Electronic Colloquium on Computational Complexity, Report No. 62 (2005)

2-Local Random Reductions to 3-Valued Functions

A. Pavan* N. V. Vinodchandran†

Abstract

Yao (in a lecture at DIMACS Workshop on structural complexity and cryptography) showed that if a language L is 2-locally-random reducible to a *Boolean function*, then $L \in$ PSPACE/`poly`. Fortnow and Szegedy quantitatively improved Yao's result to show that such languages are in fact in NP/`poly` (*Information Processing Letters*, 1992).

In this paper we extend Yao's result to show that if a language L is 2-locally-random reducible to a target function which takes values in $\{0, 1, 2\}$, then $L \in$ PSPACE/`poly`.

1 Introduction

Informally a language L is *locally-random reducible* (in short *lrr*) to a target function f if, for any input x, membership of x in L can be efficiently reduced to k random instances of f. If the reduction makes k queries then it is called a k-local random reduction. Additionally if the target function is the language L itself then it is called a *random self-reduction*. Such random reducibility notions have been very useful in many areas of theoretical computer science including, complexity theory, cryptography, and private information retrieval. Please see [Fei93] for a survey on local random reductions and their applications.

A natural question that arises is: for a language L, is there a function f so that L locally-random reduces to f? Beaver and Feigenbaum [BF90] showed that this is true: any language L is $(n+1)$-locally-random reduces to a specific function f. Beaver, Feigenbaum, Kilian, Rogaway [BFKR97] improved this result to show that in fact, for any language L, there is a function f so that L $(n/\log n)$-locally random reduces to f. In these constructions the range of the target function is very large (in fact $\Omega(2^n)$).

Are there languages that are not k-lrr to any function, for a constant k? Abadi, Feigenbaum, and Kilian [AFK89] showed that a language that is 1-lrr to any function is in NP/`poly`. For $k > 1$, Yao [Yao90] showed that any language that is 2-locally-random reducible to a Boolean function is in PSPACE/`poly`. Fortnow and Szegedy extended Yao's result to show that such languages are in fact in NP/`poly` $\cap$ co-NP/`poly` [FS92]. Regarding random self reductions, Feigenbaum, Kannan, and Nisan [FKN90] showed that any function (need not be a language/Boolean function) that is uniform 2-random self-reducible is in fNP/`poly` (fNP is the functional version of NP). Feigenbaum and Fortnow [FF93] showed that if an NP-complete set such as SAT is `poly`(n)-random self-reducible with nonadaptive queries, then the Polynomial Hierarchy collapses.

*Department of Computer Science, Iowa State University, `pavan@cs.iastate.edu`. Research supported in part by NSF grants CCR-0344817 and CCF-0430807.

†Department of Computer Science and Engineering, University of Nebraska-Lincoln, `vinod@cse.unl.edu`. Research supported in part by NSF grant CCF-0430991, Big 12 Fellowship, and University of Nebraska Layman Award.

ISSN 1433-8092

It appears that extending the results of Yao, and Fortnow and Szegedy for $k > 2$ is difficult. For instance a lower bound for the case $k > 2$ will lead to a better lower bound for the length of certain locally decodable codes. We discuss this in some detail in Section 3. In this paper we consider the following question: can we extend the results of Yao, and Fortnow and Szegedy, for target functions other than Boolean? Building the work of Yao, and Fortnow and Szegedy, we show that languages outside PSPACE/`poly` are not 2-locally-random reducible to any 3-valued function. We show the following theorem.

Theorem 1 (Main Theorem). *If a language L is 2-locally-random reducible to functions g and h that take values in $\{0,1,2\}$, then $L \in$ PSPACE/`poly`.*

2 Main Result

Definition 1 (locally-random reduction). A language L is k-locally-random reducible (k-*lrr*) to functions $g_1, \ldots, g_k$ if there are polynomial-time functions σ and f and a polynomial q so that

- $\forall x \in \{0,1\}^*, \forall r \in \{0,1\}^{q(|x|)}$, $L(x) = f(g_1(\sigma(1,x,r)), \ldots, g_k(\sigma(k,x,r)), x, r)$
- for all i, $\sigma(i,x,r)$ and $\sigma(i,y,r)$ are identically distributed when $|x| = |y|$ and $r \in \{0,1\}^{q(|x|)}$ is chosen uniformly at random.

Now we will state and prove our main result.

Theorem 1 *If a language L is 2-locally-random reducible to functions g and h that take values in $\{0,1,2\}$, then $L \in$ PSPACE/`poly`.*

Before giving the formal proof we give the proof idea. For this informal proof description, we assume that the target functions g and h are the same. We first revisit the idea behind Yao-Fortnow-Szegedy's proof.

Let L be a language that is 2-lrr to a Boolean function g. Given a string x, let p and q be two queries produced by the reduction using a random string r. Suppose the membership of x in L depends only on p, and does not depends on q. Then by knowing $g(p)$ we can decide x. In this case, we say p *sets* x. Suppose p does not set x, this means the membership of x in L depends on both p and q. The crucial observation is the following: When p does not set x, given $g(p)$, whether x belongs to L or not precisely depends on whether $g(q)$ is 0 or 1. Thus by knowing $g(p)$, and $L(x)$ we can deduce the value of $g(q)$. We call this scenario p *forces* q *via* x. Now, the proof Yao-Fortnow-Szegedy goes as follows. Let $P = \{p_1, \cdots p_m\}$ be a set of queries whose answers we know, i.e, we know the values of $g(p_i), 1 \leq i \leq m$. Let x be the input string. If there exists a $p \in P$ that sets x then we can decide the membership of x in L easily. On the other hand if no p in P sets x, then for every p there exists a query q such that p forces q via x. Thus by giving $L(x)$ as advice we can compute the value of $g(q)$ for all these q's. Thus the set of queries whose answers we know has doubled. Applying this argument iteratively we can decide answers to all the queries by specifying $L(x)$ for a polynomially small set of x's. Using this idea it follows that L is in NP/*poly*.

Observe that this proof does not go through if f is a 3-valued function. If p does not set x, we can not claim p forces q via x. To get around this problem, we do the following: Suppose p does not set x. By considering all three possible values (0, 1, and 2) for $g(q)$, we compute possible values for $L(x)$. Since $L(x)$ has only two possible values, there exist two possible values of $g(q)$ that would

say $L(x) = b$, and the remaining possibility for $g(q)$ says $L(x) = \overline{b}$. Thus majority of choices for $g(q)$ say $L(x) = b$. Now we assume $L(x) = b$ and proceed. If it really happens that $L(x) = \overline{b}$, then we provide $L(x)$ as advice. We observe that when this majority choice is wrong, $g(p)$ and $L(x)$ together force $g(q)$ to a unique value. Thus our strategy is: If p does not set x, then compute the choice for $L(x)$ by taking a majority vote. For all x's for which the majority is wrong, we provide $L(x)$ as advice. We show that this idea can be implemented to get a PSPACE/`poly` algorithm for L.

Now we present a formal proof.

Proof. Since L is 2-locally-random reducible to g and h, there exist polynomial-time computable functions f, σ such that for every x,

$$\forall r, f(g(\sigma(1,x,r)), h(\sigma(2,x,r)), x, r) = L(x).$$

Moreover, for every x, y, $|x| = |y|$, the random variables $\sigma(1,x,r)$ and $\sigma(1,y,r)$ are identically distributed, and the random variables $\sigma(2,x,r)$ and $\sigma(2,y,r)$ are also identically distributed.

We will first introduce some notation that we use for the rest of the proof. $\sigma(1,x,r)$ and $\sigma(2,x,r)$ will be denoted by $\sigma_1(x,r)$ and $\sigma_2(x,r)$ respectively. We will use p's to denote the first queries and q's to denote the second queries.

Given x, let $P(x)$ denote the multi-set of all possible first queries, and $Q(x)$ denote the multi-set of all possible second queries. That is $P(x) = \{\sigma_1(x,r) | r \in \{0,1\}^{q(|x|)}\}$ and $Q(x) = \{\sigma_2(x,r) | r \in \{0,1\}^{q(|x|)}\}$. We stress that both $P(x)$ and $Q(x)$ are multi-sets.

Without loss of generality, we assume that we can easily distinguish first queries from second queries. Let m be a bound on the length of queries in $P(x)$ and $Q(x)$. Since, σ_1, and σ_2 are polynomial time-computable, $m = \mathtt{poly}(n)$. Since L is 2-locally random reducible to g and h, if x and y are of same length, then $P(x) = P(y)$ and $Q(x) = Q(y)$. We will denote $P(0^n)$ with P_n and $Q(0^n)$ with Q_n.

Given a multi set A consisting of zeros and ones, let $\mathtt{MAJ}(A) = 1$ if at least half the members of A are 1, else $\mathtt{MAJ}(A) = 0$.

Definition 2. For an x and r, let $p = \sigma_1(x,r)$.

- We say p *sets* x if the membership of $L(x)$ can be decided by knowing only $g(p)$, i.e.,

$$f(g(p),0,x,r) = f(g(p),1,x,r) = f(g(p),2,x,r) = L(x).$$

- We say p *and* x *force* q if $h(q)$ can be computed by knowing $g(p)$ and $L(x)$. I.e., there exists unique $b \in \{0,1,2\}$ such that

$$f(g(q),b,x,r) = L(x).$$

- We say that p *weakly sets* x if $\mathtt{MAJ}(\{f(g(p),0,x,r), f(g(p),1,x,r), f(g(p),2,x,r)\}) = L(x)$.

Similarly we can define setting, forcing, and weak setting for second query $q = \sigma_2(x,r)$.

Observation: Note that if p does not set or weakly set x then p and x force q.

For each n we will construct a "query tree" T_n. Each node of the tree is labeled with a query from the multi sets P_n or Q_n and each level of the tree will be associated with a string $x \in \{0,1\}^n$. The tree will be such that by knowing $L(x)$ for each of these x's, we will be able to get answers to any query in the query tree using a PSPACE procedure.

We now give the construction of T_n. Consider strings in $\{0,1\}^n$ in the lexicographic order. For a string x, $x+1$ denotes string which follows x in this ordering. Let $\perp$ be a special string and assume $\perp < x$ for every x.

CONSTRUCTION Level 1
1. Fix a query p_0 from P_n and label the root of the tree with p_0

Let $x_0 = \perp$. Assume that we have built k levels of the tree and let $x_1, x_2, \cdots, x_{k-1}$ be the strings associated with each level of the tree. We now define the tree at level $k+1$ and associate a string x_k with level k. Define the following multi-sets.

$$Q'_k = \{q \mid q \in Q_n, \text{ and } q \text{ is at level } k\},$$

$$P'_k = \{p \mid p \in P_n, \text{ and } p \text{ is at level } k\}.$$

If $x_{k-1} = 1^n$ (we have exhausted all the input strings), or $Q'_k = Q_n$ and $P'_k = P_n$ (we have exhausted all the queries), then T_n is completely defined. Else, we construct level $k+1$ by the following procedure.

CONSTRUCTION Level $k+1$
1. $x \leftarrow x_{k-1} + 1$
2. If either (a). There exists a query q at level k that sets x
   (b). There exists r such that both $\sigma_1(x,r)$ and $\sigma_2(x,r)$ in level k
   then $x \leftarrow x+1$ and Goto Step 2
3. For each $p \in P'_k$, let r_p be the lexicographically least r such that $\sigma_1(x, r_p) = p$
   For each $q \in Q'_k$, let r_q be the lexicographically least r such that $\sigma_2(x, r_q) = q$
   Let $a_p = \texttt{MAJ}\{f(g(p), 0, x, r_p), f(g(p), 1, x, r_p), f(g(p), 2, x, r_p)\}$
   $a_q = \texttt{MAJ}\{f(0, h(q), x, r_q), f(1, h(q), x, r_q), f(2, h(q), x, r_q)\}$
4. Consider the multi set $A = \{a_q \mid q \in Q'_k\} \cup \{a_p \mid p \in P'_k\}$
   If $\texttt{MAJ}(A) = L(x)$, then $x \leftarrow x+1$ and Goto Step 2
5. Else /* `define nodes at level` $k+1$`, and associate a string with level` k */
   Associate the string x with level k
   For every $v \in Q'_k \cup P'_k$ such that $a_v = L(x)$, q has only one child with label v
   For each $p \in P'_k$ such that $a_p \neq L(x)$, p has two children with labels p and $\sigma_2(x, r_p)$
   For each $q \in Q'_k$ such that $a_q \neq L(x)$, q has two children with labels $\sigma_1(x, r_p)$ and q

We now show some properties of the tree T_n. Let x_k be the node associated with level k. We first start with the following observation.

Claim 1. *Let $p \in P'_k$, let $q = \sigma_2(x_k, r_p)$. If $a_p \neq L(x_k)$, then x_k and p force q. Similarly, let $q \in Q'_k$, let $p = \sigma_1(x_k, r_q)$. If $a_q \neq L(x_k)$, then x_k and q force p.*

Proof. Let

$$M = \{f(g(p), 0, x_k, r_p), f(q(p), 1, x_k, r_p), f(g(p), 2, x_k, r_p)\}.$$

Since x_k is the string associated with level k, p does not set x_k. Thus, all the elements in the multi-set M can not be the same. Since $a_p = \texttt{MAJ}(M) \neq L(x_k)$, there exists a unique $b \in \{0,1,2\}$

such that $L(x_k) = f(g(p), b, x_k, r_p)$. Thus knowing the values of $L(x_k)$ and $g(p)$ gives the value of $h(q)$. Thus x_k and p force q. Proof of the the second part of the claim is identical. □

Let S_k be the multi-set consisting of all nodes at level k. Next we show that size of S_{k+1} is considerably larger than the size of S_k.

Claim 2. $\forall k, |S_{k+1}| \geq \frac{3}{2}|S_k|$.

Proof. Let x_k be the string associated with level k. Step 5 of the above construction defines the nodes in level $k+1$. Observe that every node at level k has either one or two children. Every $q \in Q'_k$, for which $a_q = L(x_k)$, has exactly one child, and every $q \in Q'_k$, for which $a_q \neq L(x_k)$, q has exactly two children. Similarly, every $p \in P'_k$, for which $a_p = L(x_k)$, has exactly one child, and every $p \in P'_k$, for which $a_p \neq L(x_k)$, p has exactly two children.

By definition of x_k, we know $L(x_{k+1}) \neq \texttt{MAJ}(A)$, where $A = \{a_q \mid q \in Q'_k\} \cup \{a_p \mid p \in P'_k\}$. Thus for more than half nodes v in level k, $a_v \neq L(x_k)$. All such nodes have exactly two children. Thus $|S_{k+1}| \geq \frac{3}{2}|S_k|$. Note that here we use the fact that the sets S_k, S_{k+1}, P'_k, and Q'_k are multi sets. □

Corollary 1. *Depth of T_n is* $\texttt{poly}(n)$.

Claim 3. *Let u be a first query, and let k be a level at which u appears. Given $g(u)$ and $L(x_k)$, there is a* PSPACE *algorithm that computes the children of p. Moreover, for each child v of u, $g(v)$ ($h(v)$ if v is a second query) can be computed in* PSPACE. *A similar claim holds if u is a second query.*

Proof. Let k be a level at which u appears. Compute the least r such that $\sigma_1(x_k, r) = u$. Let $v = \sigma_2(x_k, r)$. Note that r can be computed in PSPACE and given r, v can be computed in polynomial-time. Compute

$$a_u = \texttt{MAJ}(\{f(g(u), 0, x_k, r), f(g(u), 1, x_k, r), f(g(u), 2, x_k, r\}).$$

By the construction of T_n, if $a_u = L(x_k)$, u has only one children with label u. In this case we know the value of $g(u)$. Else $a_u \neq L(x_k)$. In this case u has two children with labels u and v. Since $a_u \neq L(x_k)$, by Claim 1, u and x_k force v. Since $g(u)$ and $L(x_k)$ are known, $h(v)$ can be computed. Thus $h(v)$ can be computed in PSPACE. □

Next we claim that given $g(p_0)$, $\langle x_1, L(x_1)\rangle$, $\cdots$, $\langle x_d, L(x_d)\rangle$ where d is the height of the tree T_n, the tree T_n can be traversed in PSPACE. Given a node u, let $value(u) = g(u)$ if u is a first query, otherwise $value(u) = h(u)$.

Claim 4. *Let u a node. Given $g(p_0)$, $\langle x_1, L(x_1)\rangle$, $\cdots$, $\langle x_d, L(x_d)\rangle$ where d is the height of the tree T_n, and a level k, there is a* PSPACE *algorithm checks if u appears at level of T_n. If u appears at level k, then this algorithm computes $value(u)$.*

Proof. Let $\textsc{Reach}(w, value(w), u, s)$ be a subroutine that returns true if u can be reached from w in exactly s steps. Consider the following recursive algorithm.

REACH(w, *value(w)*, u, s)
1 `Current` $\leftarrow w$;
2 For each child v of `Current`
3 Compute $value(v)$;
4 If REACH$(v, value(v), u, s-1) = true$, then return *true*.

By Claim 3, Step 3 can be done in PSPACE. The recursion terminates when $s = 1$. Again by Claim 3, given $L(x_k)$, REACH$(node, value(node), u, 1)$ can be computed in PSPACE. By Claim 3, it follows that $value(u)$ can also be computed in PSPACE. Since the maximum out degree of T_n is 2, the above procedure can implemented in PSPACE. □

Now we are ready to give a PSPACE/`poly` algorithm for L. The algorithm is given the advice $\langle x_1, L(x_1)\rangle, \cdots, \langle x_d, L(x_d)\rangle$, p_0 and $g(p_0)$. Let x be the given input. If x is one of $x_1, \cdots, x_d$, then $L(x)$ can be computed trivially. Let $x_{k-1} < x < x_k$. Since x is not the string associated with level $k+1$, one of the following must hold:

1. There exists a node v at level k that sets x.
2. There exists a r such that $\sigma_1(x, r)$ and $\sigma_2(x, r)$ appear at level k.
3. Majority of nodes at level k weakly set x.

We can check if Statement 1 holds or not as follows: For each r compute $v = \sigma_1(x, r)$. Check if v appears in level k of T_n and if it appears compute $g(v)$. By, Claim 4 this can be done in PSPACE. Now, we can check if v sets x. If none of the first queries set x, check if any of the second queries sets x. If we succeed in finding a v that sets x, then we know $L(x)$.

If Statement 1 does not hold, we can check if Statement 2 holds as follows: For each r compute $u = \sigma_1(x, r)$ and $v = \sigma_2(x, r)$. By Claim 4, in PSPACE, we can find if u and v occur at level k. If so, we can also compute $g(u)$ and $h(v)$ and thus compute $L(x)$. Thus if Statement 2 holds, then also $L(x)$ can be computed in PSPACE.

If Statement 1 and Statement 2 both do not hold, then Statement 3 must hold. For each node v at level k compute a_v, also compute majority of a_v's. This can be done in PSPACE as we can systematically generate each node at level k. Since Statement 3 holds, majority of nodes at level k weakly set x. Thus majority of a_v's is the value of $L(x)$.

Thus L is in PSPACE/`poly`. □

3 Concluding Remarks

Is it possible to extend this tree-based technique to prove similar results for k-lrr for constant $k > 3$? A positive answer to this will improve lower-bounds for certain *locally decodable codes*. We briefly discuss this here.

A code $C : \{0,1\}^n \to \Gamma^m$ is a *k-perfectly-smooth code* if there is a probabilistic oracle algorithm A such that for every $x \in \Sigma^n$ and index i, $1 \le i \le n$, A on input i makes k random queries to $C(x)$ (given as oracle to A) and outputs x_i with probability 1. Moreover, for all i and j, the j^{th} query is uniformly distributed in $\{1, \ldots, m\}$. A k-perfectly-smooth code is also a locally decodable code with certain parameters (if k is a constant). Yao-Fortnow-Szegedy proof can be adapted to show that for a 2-perfectly-smooth code with $|\Gamma| = 2$, the length of the code m should be $2^{\Omega(n)}$ (this was

done by Beigel, Fortnow, and Gasarch [BFG02] in the setting of Private Information Retrieval). Using the techniques of this paper, we can show that for any 2-perfectly-smooth code with $|\Gamma| = 3$, $m = \Omega(2^{n \cdot \log_2 \frac{3}{2}})$. There are no exponential lower bounds known for k-perfectly-smooth codes for $k \geq 3$. Extending this tree-based technique will give an exponential lower bound for such codes.

Some lower bound techniques known for locally decodable codes can be used to prove similar results for local random reductions. In particular, techniques of [KT00, DJK$^+$02] can be used to show that any language that is k-lrr to a function is in PSPACE/$2^{\epsilon n}$ for $\epsilon = 1 - \frac{1}{k}$. These arguments will work even if the target function has range other than Boolean. See an excellent survey paper by Trevisan [Tre04] for more on locally decodable and perfectly smooth codes, and their application to complexity theory.

Finally, the local random reduction that we consider in this paper does not allow errors. This is true for lower bound results in [Yao90, FS92, FKN90]. Extending these techniques to the general nonzero error case is interesting since it may lead to a non-quantum proof of exponential lower bounds for 2-locally-decodable codes: currently the known exponential lower bound proof for 2-locally-decodable codes uses quantum information theory [KdW03].

References

[AFK89] M. Abadi, J. Feigenbaum, and J. Kilian. On hiding information from an oracle. *Journal of Computer and System Sciences*, 39:21–50, 1989.

[BF90] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 22–24 February 1990.

[BFG02] R. Beigel, L. Fortnow, and W. Gasarch. Nearly tight lower bounds for private information retrieval systems. Technical Report 2002-L001N, NEC Laboraties of America, 2002.

[BFKR97] D. Beaver, J. Feigenbaum, J. Kilian, and P. Rogaway. Locally random reductions: Improvements and applications. *Journal of Cryptology*, 10(1):17–36, Winter 1997.

[DJK$^+$02] A. Deshpande, R. Jain, T. Kavitha, J. Radhakrishnan, and S. Lokam. Better lower bounds for locally decodable codes. In *IEEE Conference on Computational Complexity*, pages 184–193, 2002.

[Fei93] J. Feigenbaum. Locally random reductions in interactive complexity theory. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:73–98, 1993.

[FF93] J. Feigenbaum and L. Fortnow. On the random-self-reducibility of complete sets. *SIAM J. Comput.*, 22:994–1005, 1993.

[FKN90] J. Feigenbaum, S. Kannan, and N. Nisan. Lower bounds on random-self-reducibility. In *Proceedings of the Fifth Annual Structure in Complexity Theory Conference*, pages 100–109. IEEE Computer Society Press, 1990.

[FS92] L. Fortnow and M. Szegedy. On the power of two-local random reductions. *Information Processing Letters*, 44(6):303–306, 1992.

[KdW03] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *In 35th Annual ACM Symposium on Theory of Computing*, pages 106–115, 2003.

[KT00] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *In Proc. of 32nd STOC*, pages 80–86, 2000.

[Tre04] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004.

[Yao90] A. Yao. An application of communication complexity to cryptography. In *A Lecture at DIMACS Workshop on Structural Complexity and Cryptography*, 1990.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'