$\begin{aligned} E[|N|] &< |N| Pr[x_1 \in N \text{ or } x_2 \in N | x \in N] \\ &\quad + 2^r Pr[x_1 \in N \ \& \ x_2 \in N | x \in \{0,1\}^r - N] \\ &\leq 2|N|^2 2^{-r} + O(|N| 2^{-3\delta r/16}) \\ &\leq 2^{(1+o(1))(1-\delta/8)^2 r}. \end{aligned}$$

Proof. Similar to Lemma 6. □

Given this lemma, the rest follows as in the RP case, once we can assume all our blocks are good. The rest of the proof is devoted to showing that we can make this assumption.

The first idea for doing this is to simulate Algorithm B on several permutations of the original R-bit string X from the source. The hope is that for most of the permutations, all the blocks will be good.

Indeed, this is what we do. Imagine we have a source suggested by the following lemma:

Lemma 9 *Let S be a source outputting b blocks of length $l = \lceil 3/\delta \rceil$ one at a time, controlled by an adversary with the following restrictions:*

(i) the adversary must decide on-line whether to completely control the value of the output block, or only to select a set of size $\geq 2^{\delta l/3}$ from which the block is chosen uniformly at random;

(ii) the adversary must yield complete control on at least $\delta b/3$ blocks.

Then an algorithm that simulates BPP using b blocks of length l from any such source S also simulates BPP using bl bits from a δ-source.

Proof. Use Lemma 2 and ideas from Lemma 3. □

Because the blocks of length l will serve as components of other, bigger blocks, we call them sub-blocks. We also call a sub-block that the adversary does not completely control a $\delta/3$-random sub-block. Note how this compares with $\delta/3$-good: intuitively, they correspond to the same idea, but the proofs will be easier using the $\delta/3$-random definition.

Assume without loss of generality that b is a prime power, e.g. a power of 2. We will use $b^2 - b$ "permutations" (they will not really be permutations, but subsequences of sub-blocks), which except for a slight modification is a probability space where $\sqrt{b}$ of the sub-blocks are chosen pairwise independently. Namely, we work over the finite field F with b elements, and for all $(x, y) \in (F - \{0\}) \times F$, we let the ith sub-block in our sequence correspond to the element $ix + y$ in F. Note that because $x \neq 0$, all the sub-blocks in the subsequence are different.

We would like to argue that for most such subsequences, all of the larger blocks, each composed of many small sub-blocks, are good in the new, permuted order. The problem with this is that a sub-block which was $\delta/3$-random in the old order may not be $\delta/3$-random in the new order. For example, imagine two sub-blocks which are always equal, but otherwise uniformly random and independent of all the other sub-blocks. Then whichever sub-block appears first in the permuted string will be $\delta/3$-random, and the other will not.

We therefore take a different approach. We first argue that with high probability, all initial segments of length at least $m_0 = 24l \log r/\delta = O(\log r/\delta^2)$ in the permuted string will be good. We then show how to obtain good blocks from a source that outputs strings with all initial segments good.

To do this first part, we first argue that for most permutations, all initial segments contain many sub-blocks that were $\delta/3$-random in the unpermuted order; we then argue that this implies that most permutations contain only good initial segments (in the new permuted order). In fact, although it is true that all initial segments will be good, we will only need to deal with initial segments with lengths $m_i = \lceil m_0 d^i \rceil$ for some i, where $d \geq 2$ is a constant to be chosen later.

Lemma 10 *For a fraction $1 - 1/\log r$ of the subsequences, all initial strings of length m_i contain at least a fraction $\delta/6$ of $\delta/3$-random sub-blocks.*

Proof. When a subsequence is picked at random from a pairwise independent space, we can use Chebychev's inequality to bound the probability that a block of length m_i contains at least $\delta/6$ fraction of $\delta/3$-random blocks (see e.g. [CG1]), given that a fraction at least $\delta/3$ of all sub-blocks are $\delta/3$-random. Using the value of m_i, this is at least $1 - 2^{-(i+1)}/\log r$. This probability only increases by removing the points from the space that we removed, so this bound holds for our probability space. Adding the geometric series, the probability that this holds for all non-negative integers i is at least $1 - 1/\log r$. □

Lemma 11 *The probability that at least $1 - 2/\log r$ fraction of the subsequences have initial segments that are $\delta^2/36$-good for each length m_i is at least $1 - 1/r$.*

Proof. The idea is that if an initial segment has many $\delta/3$-random sub-blocks, it will be good. It is not this simple, however, because the adversary can choose which subsequences to give few $\delta/3$-random sub-blocks after seeing some of the output blocks, i.e. if some subsequences look like they will be bad anyway, she need not focus on these. To get around this, we will give the adversary even more power: namely, she can decide which subsequences to give few $\delta/3$-random sub-blocks after seeing how all subsequences would have turned out with many $\delta/3$-random sub-blocks. Since the adversary can give at most $1/\log r$ subsequences few $\delta/3$-random sub-blocks, we need only bound the probability that, if every subsequence contained many $\delta/3$-random sub-blocks, that at least $1/\log r$ of them would have a $\delta^2/36$-bad initial segment of length m_i.

If an initial segment of length m_i has at least $\delta/6$ fraction of $\delta/3$-random sub-blocks, then the probability of this initial segment taking on a specific value is at most $2^{-\delta^2 m_i/18}$. Thus, the probability it is $\delta^2/36$-good is at most $2^{\delta^2 m_i/36}$ times this value, or $2^{-\delta^2 m_i/36}$. The probability that a subsequence has a bad initial segment for some i is at most the sum of these, which is at most $2 \cdot 2^{-\delta^2 m_0/36} \leq 2/r^2$, using the value of m_0 in terms of l and $l \geq 3/\delta$. Thus, the expected fraction of bad subsequences is at most $2/r^2$, so by Markov the probability this fraction exceeds $1/\log r$ is at most $2 \log r/r^2 \leq 1/r$. □

For ease of reading, redefine δ as $\delta^2/36$, so now we know that for most subsequences, all initial segments of length m_i are δ-good. Let us focus now on these

subsequences. We now explicitly set the d in the definition of m_i; namely $d = 2/\delta$ so $m_i = \lceil m_0(2/\delta)^i \rceil$. We next show that the initial segments of length m_i being δ-good implies that the block containing bit positions $m_i + 1$ through m_{i+1} is $\delta/2$-good. For fix $2^{\delta(m_{i+1}-m_i)/2}$ bad strings for such a block; even allowing all 2^{m_i} strings to be bad for the initial segment causes $2^{m_i+\delta(m_{i+1}-m_i)/2} \leq 2^{\delta m_{i+1}}$ possibilities for bad strings in the initial segment of length m_{i+1}. Thus, we can force a δ-good initial segment of length m_{i+1} to lie outside such a set.

Again to ease the comparison to previous lemmas, redefine δ as $\delta/2$. Once we have δ-good, disjoint blocks, we can select the smaller size blocks that we will need by the same process of picking sub-blocks in a pairwise independent manner. Our algorithm needs good blocks of sizes $cr, cr(1-\delta/8), cr(1-\delta/8)^2, \ldots, m_0 = O(\log r/\delta^2)$; we rewrite these block lengths as approximately $l_i = m_0(1-\delta/8)^{-i}$. Our algorithm needs $O(1/\delta)$ good blocks of each size. Picking in the pairwise independent manner, the probability that a fraction $1-(1-\delta/8)^i/k\log r$ of the blocks of size l_i are $\delta^2/36$-good is at least $1-1/r$. Thus, with high probability a fraction $1-O(1/\delta)\sum(1-\delta/8)^i/\log r = 1-O(1/\delta^4 \log r)$ of the sequences of different size blocks contain all good blocks. Because Algorithm B using only good blocks always outputs the right answer, we have:

Theorem 2 *For all $\delta > 0$, there is an algorithm that simulates BPP using a δ-source.*

6 The Difficulty of Computing MAX CLIQUE

In this section our main theorem is:

Theorem 3 *If for any constant t there is a quasi-polynomial time algorithm that always outputs a number between $\alpha^{1/t}$ and α^t, then $N\tilde{P} = \tilde{P}$.*

By replacing t by t^2, observe that this is equivalent to always outputting a number between α and α^t, which is really the formulation we use. Our proof closely follows the proof of [FG+], making great use of the proof in [BFL] that NEXP = MIP. Actually, the fact that we really need is that any language in NEXP is accepted by a polynomial time probabilistic oracle machine, which is equivalent to NEXP = MIP [FRS].

Definition 10 *A language L is accepted by a probabilistic oracle machine M iff*

$$x \in L \Rightarrow (\exists \text{oracle } O) Pr_r[M^O(x,r) = 1] = 1$$
$$x \notin L \Rightarrow (\forall \text{oracles } O) Pr_r[M^O(x,r) = 1] < 1/4.$$

Denoting the maximum number of random and communication bits used on inputs of size n as $r(n)$ and $c(n)$, respectively, Feige et.al. give the following improvement of [BFL]:

Theorem 4 (FG+) *Any language L in $NTIME(T(n))$ (for $n \leq T(n) \leq 2^{poly(n)}$) is accepted by a probabilistic oracle machine running in time $T(n)^{O(\log\log T(n))}$ and having $r(n) + c(n) = O(\log T(n) \log\log T(n))$.*

Using this, they construct a graph G_x which has a large clique iff $x \in L$. In order to do this, they define transcripts and a notion of consistency among them. A transcript is basically a set of questions to and answers from the oracle; two transcripts are consistent if the oracle is consistent:

Definition 11 (FG+) *A string $t = r, q_1, a_1, \ldots, q_l, a_l$ is a transcript of a probabilistic oracle machine M on input x if $|r| = r(n), |q_1, a_1, \ldots, q_l, a_l| \leq c(n)$, and for every i, $q_i = M(x, r, < q_1, a_1, \ldots, q_{i-1}, a_{i-1} >)$. A transcript is accepting if M on input x, random string r, and history of communiction (questions and answers) $< q_1, a_1, \ldots, q_l, a_l >$ accepts x.*

Definition 12 (FG+) *Two transcripts $t = r, q_1, a_1, \ldots, q_l, a_l$ and $\hat{t} = \hat{r}, \hat{q_1}, \hat{a_1}, \ldots, \hat{q_l}, \hat{a_l}$ are consistent if for every i, $q_i = \hat{q_i}$ implies $a_i = \hat{a_i}$.*

We can now define G_x: the vertices are all accepting transcripts, and two nodes are connected iff the corresponding transcripts are consistent. It is then not hard to see:

Lemma 12 (FG+) $\max_O Pr_r[M^O(x,r) = 1] \cdot 2^{r(n)} = \alpha(G_x)$.

Thus, if $x \in L$ then $\alpha(G_x) = 2^{r(n)}$ and if $x \notin L$ then $\alpha(G_x) < 2^{r(n)}/4$.

To get the second part of their theorem, Feige et.al. construct the graph G'_x corresponding to a protocol M'. M' runs $\log^{O(1)} T(n)$ independent iterations of M on x. This reduces the error probability if $x \notin L$ and therefore produces a wider separation in the clique sizes.

Yet once we fix an oracle O, M^O basically corresponds to a co-RP machine: always accepting when $x \in L$ and usually rejecting if $x \notin L$. Thus we can apply Theorem 1, which we rephrase here:

Theorem 5 (Z) *Let $\epsilon > 0$ be given. Then we can choose a constant c such that the following holds. Let A be any co-RP algorithm that uses r bits to reduce the error probability to $1/4$ if $x \notin L$. Then for $s = cr\log r$ there is an algorithm that uses s random bits, produces $m = poly(r)$ r-bit strings $y_1, \ldots, y_m$ such that if $x \notin L$, the probability that A accepts all y_i is at most $2^{-(1-\epsilon)s}$.*

Proof of Theorem 3: Take $\epsilon = 1/t$ and form M' by running M on x with the random strings $y_1, \ldots, y_m$. Construct G'_x as above. Observe that if $x \in L$ then $\alpha(G_x) = 2^s$, and if $x \notin L$ then $\alpha(G_x) < 2^{s/t}$. Thus any algorithm that always outputs a number between $\alpha(G_x)$ and $\alpha(G_x)^t$ can always tell whether or not $x \in L$. Since G'_x has at most $2^{m(r(n)+c(n))} = 2^{polylog(T(n))}$ vertices, this yields the theorem.

7 Implicit $O(1)$ Probe Search

As we mentioned earlier, Fiat and Naor showed a correspondence between the existence of an $O(1)$ probe search scheme and the constructibility of a certain combinatorial structure called a rainbow:

Definition 13 *A (c, m, n, t)-rainbow is a coloring of all t-tuples (without repetitions) of elements in $\{1, \ldots, m\}$ with c colors, so that for any subset $S \subset \{1, \ldots, m\}$, $|S| = n$, all c colors appear in the t-tuples over S.*

Theorem 6 (FN) *Let $c = \max(n, \log m)$. The existence of a $(c, m, n, t = O(1))$-rainbow yields an implicit $O(1)$ probe search scheme for n elements from the domain $\{1, \ldots m\}$. Conversely, given an implicit $O(1)$ probe search scheme for n elements chosen from the domain $\{1, \ldots, m\}$, an $(n, m, n, t = O(1))$-rainbow can be constructed.*

Thus Fiat and Naor prove their main theorem by constructing an $(n, n^k, n, O(1))$-rainbow for any constant k. Moreover, their proof implies the following useful lemma:

Lemma 13 (FN) *If p_1, p_2, p_3 are polynomials and m is a function of n such that a $(c, m, n, O(1))$-rainbow can be constructed, then a $(p_1(c), p_2(m), p_3(n), O(1))$-rainbow can be constructed.*

Fiat and Naor [FN] further show how to use dispersers to construct rainbows:

Definition 14 *An (m, n, d, a, b)-disperser is a bipartite graph with m nodes on the left side, each with degree d, and n nodes on the right side, such that every subset of a nodes on the left side is connected to at least b nodes on the right.*

Sipser [Sip] showed that $(m, n, \log^2 n, n, n/2)$-dispersers exist, and conjectured that they could be explicitly constructed. Fiat and Naor [FN] show that under this conjecture, an $(n, m = n^{\log n}, n, O(1))$-rainbow can be constructed. Indeed, this construction is an immediate corollary of the following two lemmas:

Lemma 14 (FN) *Given a $(d, m, n, O(1))$-rainbow and an $(m, c, d, n, c/2)$-disperser, a $(c, m, n, O(1))$-rainbow can be constructed.*

Lemma 15 (FN) *For $m = n^{polylog(n)}$, there exists an explicit construction for a $(\log n, m, n, O(1))$-rainbow.*

We show how to use these two lemmas with our own disperser constructions to construct a rainbow with m superpolynomial in n. First we note the connection between the efficient construction of dispersers and the simulation of RP using δ-sources:

Lemma 16 (CWi) *For $\delta' > \delta$, RP can be simulated using s bits from a δ'-source iff a $(2^s, 2^r, poly(r), 2^{\delta s}, 2^{r-1})$-disperser can be efficiently constructed.*

Rewriting our results on simulating RP:

Theorem 7 *For all $n, \delta = \delta(n)$, $(m(n,\delta) = n^{O(\log\log n/\delta^3)}, n, (\log n)^{O(\frac{1}{\delta^2}\log\frac{1}{\delta})}, m(n,\delta)^\delta, n/2)$-dispersers can be explicitly constructed.*

We also need that such dispersers exist for larger m:

Corollary 2 *For all $n, \delta = \delta(n)$ and all $m \geq m(n,\delta)$ $(m, n, (m/m(n,\delta))(\log n)^{O(\frac{1}{\delta^2}\log\frac{1}{\delta})}, m^\delta, n/2)$-dispersers can be explicitly constructed.*

Proof. Say $m = m(n,\delta)^k$. Express elements in $\{1, \ldots, m\}$ as elements of $\{1, \ldots, m(n,\delta)\}^k$. Connect $(v_1, \ldots, v_k)$ to all of the neighbors of each of $v_1, \ldots, v_k$ in the original disperser given by Theorem 7. Because for any set S of size m^δ, there must exist some component i such that S takes on at least $(m^\delta)^{1/k} = m(n,\delta)^\delta$, the corollary follows. □

Nevertheless, using these dispersers and applying Lemma 14 to Lemma 15 does not yield any improvements immediately. Rather, we must apply Lemma 14 twice, construct a new disperser, and apply Lemma 14 again. First, we apply Lemma 14 twice:

Lemma 17 *There is an explicit construction for an $(n, m = n^{\log\log n/\delta^3}, m^\delta, O(1))$-rainbow, for $\delta = \log\log\log n/\sqrt{\log\log n}$.*

Proof. Construct two dispersers using Corollary 2 and Theorem 7, respectively, both with the same m and $\delta = \log\log\log n/\sqrt{\log\log n}$. For the first, use an n in Corollary 2 (which we will call n_1 so as not to confuse with the n in this lemma) equal to $n_1 = 2^{(\log\log n)^2/\log\log\log n}$. Then apply Lemmas 14 and 13 to construct an $(n_1, m, m^\delta, O(1))$-rainbow. Next construct a disperser from Theorem 7 with n as in the statement of this lemma. Applying Lemmas 14 and 13 completes the proof.

□

The new disperser construction we will need is:

Lemma 18 *For all n, δ, $(m = n^{O(1/\delta)}, n, 2^{O(\sqrt{\frac{\log n}{\delta}\log\log\frac{1}{\delta}})}, n/2)$-dispersers can be explicitly constructed.*

Proof. Let $r = \lg n$. We wish to find an r-bit witness using the optimal number of random bits from a δ-source, namely $O(r/\delta)$, but we are allowed a lot of time, i.e. we can try $2^{O(s)}$ possibilities, where $s = \sqrt{\frac{r}{\delta}\log\log\frac{1}{\delta}}$.

The key idea is to imagine we have an additional $O(s)$ independent and uniformly random bits – in reality we will try all $2^{O(s)}$ possibilities – and use the Leftover Hash Lemma. A first try would be to say a fraction $\delta/2$ of the s-bit blocks from the δ-source are $\delta/2$-good; if we hash these good blocks down to $\delta s/2$ bits each and concatenate them together, we will have a quasi-random string. The problem with this is that it loses two factors of $\delta/2$, one because only a fraction $\delta/2$ of the blocks are good, the other because we have to hash down by a factor of $\delta/2$. This would require $O(r/\delta^2)$ bits from the δ-source; we are allowed only $O(r/\delta)$.

Intuitively, we were throwing away alot of our randomness before, because either many blocks which were, say, 1/2-good, and we assumed they were only $\delta/2$-good, or, say, 1/2 the blocks were $\delta/2$-good, and we assumed only a fraction $\delta/2$ were. We could be more efficient by assigning guesses about the goodness of each block, say among the values $0, \delta/4, \delta/2, \delta, \ldots, 2^k\delta$, where $k = \lfloor \lg 1/\delta \rfloor$. For one such set of guesses, each guess will either be between 1/2 and 1 times the actual "goodness," or else only $\delta/4$ below the actual goodness (because the goodness is in $(0, \delta/4)$ and we guessed 0). The second part causes us to lose at most $\delta/4$ of the goodness, and the first at most another constant factor, so this should keep us from losing much randomness. This is formalized in the following lemma:

Lemma 19 *Suppose we ask for bl bits from a δ-source, viewed as b blocks of length l. Assume $2^{\delta l/2} > k+4$, where $k = \lfloor \lg 1/\delta \rfloor$. Then there exists a sequence $\{\delta_i\}_{i=1}^b$, $\delta_i \in \{0, \delta/4, \delta/2, \ldots, 2^k\delta\}$, such that the ith block is δ_i-good and $\sum_{i=1}^m \delta_i \geq \delta b/4$.*

Proof. Construct a tree of initial sequences X_i, as in Lemma 2, and let $L(X_i)$ denote the number of leaves underneath X_i. We prove our lemma by induction on the statement: there exists an initial sequence X_i and values $\{\delta_j\}_{j=1}^i$ so that for all $j \leq i$, the jth block is δ_j-good, and

$$L(X_i) \geq 2^{(\delta b - 2(\sum_{j=1}^i (\delta_j + \delta/4)))l}.$$

This will prove the lemma because $L(X_b) = 1$ so

$$\delta b - 2(\sum_{j=1}^b (\delta_j + \delta/4)) \leq 0,$$

and therefore $\sum_{j=1}^b \delta_j \geq \delta b/4$.

The base case of the induction is clear. For the inductive step, assume it holds for a given i, but not for $i+1$. Let $\alpha = \delta b - 2(\sum_{j=1}^i (\delta_j + \delta/4))$. Then all strings x_{i+1} must correspond to initial sequences X_{i+1} with $L(X_{i+1}) < 2^{(\alpha - \delta/2)l}$; all $\delta/4$-good strings x_{i+1} correspond to initial sequences X_{i+1} with $L(X_{i+1}) < 2^{(\alpha-\delta)l}$; in general, all $2^t\delta$-good strings correspond to initial sequences with $L(X_{i+1}) < 2^{(\alpha - 2^{t+1}\delta - \delta/2)l}$. Denoting by B the set of $2^t\delta$-bad strings, we see that the maximum number of leaves we can have is

$$\sum_{t=-2}^{k+1} |B| \max\{L(X_{i+1}) | x_{i+1} \in B\}$$

$$\leq \sum_{t=-2}^{k+1} 2^{2^t\delta l} 2^{(\alpha - 2^t\delta - \delta/2)l} = (k+4)2^{(\alpha-\delta/2)l} < 2^{\alpha l},$$

contradicting our inductive assumption.

□

Thus, we view our s random bits as a hash function h mapping s bits to s bits (we will always map to fewer than s bits, but we ensure there are enough bits in the description of h). We divide our $8r/\delta$-bit string from the δ-source into $8r/\delta s$ blocks of s bits each. For each sequence $\{\delta_i\}$, $\sum \delta_i \geq 2r$, $\delta_i \in \{0, \delta/4, \delta/2, \ldots, 2^k\delta\}$, we assume the ith block x_i is δ_i-good. Lemma 19 implies that for one such choice of the δ_i the assumption will be true. We then use h to hash x_i to $\delta_i s/2$ bits. The Leftover Hash Lemma, along with Lemma 4, inductively implies that the sequence $h(x_1) \circ h(x_2) \circ \ldots \circ h(x_{8r/\delta s})$ is quasi-random within $\sum 2^{-\delta_i s/4} << 1$. Thus, with non-zero probability this string lies in any witness set of size $n/2$, i.e. it does so for at least one choice of h and δ_i-good x_i.

The total number of possibilities we have to try is

$$(\#\text{ possible } h\text{'s})(\#\text{ possible sequences } \{\delta_i\})$$

$$= 2^{O(s)}(k+4)^{O(r/\delta s)} = 2^{O(s)},$$

as required.

□

We can now prove our main result:

Theorem 8 *There is an explicit construction for an $(n, m = n^{\sqrt{\log\log n}/\log\log\log n}, n, O(1)$-rainbow, and hence implicit $O(1)$-probe search can be done for this value of m.*

Proof. Using Lemma 17, build a $(c = 2^{O(\sqrt{\log m \log\log\frac{1}{\delta}})}, m, n = m^\delta, O(1))$-rainbow, for $\delta = \log\log\log n/\sqrt{\log\log n}$. Note that Lemma 17 allows for a larger value of c, but we can always use a smaller one. Using Lemma 13 to show that we can ignore the big O's

in the exponents, and Lemma 14 to combine this rainbow with the disperser of Lemma 18 yields the theorem. □

Acknowledgements

I am grateful to Abhijit Sahay and Madhu Sudan for finding the mistake in my earlier proof, as well as to Umesh Vazirani, Madhu Sudan, and Moni Naor for other valuable discussions.

References

[Blu] M. Blum, "Independent Unbiased Coin Flips from a Correlated Biased Source: a Finite Markov Chain," *Combinatorica*, 6 (2): 97-108, 1986.

[BBR] C.H. Bennett, G. Brassard, and J.-M. Robert, "Privacy Amplification by Public Discussion," *SIAM Journal on Computing*, 17 (2): 210-229, 1988.

[BL] M. Ben-Or and N. Linial, "Collective Coin Flipping Robust Voting Schemes and Minimal Banzhaf Values," 26th FOCS, 1985, pp. 408-416.

[CWe] L. Carter and M. Wegman, "Universal Hash Functions," *J. Comp. and Syst. Sci.*, 18(2): 143-154, 1979.

[CG1] B. Chor and O. Goldreich, "On the Power of Two-Points Based Sampling," Technical Report, MIT Laboratory fo Computer Science.

[CG2] B. Chor and O. Goldreich, "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity," *SIAM J. Comput.*, 17(2): 230-261, 1988.

[CG+] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolenski, "The Bit Extraction Problem or t-Resilient Functions," 26th FOCS, 1985, pp. 396-407.

[CWi] A. Cohen and A. Wigderson, "Dispersers, Deterministic Amplification, and Weak Random Sources," 30th FOCS, 1989, pp. 14-19.

[DFK] M. Dyer, A. Frieze, and R. Kannan, " A random polynomial time algorithm for approximating the volume of convex bodies," 21st STOC, 1989, pp. 375-381.

[FG+] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, M. Szegedy, "Approximating Clique is Almost NP-Complete." These FOCS Proceedings.

[FN] A. Fiat and M. Naor, "Implicit $O(1)$ Probe Search," 21st STOC, 1989, pp. 336-344.

[FNSS] A. Fiat, M. Naor., J.P. Schmidt, and A. Siegel, "Non-Oblivious Hashing," 20th STOC, 1988, pp. 367-376.

[FKS] M.L. Fredman, J. Komlos, and E. Szemeredi, "Storing a Sparse Table with $O(1)$ Worst Case Access Time," *Journal of the Association for Computing Machinery*, 31: 538-544, 1984.

[ILL] R. Impagliazzo, L. Levin, and M. Luby, "Pseudo-Random Generation from One-Way Functions," 21st STOC, 1989, pp. 12-24.

[IZ] R. Impagliazzo and D. Zuckerman, "How to Recycle Random Bits," 30th FOCS, 1989, pp. 248-253.

[K] R.M. Karp, "Reducibility Among Combinatorial Problems. In R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations*, 1972, pp. 85-103.

[LLS] D. Lichtenstein, N. Linial, and M. Saks, "Imperfect Sources of Randomness and Discrete Controlled Processes," 19th STOC, 1987, pp. 169-177.

[MNT] Y. Mansour, N. Nisan, and P. Tiwari, "The Computational Complexity of Universal Hashing," 22nd STOC, 1990, pp. 235-243.

[Rab] M. O. Rabin, "Probabilistic Algorithm for Testing Primality," *Journal of Number Theory*, 12: 128-138, 1980.

[SS] A. Sahay and M. Sudan, personal communication.

[San] M. Santha, "On Using Deterministic Functions in Probabilistic Algorithms," *Information and Computation*, 74(3): 241-249, 1987.

[SV] M. Santha and U. Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources," 25th FOCS, 1984, pp. 434-440.

[Sho] V. Shoup, "New Algorithms for Finding Irreducible Polynomials over Finite Fields," 29th FOCS, 1988, pp. 283-290.

[Va1] U. Vazirani, "Efficiency Considerations in Using Semi-Random Sources," 19th STOC, 1987.

[Va2] U. Vazirani, "Randomness, Adversaries and Computation," PhD Thesis, University of California, Berkeley, 1986.

[Va3] U. Vazirani, "Strong Communication Complexity or Generating Quasi-Random Sequences from Two Communicating Semi-Random Sources," *Combinatorica*, 7 (4): 375-392, 1987.

[VV] U. Vazirani and V. Vazirani, "Random Polynomial Time is Equal to Semi-Random Polynomial Time," 26th FOCS, 1985, pp. 417-428.

[vN] J. von Neumann, "Various Techniques Used in Connection with Random Digits," Notes by G.E. Forsythe, National Bureau of Standards, *Applied Math Series*, 12:36-38, 1951. Reprinted in *Von Neumann's Collected Works*, 5:768-770, 1963.

[Z] D. Zuckerman, "General Weak Random Sources," 31st FOCS, 1990, pp. 534-543.