

Notes on “Natural Proofs”  
or  
Why proving  $P \neq NP$  is hard

Ashley Montanaro\*

Department of Computer Science, University of Bristol,  
Woodland Road, Bristol, BS8 1UB, England.

December 13, 2005

## 1 Introduction

Imagine that, like all good computer scientists, you want to prove that  $P \neq NP$ . A “natural” way to go about this is to show that the computation of some function believed not to be in  $P$  (such as solving the discrete logarithm problem) must take more than polynomial time. We might also expect that such a proof would enable us to come up with some easily-computable predicate which we can apply to the truth table of a function to determine whether it’s polynomial-time computable or not. The bad news is that Razborov and Rudich have shown that such a proof cannot exist!

## 2 Definitions

We consider  $n$ -bit Boolean functions  $f_n : \{0, 1\}^n \mapsto \{0, 1\}$ . Set  $F_n = \{f_n\}$ . A *combinatorial property*  $C_n$  is a subset of Boolean functions  $C_n \subseteq F_n$ .

---

\*E-mail: montanar@cs.bris.ac.uk

## 2.1 Natural proofs

$C_n$  is *natural* if it contains a subset  $C_n^*$  (for the rest of these notes,  $C_n^*$  will just be equal to  $C_n$ ) with the following two properties.

**(Constructivity)** The question of whether a function  $f_n \in C_n^*$  can be decided in time polynomial in the size of the truth table, i.e. in time  $2^{O(n)}$ .

**(Largeness)** At least a fraction of  $1/2^{O(n)}$  of the functions in  $F_n$  must be in  $C_n^*$ .

For  $C_n$  to be *useful against P/poly*, we will associate it with the set of superpolynomial Boolean functions, i.e. those whose circuit size is  $> n^k$  for all constant  $k$  and sufficiently large  $n$ . Intuitively, it seems that the “largeness” condition should follow immediately from this, as we might expect most functions to be hard to compute. In fact, we can formalise this (see section 4). Equivalent definitions can be formulated for proofs useful against other circuit complexity classes, but we will only consider P/poly here.

## 3 Natural proofs and random functions

A *pseudo-random function generator* is a polynomial-time computable function  $f : \{0, 1\}^{n^c} \mapsto F_n$ , where  $c > 1$ .  $f$  takes as input an  $n^c$ -bit key  $k$ , and outputs an  $n$ -bit function  $f(k)$ , which is indistinguishable from a random function to any program that runs in time polynomial in the truth table of  $f(k)$ , i.e. in time  $2^{O(n)}$ . In particular, it is impossible to guess the key, as this would require time  $2^{O(n^c)}$ . Using our test  $C_n$ , we can distinguish the output of such pseudo-random function generators from true randomness, i.e.:

$$\left| \Pr_{g \in F_n} [C_n(g) = 1] - \Pr_{k \in \{0,1\}^{n^c}} [C_n(f(k)) = 1] \right| \geq \frac{1}{2^{O(n)}} \quad (1)$$

That is,  $f(k)$  is polynomial-time computable, and thus will never be in the set  $C_n$ . However, by the largeness property, a large proportion of random functions will be in  $C_n$ , and hence we are able to distinguish the two. Note that we’ve used all the properties: constructivity, largeness and usefulness. So, if we have a natural proof, pseudo-random function generators cannot exist. The next stage is to show that we can construct these function generators from one-way functions.

### 3.1 Pseudo-randomness

A *pseudo-random generator*  $G_n : \{0, 1\}^n \mapsto \{0, 1\}^{2n}$  is a polynomial-time computable function that amplifies  $n$  random bits to  $2n$  pseudo-random bits. We define the hardness  $H(G_n)$  as the size of the minimal circuit that can distinguish the output of  $G_n$  from actual randomness.

Formally,  $H(G_n)$  is the minimal  $S$  for which there exists a circuit  $C$  of size  $\leq S$  such that

$$\left| \Pr_{x \in \{0,1\}^n} [C(G_n(x)) = 1] - \Pr_{y \in \{0,1\}^{2n}} [C(y) = 1] \right| \geq \frac{1}{S} \quad (2)$$

We will assume that these pseudo-random generators exist if one-way permutations exist (there is a standard construction due to Blum & Micali). We will now demonstrate that a pseudo-random function generator  $f : \{0, 1\}^{n^c} \mapsto F_n$  can be produced from a pseudo-random generator  $G : \{0, 1\}^n \mapsto \{0, 1\}^{2n}$ . Split the output of  $G$  into two halves, and let  $G_0, G_1$  denote the first and last  $n$  bits respectively. We will produce a pseudo-random function  $f$  by repeating this splitting process, feeding each half back into  $G$ . The final output will be the first bit of the result of the last split. That is, for an input  $y$  and a “key”  $x$ ,  $f(x)(y) = \text{MSB}(G_{y_n} \circ G_{y_{n-1}} \circ \dots \circ G_{y_1}(x))$ .

It remains to show that this function generator is secure. That is, we need to show that if we can distinguish the output from a random function, then we can distinguish the output of the pseudo-random generator from a random string. The idea is to produce a “hybrid” machine that uses part randomness, part pseudo-randomness. If we consider the computation of the pseudo-random function as a tree, the hybrid  $H_k$  injects real randomness in at level  $k$  of the tree (so  $H_0$  is the same as  $f$ , and  $H_n$  is the same as a real random function).

We can see that any machine distinguishing  $H_0$  from  $H_n$  must be able to distinguish  $H_k$  from  $H_{k+1}$  for some  $k$ . Intuitively, this distinguishes a real random string from a fake one at level  $k$ . For a more rigorous construction, see Goldreich: Foundations of Cryptography.

So we finally have the result.  $H(G_{n^c}) \leq 2^{O(n)}$ , and thus:

**Theorem 3.1.** *If  $2^{n^\epsilon}$ -hard pseudo-random generators exist, there is no natural proof against  $P/\text{poly}$ .*

## 4 Formal complexity measures

We can show that the “largeness” criterion in our definition of natural proofs is automatic for all “reasonable” measures of circuit complexity. We define a formal complexity measure  $\mu(f)$  as a function mapping the truth table of every  $n$ -bit function to a non-negative integer, where  $\mu(x_i) \leq 1$  and  $\mu(\bar{x}_i) \leq 1$  for all  $i$ , and also

$$\mu(f \vee g) \leq \mu(f) + \mu(g) \tag{3}$$

$$\mu(f \wedge g) \leq \mu(f) + \mu(g) \tag{4}$$

For example, circuit size is a formal complexity measure.

**Lemma 4.1.** *Suppose  $\mu$  is a formal complexity measure and there exists  $f \in F_n$  such that  $\mu(f) \geq c$  for some  $c$ . Then, for at least  $1/4$  of all functions  $g \in F_n$ ,  $\mu(g) \geq c/4$ .*

*Proof.* For some random  $g$ , say  $f = h \oplus g$ . Note that  $h$  is also random and  $h = f \oplus g$ . We have  $f = (\bar{h} \wedge g) \vee (h \wedge \bar{g})$ . Now if  $> 3/4$  of functions  $g'$  have  $\mu(g') < c$ , then with some probability, all of  $h, \bar{h}, g, \bar{g}$  have  $\mu < c$ , and hence  $\mu(f) < c$  and we have a contradiction.  $\square$

Razborov and Rudich also provide a more complicated proof that attains a better bound.