Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 167 (2011)

An improving on Gutfreund, Shaltiel, and Ta-Shma's paper "If NP Languages are Hard on the Worst-Case, Then it is Easy to Find Their Hard Instances"

Nikolay Vereshchagin * E-mail: ver@mccme.ru

Moscow State University, Leninskie gory 1, Moscow 119992, Russia

Abstract. Assume that NP $\not\subset$ BPP. Gutfreund, Shaltiel, and Ta-Shma in [Computational Complexity 16(4):412-441 (2007)] have proved that for every randomized polynomial time decision algorithm D for SAT there is a polynomial time samplable distribution such that D errs with probability at least $1/6-\varepsilon$ on a random formula chosen with respect to that distribution. In this paper, we show how to increase the error probability to $1/3-\varepsilon$.

* The work was in part supported by the RFBR grant 09-01-00709 and the ANR grant ProjetANR-08-EMER-008

ISSN 1433-8092

1 Introduction

A goal of the Average Case Complexity is to show that one way functions exist under a worst case hardness assumption like NP $\not\subset$ BPP.[1] Or, at least to show that SATISFIABILITY (SAT) is hard on average. The latter can be understood in two quite different ways, both of which use the notion of a sampler. Defining samplers, we will use the framework of Bogdanov and Trevisan [1] rather than the original Levin's one from [2].

Definition 1. *A sampler is a polynomial time probabilistic algorithm G that given 1^n as input outputs a Boolean formula. If the length of the output formula is always exactly n, we call the sampler* proper*. Sequences $\mu_o, \mu_1, \mu_2, \ldots$ of distributions for which there is a polynomial time sampler are called* polynomial time samplable ensembles of distributions*.*

In this paper, we consider Boolean formulas in the basis $\neg, \vee, \wedge, 0, 1$. The length $|\varphi|$ of a formula φ is defined as the number of symbols in it: every variable is counted as one symbol. Actually, in [1] samplers generate binary strings and not formulas, and samplers are always proper. To generate formulas, it is more natural to consider non-proper samplers, because this makes the results encoding-invariant.

The two ways to understand that SAT is hard on average are the following.

(a) There is a sampler G such that for every probabilistic polynomial time algorithm S, for infinitely many n with probability close to 1, the formula produced by $G(1^n)$ is satisfiable but S does not find its satisfying assignment. (The probability is with respect to the product of the uniform distribution over G's internal tosses and the uniform distribution over S's internal tosses.)

(b) There is a sampler G such that the following holds: For every probabilistic polynomial time decision algorithm D, for infinitely many n with probability close to $1/2$, D errs on the formula ψ produced by $G(1^n)$ (which means that D answers YES while ψ is not satisfiable or vice verse).

Note that for every decision problem the success probability $1/2$ can be obtained by a mere random guessing the result. Decision problems for which the success probability cannot deviate much from $1/2$ are called *hard core predicates*. They are used in cryptography and in construction of pseudo random generators.

The goal (a) is also related to constructing (strongly) one-way functions widely used criptography and derandomization. A polynomial time computable function f is *one-way* if for any probabilistic polynomial time algorithm that given $f(x)$ and $1^{|x|}$ tries to find a pre-image of $f(x)$ of length $|x|$ errs with probability close 1 for all sufficiently large $|x|$. If this happens only for infinitely many $|x|$ (for every probabilistic polynomial time inverting algorithm), then f is called "infinitely often" (i.o.) one-way function. Using NP completeness of SAT, one can show that if i.o. one-way functions exist then (a) is true. On the other

[1] NP $\not\subset$ BPP means that there is no polynomial time randomized algorithm that given any Boolean formula with probability at least $2/3$ correctly decides whether it is satisfiable.

hand, if there is a sampler as in (a), which generates a satisfiable formula ψ *together with its satisfying assignment* a, then the projection $(\psi, a) \mapsto \psi$ is an i.o. one-way function.

It is worth to mention that deciding satisfiability reduces to searching a satisfying assignment and the other way around (use a binary search). By the result of [3] the same holds in average case complexity, too, which is not trivial any more. More specifically, if for every polynomial time samplable distribution over formulas there is a polynomial time algorithm that with probability close to 1 correctly decides whether a given formula is satisfiable (closeness means that the difference is less than $1/p(n)$ for all polynomials p and large enough n), then for every polynomial time samplable distribution over formulas there is a polynomial time algorithm that with probability close to 1 correctly decides whether a given formula is satisfiable and finds a satisfying assignment if this is the case.

The paper [4] makes a step towards the goal (a). Namely, [4] shows that the assumption NP $\not\subset$ BPP implies the following weaker version of (a), in which we *allow the sampler* G *depend on the decision algorithm* D:

(a') For every polynomial time probabilistic algorithm S and ε there is a sampler G such that for infinitely many n, with probability at least $1 - \varepsilon$ the following holds: the formula φ produced by $G(1^n)$ is satisfiable and its length is at least n, however $S(\varphi)$ does not find its satisfying assignment.

In this result, it is important that the length of the formula produced by $G(1^n)$ tends to infinity, as n tends to infinity. Otherwise, the statement would become trivial, as G might produce any fixed formula on which S errs with high probability.

Moreover, in [5], under the same assumption, it is shown that for every such S there is a sampler G that for infinitely many n, with probability at least $1-\varepsilon$ produces a satisfiable formula φ and *its satisfying assignment* but $S(\varphi)$ does not find its satisfying assignment with probability at least $2/3$.

The paper [6] (whose conference version appeared two years before [4]) makes a similar progress regarding the goal (b). Namely, [6] shows that the assumption NP $\not\subset$ BPP implies the following weaker version of (2), in which we again allow the sampler G depend on the decision algorithm D:

(b') For every probabilistic polynomial time algorithm D there is a *proper* sampler G such that the following holds: for almost all n with probability at least 0.03, D errs on the formula ψ produced by $G(1^n)$ and its length is at least n. Actually, [6] requires the syntax of formulas allow padding: given a formula φ one can find in polynomial time a formula of length $|\varphi| + 1$, which is equivalent to φ.

The authors of [6] remark that they do not optimize constants and by careful calculation the constant 0.03 can be improved; however they do not see how to get it above the barrier of 1/3. Indeed, a careful calculation shows that 0.03 may be replaced by $1/24 - \varepsilon$ (for any positive ε). But we do not see how, using the techniques of [6], to get the constant above 1/24.

In this paper:

– We notice that, for non-proper samplers, the constant 0.03 in (b') can be improved to $1/6 - \varepsilon$.
– We show that, using an extra trick, one can get even $1/3-\varepsilon$ (for non-proper samplers).

In other words, we show how to double the error probability in (b'). However, our result still does not break the 1/3 barrier and the question whether one can replace 1/3 by 1/2 remains open. Note that the barrier of 1/2 can be broken for Σ_k^p predicates for every $k > 1$. A result of [4] states that if Σ_k^p is not included in BPP then for every probabilistic polynomial time algorithm A there is a sampler G such that for infinitely many n, algorithm A errs on $G(1^n)$ with probability close to 1/2. As we said, for $k = 1$ (that is for NP), this is still open.

2 Generating hard instances of search version of SAT

We start with presenting the main construction of [6] so that it be clear what our contribution is.

Definition 2. *The* search version of SAT *is the following problem: given a Boolean formula φ find its satisfying assignment. A (randomized)* SAT solver *is a (randomized) polynomial time algorithm that for every input formula φ either finds its satisfying assignment, or says "don't know". A SAT solver D* errs *on ψ if ψ is satisfiable and $D(\psi) =$"don't know".*

Theorem 1 ([6]). *Assume that NP $\neq$ P. Given a deterministic SAT solver S one can construct a deterministic polynomial time procedure that given 1^n produces a formula ψ_n of length at least n such that S errs on ψ_n for infinitely many n.*

Proof. Consider the following search problem in NP.
Search problem P:
Instance: a string 1^n over the unary alphabet.
Solution: a pair (ψ, a) where ψ is a satisfiable formula of length n such that $S(\psi) =$"don't know", and a is its satisfying assignment.

We will call an instance 1^n of P *solvable* if such pair (ψ, a) exists. As SAT is NP complete, the search problem P reduces to the search version of SAT. This means that there is a polynomial time algorithm that given 1^n finds a formula, called φ_n, such that:
(1) if the instance 1^n of P is solvable then φ_n is satisfiable, and
(2) given any satisfying assignment of φ_n we can find (in polynomial time) a solution to the instance 1^n of problem P.
The length of φ_n is bounded by a polynomial n^d and w.l.o.g. we may assume that $|\varphi_n| \geq n$.

The procedure works as follows: given 1^n, as input
(a) find the formula φ_n;
(b) run $S(\varphi_n)$;

(c) if $S(\varphi_n)$="don't know"then output φ_n and halt;
(d) otherwise $S(\varphi_n)$ produces a satisfying assignment for φ_n; given that assignment find in polynomial time a solution (ψ, a) to the instance 1^n of the problem P; output ψ and halt.

Since we assume that P $\neq$ NP, for infinitely many n the instance 1^n of P is solvable. For such n either $S(\varphi_n)$="don't know"(and thus S errs on φ_n), or (ψ, a) is a solution to 1^n (and thus S errs on ψ).

The next construction of [6] allows to generalize this theorem to randomized SAT solvers. This is done as follows. Let S be a randomized SAT solver working in time n^c and let r be string of length at least n^c. We will denote by S_r the algorithm S that uses bits of r as coin flips. Note that S_r a deterministic algorithm.

Theorem 2 ([6]). *Assume that NP $\not\subset$ BPP. Then for some natural constant d the following holds. Let S be a randomized SAT solver and let n^c denote its running time on formulas of length n. Then there is a deterministic polynomial time procedure that given any binary string r of length $n^{c^2 d}$ produces a formula η_r of length between n and n^{cd}, where for any positive ε for infinitely many n the following holds. For a fraction at least $1-\varepsilon$ of r's the algorithm S_r errs on η_r.*

Notice that the length of η_r is at most n^{cd}. Therefore the running time of S for input η_r is at most $n^{c^2 d}$. Hence $S_r(\eta_r)$ is well defined.

Proof. The proof is very similar to that of the previous theorem. The only change is that we have to replace the search problem P by the following problem P':
Instance: a binary string r' of length n^c (for some n).
Solution: a satisfiable formula ψ of length n and its satisfying assignment a such that $S_{r'}(\psi)$ ="don't know".

Let $r' \mapsto \varphi_{r'}$ be a reduction of P' to the search version of SAT. The length of $\varphi_{r'}$ is bounded by a polynomial n^{cd} of $|r'| = n^c$ and w.l.o.g. we may assume that $|\varphi_{r'}| \geq n$.

The procedure required in the theorem, called **Procedure A**, works as follows: given r of length $n^{c^2 d}$, as input,
(a) let r' stand for the prefix of r of length n^c;
(b) find the formula $\varphi_{r'}$; recall that satisfying assignments of $\varphi_{r'}$ are basically pairs (a formula ψ of length n, its satisfying assignment a) such that $S_{r'}(\psi)$ ="don't know";
(c) run $S_r(\varphi_{r'})$;
(d) if $S_r(\varphi_{r'})$="don't know"then output $\varphi_{r'}$ and halt;
(e) otherwise $S_r(\varphi_{r'})$ produces a satisfying assignment for $\varphi_{r'}$; given that assignment find in polynomial time a solution (ψ, a) to the instance r' of the problem P'; output ψ and halt. (End of Procedure A.)

Let η_r stand for the formula output by the procedure. Since we assume that NP $\not\subset$ BPP, for every positive ε the randomized searching algorithm S errs with probability at least $1-\varepsilon$ for infinitely many input formulas. This implies that for infinitely many n the number of solvable instances r' of the problem P' is at least $(1-\varepsilon)2^{n^c}$. For those r's the formula $\varphi_{r'}$ is satisfiable. Therefore, for all but a fraction ε of r's the algorithm S_r errs on $\varphi_{r'}$ or $S_{r'}$ errs on ψ, which implies that S_r errs on ψ as well.

Remark 1. Theorem 2 holds for $\varepsilon = 1/n^k$ for any constant k. Indeed, the assumption NP $\not\subseteq$ BPP implies that the randomized searching algorithm S errs with probability at least $1 - |\varphi|^{-k}$ for infinitely many input formulas φ.

3 Generating hard instances of the decision version of SAT

We say that a randomized decision algorithm D with randomness r *errs on a formula* φ if $D_r(\varphi)$ =YES and φ is not satisfiable or vice verse.

Here is our main result.

Theorem 3. *If NP $\not\subseteq$ BPP then for every probabilistic polynomial time decision algorithm D and every positive ε there is a sampler G such that for infinitely many n with probability at least $1-\varepsilon$ the decision algorithm D errs on the formula produced by $G(1^n)$ with probability at least $1/3-\varepsilon$ and the length of the formula is at least n.*

Remark 2. This result strengthens a result that is implicit in [6], which states the same with 1/6 in place of 1/3. In the proof we will explain what is the difference between the construction in [6] and ours. For proper samplers the constant 1/6 should be reduced to 1/24 by the following reason. Using a padding we may assume that the formula output by the sampler has length either n, or n^{cd} (and not in between). Consider a new sampler $\tilde{G}$ that runs $G(1^n)$ and $G(1^{n^{1/cd}})$ and if either of the runs produces a formula of length n, then we output that formula (if both runs produce a formula of length n then we output each of them with probability 1/2). This yields the constant $1/24-\varepsilon$. Indeed, assume that $G(1^m)$ produces a formula φ such that $D(\varphi)$ errs with probability $1/6-\varepsilon$. Then either the event "$D(\varphi)$ errs and the length of φ is m" or the event "$D(\varphi)$ errs and the length of φ is m^{cd}" has probability at least $1/12-\varepsilon/2$. In the first case the probability of the event "D errs on the output of $G(1^m)$" is at least $1/24-\varepsilon/4$. In the second case the probability of the event "D errs on the output of $G(1^{m^{cd}})$" is at least $1/24-\varepsilon/4$.

Proof (of Theorem 3). Let D and ε be given. First we use the standard amplification, as in [7], to transform the algorithm D into another decision algorithm $\bar{D}$ with a smaller error probability.

Given a formula φ of length n as input the algorithm $\bar{D}$ invokes $D(\varphi)$ polynomial number K of times and outputs the most frequent result among all the results obtained in those runs. If K is large enough (but still polynomial in n) then the probability that the frequency of the result YES in those K runs differs from the probability that $D(\varphi)$ = YES by more than ε is exponentially small in

n. This follows from the Chernoff bound. Note that the number of formulas of length n is also exponential in n. Moreover, we can choose $K = \mathrm{poly}(n)$ so that with probability at least $1 - 2^{-n}$ there is no formula φ of length n for which the frequency of the result YES deviates from the probability that $D(\varphi) = \mathrm{YES}$ by at most ε.

Using the standard binary search techniques we transform the algorithm $\bar{D}$ to a SAT solver S. That is, given a formula φ the algorithm S first runs $\bar{D}(\varphi)$. If the result is YES then it substitutes first $x = 0$ and then $x = 1$ for the first variable x in φ and runs $\bar{D}$ on the resulting formulas $\varphi_{x=0}$, $\varphi_{x=1}$. If at least one of these runs outputs YES, we replace φ by the corresponding formula and recurse. Otherwise we return "don't know"and halt.

If $\bar{D}$ returns NO for the input formula φ, we return "don't know"and halt. Finally, if we have substituted 0s and 1s for all variables and the resulting formula is true, we return the satisfying assignment we have found, and otherwise we return "don't know".

Let n^c be the upper bound of S's running time for input formulas of length n and let r be a string of length n^c used as randomness for S. In its run for input φ the algorithm S_r uses parts of r as coin flips for $\bar{D}$. With some abuse of notation we will denote by $\bar{D}_r$ the algorithm $\bar{D}$ with that randomness. In the same way the notation D_r is understood.

The heart of the construction is a procedure that given any formula ψ and randomness r such that S_r errs on ψ returns at most three formulas such that the algorithm D errs on at least one of those formulas with high probability.

Procedure B. Given a satisfiable input formula ψ and r such that $S_r(\psi) =$"don't know", run $S_r(\psi)$ to find the place in the binary search tree where S_r is stuck. By the construction of S this may happen in the following three cases:

(1) $\bar{D}_r(\psi)$=NO. In this case output ψ.

(2) $S_r(\psi)$ performs the binary search till the very end, it finds a formula η obtained from original formula by substituting all its variables by 0,1 such that η is false while $\bar{D}_r$ claims that η is true. In this case output η.

(3) In the remaining case $S_r(\psi)$ is stuck in the middle of the binary search and thus has found a formula φ and its variable x such that $\bar{D}_r(\varphi)$ =YES while both $\bar{D}_r(\varphi_{x=0})$ and $\bar{D}_r(\varphi_{x=1})$ are NO. In this case return $\varphi, \varphi_{x=0}, \varphi_{x=1}$. (End of Procedure B.)

We will call formulas returned by this procedure by α, β, γ.[2] They depend on input formula ψ and on randomness r.

By Theorem 2 applied to the search algorithm S there is a polynomial procedure (called Procedure A in the proof) with the following property. Given a string r of length $n^{c^2 d}$ the procedure returns a formula η_r of length between n and n^{cd} such that for infinitely many n, S_r errs on η_r (except for a fraction at most ε of r's).

The sampler G from [6] works as follows. For input 1^n choose a random string r of length $n^{c^2 d}$. Then apply Procedure A to r to obtain η_r. Then apply

[2] Without loss of generality we may assume that Procedure B always outputs three formulas.

Procedure B to S, r and η_r to obtain three formulas α, β, γ. Finally choose one of these formulas at random, each with probability 1/3, and output it.

Fix a positive ε. We claim that for infinitely many n with probability at least $1-2\varepsilon$ the algorithm D errs on the formula produced by $G(1^n)$ with probability at least $1/6-\varepsilon$. To prove this claim notice that $S_r(\eta_r)$ calls $\bar{D}_r$ at most $2n^{cd}$ times (two times for each variable). Each time $\bar{D}_r$ is called on an input formula φ of length between n and n^{cd}. Call a string r of length n^{c^2d} *bad* if in at least one of these runs of $\bar{D}_r$ the frequency of YES answers of D for input φ differs from the probability of the event $D(\varphi)$=YES by more than ε (recall that $\bar{D}_r(\varphi)$ runs $D(\varphi)$ some K times). By construction of $\bar{D}$ a fraction at most

$$\sum_{l=n}^{n^{cd}} 2n^{cd}2^{-l} < n^{cd}2^{-n+2} \leq \varepsilon$$

r's are bad (for all large enough n). If r is good and S_r errs on η_r then the error probability of D on the formula output by Procedure B is at least $1/3(1/2-\varepsilon)$. And for infinitely many n the probability that S_r does not err on η_r is at most ε. Thus for infinitely many n with probability at least $1-2\varepsilon$ both S_r errs on η_r and r is good hence D errs on the output formula with probability at least $1/3(1/2-\varepsilon)$.

Up to now we have just recited the arguments from [6]. Now we will present a new trick, which improves the constant 1/6 to 1/3. We will change in the work of this sampler the very last step. This time we will output α, β, γ with different probabilities. Recall that Procedure B has run the algorithm $\bar{D}_r$ on inputs α, β, γ, and algorithm $\bar{D}_r$ has done majority vote among some number K of runs of the algorithm D_r on α, β, γ, respectively. The algorithm $\bar{D}_r$ has output the most frequent answer produced in those runs and we know that at least one of results $\bar{D}_r(\alpha), \bar{D}_r(\beta), \bar{D}_r(\gamma)$ is incorrect for all r's except for a fraction at most ε. Let u, v, w stand for the frequencies of the most frequent results of D_r in the runs of $\bar{D}_r$ on α, β, γ, respectively. Obviously, all the numbers u, v, w are at least 1/2. If $u < 2/3$, we just output α (with probability 1) and halt, as in this case the probabilities of both events $D(\alpha)$ =NO, $D(\alpha)$ =YES are greater than $1/3-\varepsilon$ (assuming that r is good). We proceed similarly if v or w is less than 2/3.

Otherwise find non-negative rational numbers p, q, s such that $p+q+s=1$ and all the numbers

$$pu+q(1-v)+s(1-w), \quad p(1-u)+qv+s(1-w), \quad p(1-u)+q(1-v)+sw \quad (1)$$

are at least 1/3 (we will argue later that such p, q, s exist). Then output α, β, γ with probabilities p, q, s, respectively, and halt.

Assume that at least one of the results

$$\bar{D}_r(\alpha), \quad \bar{D}_r(\beta), \quad \bar{D}_r(\gamma)$$

is wrong and r is good. If the answer $\bar{D}_r(\alpha)$ is wrong, then the probability that D errs on α is at least $u-\varepsilon$. The probability that D errs on β is at least $1-v-\varepsilon$.

The same holds for γ. Thus the overall probability that D errs on the resulting formula is at least

$$p(u-\varepsilon)+q(1-v-\varepsilon)+s(1-w-\varepsilon)\geq 1/3-\varepsilon.$$

In the case when one of the results $\bar{D}_r(\beta), \bar{D}_r(\gamma)$ is incorrect, we need that the second and the third numbers in (1) be at least 1/3.

Take into account a fraction at most ε of bad r's and also a fraction at most ε of r's such that S_r does not err on η_r. We obtain that with probability at least $1-2\varepsilon$ the algorithm D errs on the formula produced by $G(1^n)$ with probability at least $1/3-\varepsilon$.

It remains to show that there are non-negative p, q, s such that $p+q+s=1$ and all the numbers in (1) are at least 1/3. Note that the arithmetic mean of those numbers is equal

$$\frac{1+p(1-u)+q(1-v)+s(1-w)}{3}\geq\frac{1}{3}.$$

Thus it suffices to show that there are non-negative p, q, s such that all the three numbers in (1) are equal (and thus the maximum equals to the arithmetical mean):

$$pu+q(1-v)+s(1-w)=p(1-u)+qv+s(1-w)=p(1-u)+q(1-v)+sw.$$

The first inequality means that $p(2u-1)=q(2v-1)$ and the second one means that $q(2v-1)=r(2w-1)$. Thus all the three numbers are equal, if p, q, s are proportional to $1/(2u-1), 1/(2v-1), 1/(2w-1)$. As all u, v, w are bounded away from 1/2 (we are assuming that these numbers are at least 2/3), all these numbers are bounded by a constant. Thus we are able to find in polynomial time the desired p, q, s.

Remark 3. Theorem 3 remains true for $\varepsilon=1/n^k$ for any constant k.

Remark 4. Say that NP $\not\subset$ BPP *everywhere* if there is a constant c such that for every randomized SAT solver S and all $n>1$, S errs on a formula of length between n and n^c. (A randomized SAT solver S errs on a formula φ if $S(\varphi)=$"don't know"with probability more 1/2.)

If instead of NP $\not\subset$ BPP we assume that NP $\not\subset$ BPP everywhere then all our result holds in a stronger form: the quantifier "for infinitely many n" may be replaced by the universal quantifier.

Theorem 4. *If NP $\not\subset$ BPP everywhere then for every probabilistic polynomial time decision algorithm D and every positive ε there is a sampler G such that* for all n *the decision algorithm D errs on $G(1^n)$ with probability at least* $1/3-\varepsilon$ *and the length of the formula is at least n.*

The proofs of this theorem is entirely similar to that of Theorem 3 and thus we omit it. We only have to replace, in the definition of the search problem P, the requirement "the length of ψ is n" by the requirement "the length of ψ is between n and n^c" (and make a similar change in the definition of problem P'). The constructed sampler will work for almost all n, which is enough, as we can change its behavior for the remaining n.

References

1. Andrej Bogdanov and Luca Trevisan, Average-Case Complexity, *Foundations and Trends in Theoretical Computer Science* 1(2) (2006) 1–106.
2. Leonid A. Levin, Average Case Complete Problems. *SIAM J. Comput.* 15:1 (1986) 285–286.
3. Shai Ben-David, Benny Chor, Oded Goldreich. Michael Luby, On the Theory of Average Case Complexity. *STOC 1989* 204–216
4. D. Gutfreund, Worst-Case Vs. Algorithmic Average-Case Complexity in the Polynomial-Time Hierarchy, *Proceedings of the 10th International Workshop on Randomization and Computation, RANDOM 2006*, Lecture Notes in Computer Science Volume 4110, 2006, pp 386-397.
5. Andrej Bogdanov , Kunal Talwar , Andrew Wan. Hard instances for satisfiability and quasi-one-way functions. Proceedings of Innovations in Computer Science (ICS 2009). Tsinghua University Press 2009, pp. 290-300.
6. D. Gutfreund, R. Shaltiel, A. Ta-Shma, If NP Languages are Hard on the Worst-Case, Then it is Easy to Find Their Hard Instances. *Computational Complexity (CC),* 16:4 (2007) 412-441.
7. L. M. Adleman. Two Theorems on Random Polynomial Time. FOCS 1978: 75-83

ECCC ISSN 1433-8092

http://eccc.hpi-web.de