On Pseudodeterministic Approximation Algorithms

Peter Dixon[1]
Iowa State University, Ames, USA
tooplark@iastate.edu

A. Pavan[2]
Iowa State University, Ames, USA
pavan@cs.iastate.edu

N. V. Vinodchandran[3]
University of Nebraska, Lincoln, USA
vinod@cse.unl.edu

Abstract

We investigate the notion of *pseudodeterminstic approximation algorithms*. A randomized approximation algorithm A for a function f is *pseudodeterministic* if for every input x there is a *unique value* v so that $A(x)$ outputs v with high probability, and v is a good approximation of $f(x)$. We show that designing a pseudodeterministic version of Stockmeyer's well known approximation algorithm for the NP-membership counting problem will yield a new circuit lower bound: if such an approximation algorithm exists, then for every k, there is a language in the complexity class $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ that does not have n^k-size circuits. While we do not know how to design such an algorithm for the NP-membership counting problem, we show a general result that any randomized approximation algorithm for a counting problem can be transformed to an approximation algorithm that has a *constant number* of *influential* random bits. That is, for most settings of these influential bits, the approximation algorithm will be pseudodeterministic.

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Theory of computation → Circuit complexity

Keywords and phrases Approximation Algorithms, Circuit lower bounds, Pseudodeterminism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.61

1 Introduction

Consider the computational problem: Given an input 1^n, generate an n-bit prime. Is there a *deterministic polynomial-time* algorithm for this problem? Even though primality testing can be done in deterministic polynomial time, we do not know whether a deterministic polynomial-time algorithm exists for this problem. However, a straightforward *probabilistic algorithm* exists for this task: Randomly pick an n-bit integer and output it if it is a prime. The density of primes implies that this algorithm succeeds with inverse polynomial probability. Hence by repeating this procedure polynomially many times, we can output an n-bit prime number with very high probability. A drawback of this algorithm is that the output of the algorithm depends on the random choices it makes. In particular, on two different random choices the algorithm may output two different prime numbers. A natural question to ask is:

[1] Research Supported in part by NSF grant 1421163.
[2] Research Supported in part by NSF grant 1421163.
[3] Research Supported in part by NSF grant 1422668.

is there a randomized, polynomial-time algorithm that on input 1^n outputs a *unique n-bit prime number* on most random choices. Such an algorithm is called a *pseudodeterministic algorithm*. This notion of pseudodeterminism was formulated by Gat and Goldwasser [7], motivated by its applications in distributed computing and cryptography.

Let f be a multi-valued function (i.e. $f(x)$ is a non-empty set). We say that a probabilistic algorithm A computes f if $A(x)$ outputs a value in the set $f(x)$ with high probability. A probabilistic algorithm A for f is *pseudodeterministic* if for every x, there exists *a unique* $u \in f(x)$ such that $A(x)$ outputs u with high probability [7]. The problem of designing pseudodeterministic algorithms is interesting in scenarios where we know of a probabilistic polynomial time algorithm that computes f, but no deterministic algorithm for f is known.

Related Work

Since the work of Gat and Goldwasser, the notion of pseudodeterministic algorithms has received moderate attention [7, 8, 9, 10, 11, 16]. Now we know of pseudodeterministic algorithms for certain algebraic problems such as finding non-residues of $\mathbb{Z}_p$, and finding non-roots of multivariate polynomials [7]. The work of Oliveria and Santhanam gives a pseudodeterministic, sub-exponential time algorithm for generating primes (that works at infinitely many input lengths) [16]. Recently Goldwasser and Grossman obtained a pseudodeterministic NC algorithm that computes perfect matchings in bipartite graphs [9]. The work of Goldreich, Goldwasser and Ron investigates the power and limitations of pseudodeterministic algorithms in the context of general probabilistic algorithms and sub-linear time algorithms [8]. The notion of *pseudodeterminism* has been studied earlier in the literature in the context of approximation algorithms [6], where the authors relate the existence of such algorithms for approximation problems to communication complexity of certain key-agreement problems. They used the term *monic selection* to capture the notion of *pseudodeterminism*. To the best of our knowledge, pseudodeterminism in the context of computing approximations of function has not been studied further.

Our Contribution

In this paper, we investigate pseudodeterministic approximation algorithms. Given a function f whose range is the integers, we say that a probabilistic algorithm A is a *pseudodeterministic approximation algorithm* for f if, for every input x, there is a value v such that $A(x)$ outputs v with high probability and v is a "good approximation" of $f(x)$, (for a formal definition, please see the next section). We consider the following counting problem which we call NP-*membership counting problem*: Given a language L in NP, compute the number of strings in L at a given length. The well-known result due to Stockmeyer [19, 20, 3] shows that NP-membership counting problem can be approximated in $\mathrm{BPP}_{tt}^{\mathrm{NP}}$, where $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ denotes the class of problems solvable in probabilistic polynomial-time with nonadaptive queries to SAT[4]. However, Stockmeyer's algorithm is not pseudodeterministic: the algorithm can output different good approximations on different probabilistic paths. A natural question is: Is there a pseudodeterministic $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ approximation algorithm for NP-membership counting problem? We relate this question to establishing new circuit lower bounds.

Proving circuit lowerbounds is one of the most significant research directions in complexity theory. While it is widely believed that there are languages in NP that cannot be solved by subexponential-size Boolean circuits, we do not even know how to prove that NP does

[4] Even though $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ denotes a class of languages, here we slightly abuse the notation and view it as a function class.

not have linear-size circuits. This leads to investigating the question: "What is the smallest complexity class that provably cannot be solved by a linear-size circuit?" The first result in this direction is due to Kannan [13], who showed that there are languages in the second level of the Polynomial-time Hierarchy (Σ_2^{P}), that do not have linear-size circuits. This result has been improved to show that there are languages in $\mathrm{ZPP}^{\mathrm{NP}}$ that do not have linear-size circuits [4, 15]. Subsequently, Sengupta [5] showed that the complexity class $\mathrm{S}_2^{\mathrm{P}}$ does not have linear-size circuits. This together with Cai's result [5] that $\mathrm{S}_2^{\mathrm{P}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$, improved the upper bound to $\mathrm{S}_2^{\mathrm{P}}$. It is also known that the complexity class PP has languages that do not have linear-size circuits [21]. Currently a significant open question is to show that the class MA does not have linear-size circuits. A step in this direction is a result due to Santhanam [18] who showed that MA//1 does not have linear-size circuits (MA//1 is a "promise version" of MA: see the next section for a formal definition). We note that in all these circuit lower bound results, "linear-size" can be replaced by "n^k-size circuits" (for a fixed $k \geq 0$).

We ask the following question: Can we show that $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ has languages that do not have linear-size circuits? Note that $\mathrm{MA} \subseteq \mathrm{ZPP}_{tt}^{\mathrm{NP}} \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ and hence it is a natural question. In this work, we show that this goal can be achieved if we can design a psedodeterminstic version of Stockmeyer's approximate counting algorithm for the NP-membership counting problem.

Theorem. *If there is a pseudodeterministic* $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ *approximation algorithm for the* NP*-membership counting problem, then for every* k *there is a language* L_k *in* $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ *that does not have* n^k*-size circuits.*

We note that there are oracles with respect to which $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ has linear-size circuits [1]. Hence a pseudodeterministic algorithm for the NP-membership counting problem will lead to non-relativizable circuit lower bounds.

Can we design a pseudodeterministic $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ approximation algorithm for the NP-membership counting problem? While we are unable to answer this question, we show a very general result: every randomized approximate counting algorithm can be transformed to an approximation algorithm that has a constant number of *influential random bits* in the sense defined recently by Grossman and Liu [12]. Grossman and Liu [12] generalized the notion of pseudodeterministic algorithms to *influential bit algorithms*. They defined this new notion in the context of logspace computation and applied it to certain search problems in randomized logspace. This notion can be adapted to polynomial-time bounded settings. Informally, a probabilistic algorithm A is a $k(n)$*-bit influential algorithm*, if A (on an input x of length n), uses $k(n) + r(n)$ random bits, and for most choices of the first $k(n)$-random bits, A behaves in a pseudodeterministic manner. That is, for most $p \in \Sigma^{k(n)}$, there exists a v such that $A(x, pr)$ outputs v with high probability (where r is randomly chosen from $\Sigma^{r(n)}$). For two different strings p_1 and p_2, the outputs of $A(x, p_1 r)$ and $A(x, p_2 r)$ could differ. However, if we fix a "good" $k(n)$-bit string p, then $A(x, pr_1)$ is the same as $A(x, pr_2)$ for most choices of r_1 and r_2. Note that an algorithm is pseudodeterministic if and only if it is a 0-influential bit algorithm. We show that any randomized relative-error approximate counting algorithm can be made to have a constant number of influential bits. This implies that

Theorem. *There is a* $O(1)$*-bit influential,* $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ *approximation algorithm for the* NP*-membership counting problem.*

2 Preliminaries

We assume familiarity with standard notation and definitions from complexity theory [2].

▶ **Definition 1.** Let f be a function whose range is the integers. We say that a probabilistic algorithm A is an (ϵ, δ)-approximation algorithm for f if for every x, the random variable $A(x)$ has the following property: $\Pr[(1-\epsilon)f(x) \leq A(x) \leq (1+\epsilon)f(x)] \geq (1-\delta)$. We say that A is an (ϵ, δ) *pseudodeterministic approximation algorithm* for f if for every x there exists an integer v such that

$$(1-\epsilon)f(x) \leq v \leq (1+\epsilon)f(x) \text{ and } \Pr[A(x) = v] \geq 1-\delta.$$

In general an approximation algorithm can output different good approximations on different random choices. For an approximation algorithm A to be pseudodeterministic, A has to usually output a *unique* approximation v which is independent of the random string, for every input.

▶ **Definition 2.** Let f be a function whose range is the integers. We say that f has an (ϵ, δ)-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ *pseudodeterministic approximation algorithm* if there exists an (ϵ, δ), polynomial-time, pseudodeterministic approximation algorithm for f, that makes nonadaptive queries to SAT.

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary-relation that is decidable in NP. Let $f_R(x)$ be the number of strings y such that $\langle x, y\rangle \in R$. The class $\sharp\mathrm{NP}$ is defined as $\sharp\mathrm{NP} = \{f_R \mid R \text{ is a relation that is decidable in NP}\}$. The following well-known theorem due to Stockmeyer gives a $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ approximation algorithm for problems in $\sharp\mathrm{NP}$.

▶ **Theorem 3** ([19]). *Every problem in* $\sharp\mathrm{NP}$ *has a* $(1/n, 1/2^n)$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ *approximation algorithm.*

Stockmeyer's algorithm is not pseudodeterministic: it may produce different, correct approximations on different random choices.

Recently, Grossman and Liu defined the notion of *influential bits* as a generalization of pseudodeterminism [12]. They defined it in the logspace regime and applied it to search problems in RL. We adapt it to the context of approximation algorithms.

▶ **Definition 4.** Let f be a function whose range is the integers. For a function $k : \mathbb{N} \to \mathbb{N}$, we say that a randomized (ϵ, δ)-approximation algorithm A for f is $k(n)$*-bit influential* if for every x, the following holds: A takes random string $r = st$ where $|s| \leq k(n)$ and for more than $\frac{2}{3}$ of strings s, there exists an integer v_s such that $\Pr_t[A(x, st) = v_s] \geq 1-\delta$ and $(1-\epsilon)f(x) \leq v_s \leq (1+\epsilon)f(x)$.

Note that an algorithm is pseudodeterministic if and only if it is 0-bit influential. Next we define a variant of the complexity class AM.

▶ **Definition 5.** A language L is in $\mathrm{AM}//1$ if there exists a polynomial-time machine V, a polynomial p, and a sequence of bits $b_1, b_2, \cdots$ such that

$$x \in L \Rightarrow \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \geq 2/3,$$

$$x \notin L \Rightarrow \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \leq 1/3,$$

where n is the length of x.

Santhanam showed that for every k there is a language in AM//1 that does not have n^k-size circuits [18] (Santhanam showed the lower bound for MA//1, but we will only need the AM//1 bound for establishing our result).

▶ **Theorem 6** ([18]). *For every k, there is a language $L_k \in$ AM//1 so that L_k cannot be computed by n^k-size Boolean circuits.*

3 Pseudodeterminism for ♯NP and circuit lower bounds

In this section we show that the existence of a pseudodeterministic version of Stockmeyer's approximate counting algorithm for ♯NP implies new circuit lower bounds.

▶ **Theorem 7.** *If for every function f in ♯NP there exists an $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ pseudodeterministic approximation algorithm for f, then for every $k > 0$, there is a language L_k in $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ such that L_k cannot be computed by n^k-size Boolean circuits.*

Proof of this theorem follows from Lemma 8, and Theorem 9 stated below. Theorem 9 could be of independent interest.

▶ **Lemma 8.** *If for every function f in ♯NP there exists an $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ pseudodeterministic approximation algorithm for f, then for every $k > 0$, there is a language L_k in $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ such that L_k cannot be computed by n^k-size Boolean circuits.*

▶ **Theorem 9.** *If* NP *has polynomial-size circuits, then $\mathrm{BPP}^{\mathrm{NP}}_{tt} = \mathrm{ZPP}^{\mathrm{NP}}_{tt}$.*

Assuming Lemma 8 and Theorem 9, Theorem 7 follows from the type of argument due to Kannan [13]. If NP does not have polynomial-size circuits, trivially $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ does not have n^k-size circuits. Assume that NP has polynomial-size circuits. By Lemma 8 we have $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ does not have n^k-size circuits, and by Theorem 9 $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ is the same as $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$. Hence $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$ does not have n^k-size circuits.

Proof of Lemma 8. Fix a $k > 0$ and let L be a language in AM//1 that does not have n^k-size circuits [18]. Thus there exists a sequence of bits $b_1, b_2, \cdots$, a polynomial p and a polynomial-time machine V such that

$$x \in L \quad \Rightarrow \quad \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \geq 2/3, \text{ and}$$
$$x \notin L \quad \Rightarrow \quad \Pr_{r \in \Sigma^{p(n)}}[\exists y \in \Sigma^{p(n)},\ V(xb_n, r, y) = 1] \leq 1/3$$

Consider the following NP relation:

$$R = \{\langle xb, r\rangle \mid r \in \Sigma^{p(|x|)}, b \in \{0,1\}, \exists y \in \Sigma^{p(n)} V(xb, r, y) = 1\}$$

The corresponding ♯NP problem is $f_R(xb) = |\{r \mid \langle xb, r\rangle \in R\}|$. By our hypothesis, there is a $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}^{\mathrm{NP}}_{tt}$ pseudodeterministic approximation algorithm M for f_R. Let S_{xb} denote the set $\{r \mid \langle xb, r\rangle \in R\}$. Consider the following $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ algorithm.

Input: x and a bit b.
Run M on xb. Let v be the output.
If $v \geq 2^{p(n)-1}$ then accept, else reject.

We will first show that on every input $\langle x, b\rangle$, the above algorithm either accepts with high probability or rejects with high probability. Let $\langle x, b\rangle$ be an input. Since M is a pseudodeterministic algorithm, there is an integer v such that M outputs v with probability at least $\frac{9}{10}$. If v is at least $2^{p(n)-1}$, then the above algorithm accepts, else it rejects. Thus on every input $\langle x, b\rangle$, the above algorithm either accepts with probability $\frac{9}{10}$ or rejects with probability at least $\frac{9}{10}$. Let L' be the language accepted by the above algorithm. Clearly $L' \in \mathrm{BPP}^{\mathrm{NP}}_{tt}$.

▶ **Claim 9.1.** *L' cannot be computed by n^k-size Boolean circuits.*

Assume that L' is accepted by a circuit family of size n^k. We will show that there is an n^k-size circuit family for L, which gives us a contradiction. Fix an input length n and let C_n be a circuit that accepts L' at length $n+1$. The definition of advice guarantees that for each input length, there is some advice string (in our case, a single bit) for which the circuit correctly decides the language. Let b_n be the *correct advice* bit for strings of length n for the language L. Consider the following circuit D_n: $D_n(x)$ is same as $C_n(x, b_n)$. I.e., with b_n hardwired in C_n. Thus D_n on input x (of length n) evaluates C_n on input $\langle x, b_n\rangle$. Clearly the size of D_n is n^k. We now show that the circuit family D_n accepts the language L.

Consider the case that x is in L. Note that $f_R(xb_n)$ is at least $\frac{2}{3}2^{p(n)}$. Thus on input $\langle x, b_n\rangle$ M outputs a number v with probability at least $2/3$ such that $v \geq \frac{9}{10} f_R(xb_n)$. Since $f_R(xb_n)$ is at least $\frac{2}{3}2^{p(n)}$, we have that v is bigger than $2^{p(n)-1}$. Thus M accepts $\langle x, b_n\rangle$ with probability at least $\frac{9}{10}$. Thus $\langle x, b_n\rangle \in L'$, and the circuit C_n accepts $\langle x, b_n\rangle$. Thus D_n accepts x. A similar argument shows that if x is not in L, then D_n rejects x. Since the size of D_n is n^k, we have that L is accepted by n^k-size circuits, which is a contradiction. ◀

Next, we complete the proof of Theorem 9.

Proof of Theorem 9. The argument goes in two steps: (1) under the assumption that NP has polynomial-size circuits, it can be shown that a Boolean function has $S(n)$-size SAT oracle circuits if and only if it has $S'(n)$-size Boolean circuits, where $S'(n)$ and $S(n)$ are polynomially related. (2) A ZPP machine can randomly pick a truth-table t of a Boolean function on $O(\log n)$-bits and verify using the NP oracle that t does not have $2^{\epsilon n}$-size circuits. Use this hard function to construct a pseudorandom generator. Use this PRG to derandomize a $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ computation to $\mathrm{ZPP}^{\mathrm{NP}}_{tt}$. We provide more details.

Since NP has polynomial-size circuits, there is a constant $\ell > 0$ such that SAT has n^ℓ-size circuits. Suppose that a Boolean function g has $2^{\gamma n}$-size SAT-oracle circuit family $\{D_n\}$. Consider D_n. We can convert this into a circuit that does not make oracle calls to SAT. This circuit can generate queries (to SAT) of size at most $2^{\gamma n}$. Since SAT has n^ℓ-size circuits, each query can be answered by a circuit of size $2^{\ell\gamma n}$. Thus if we replace each oracle call in D_n by a circuit of size at most $2^{\ell\gamma n}$, we obtain Boolean circuit of size $2^{\delta n}$ for g where $\delta = 2l\gamma$. This circuit does not make any queries to SAT.

We will use the following derandomization result due to Klivans and Melkebeek [14]

▶ **Theorem 10.** *Let M be a $\mathrm{BPP}^{\mathrm{NP}}_{tt}$ machine that uses $t(n)$ random bits on inputs of length n. For every $\gamma > 0$, there exists b, c and a polynomial-time computable family of functions $\{F_n\}$ such that $F_n : \Sigma^{n^c} \times \Sigma^{b \log n} \to \Sigma^{t(n)}$, and if u is the truth-table of a $c \log n$-bit Boolean function with* SAT*-oracle circuit complexity $\Omega(2^{\gamma n})$, then for every x of length n*

$$\left| \Pr_{r \in \Sigma^{t(n)}}[M(x, r) = 1] - \Pr_{r \in \Sigma^{b \log n}}[M(x, F_n(u, r) = 1] \right| \leq 0.1$$

Let L be a language in $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ and M be a $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ machine that accepts L and runs in $t(n)$ time. Using Theorem 10, the following is a $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$ machine that simulates M: Randomly pick u of size n^c – truth table of a function over $c \log n$ bits. By making a query to the NP oracle, check if u has circuit complexity at least $2^{\epsilon n}$. If not, then output "I do not know". Else, we can conclude that u has SAT-oracle circuit complexity at least $2^{\gamma n}$ for some $\gamma > 0$. Now simulate M using $F_n(u, r)$ as random bits for every $r \in \Sigma^{b \log n}$ and accept if the majority of simulations accept. Since with very high probability a randomly chosen u has circuit complexity at least $2^{\epsilon n}$, by Theorem 10 the simulation is correct. Thus L is in $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$. ◀

We can extend Theorem 7 to a subclass of $\mathrm{ZPP}_{tt}^{\mathrm{NP}}$.

▶ **Theorem 11.** *If for every function f in $\sharp\mathrm{NP}$ there exists an $(\frac{1}{10}, \frac{1}{10})$-$\mathrm{BPP}_{tt}^{\mathrm{NP}}$ pseudodeterministic approximation algorithm, then for every $k > 0$, there is a language L_k in $\mathrm{ZPP}_{tt}^{\mathrm{NP}} \cap \mathrm{S}_2^{\mathrm{P}}$ such that L_k cannot be computed by n^k-size Boolean circuits.*

Proof. If NP does not have polynomial-size circuits, then the statement of the theorem trivially holds. If NP has polynomial-size circuits, then by Sengupta's result [5], the Polynomial-time Hierarchy collapses to $\mathrm{S}_2^{\mathrm{P}}$. Thus $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ is in $\mathrm{S}_2^{\mathrm{P}}$. Thus by Theorem 9, if NP has polynomial-size circuits, then $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ is a subset of $\mathrm{ZPP}_{tt}^{\mathrm{NP}} \cap \mathrm{S}_2^{\mathrm{P}}$. By Lemma 8, under the hypothesis, $\mathrm{BPP}_{tt}^{\mathrm{NP}}$ does not have n^k-size circuits. The theorem follows. ◀

4 Constant-bit Influential algorithm for approximate counting

In this section we show that every probabilistic algorithm that computes an (ϵ, δ)-approximation to a function f can be transformed into a $O(1)$-bit influential algorithm.

▶ **Theorem 12.** *Let $\epsilon \in o(1)$, and $\delta \leq 1/3$. Let f be a function whose range is the integers that admits an (ϵ, δ)-approximation algorithm. Then there is a $O(1)$ influential-bit, $(O(\epsilon), \delta)$-approximation algorithm for f.*

Implicit in the work of Saks and Zhou [17] is a $O(\log n)$-bit influential, *absolute error* approximation algorithm for matrix powering. The main technical tool they use is that of *randomized rounding*. In the *relative error setting* that we are interested in, their rounding scheme does not work. We use an adaptive randomized rounding scheme that can handle relative errors and use this to design a constant-bit influential algorithm. Before we present our proof, we provide a high level overview of Saks and Zhou's proof adapted to our setting, and explain why it is not straight forward to apply in the relative error setting.

Let f be a function from Σ^* to integers such that n-bit integers are mapped to the range $[0, \cdots, 2^n]$. We say that an algorithm A computes (ϵ, δ), *absolute* approximation for f, if for every x, the value of $A(x)$ lies between $f(x) - \epsilon 2^n$ and $f(x) + \epsilon 2^n$ with probability at least $1 - \delta$. Let s be an integer such that $\epsilon 2^n$ is close to 2^{n-s}, thus the approximation error is close to 2^{n-s}. Saks and Zhou consider the following rounding operator $R_{b,s}(x)$, where b is a 4-bit integer. Subtract $2^{n-s} \times b$ from x and replace the last $n - s - 4$ bits of the resulting number with zeros. Saks and Zhou show that if z_1 and z_2 are any two good approximations of $f(x)$ (I.e, both z_1 and z_2 lie between $f(x) - \epsilon 2^n$ and $f(x) + \epsilon 2^n$), then for a random choice of b, $R_{b,s}(z_1) = R_{b,s}(z_2)$, and $R_{b,s}(z_1)$ is still a good approximation of $f(x)$ with high probability. From this it follows that any absolute error approximation algorithm can be made 4-bit influential. In the case of matrix powering, the rounding has to be applied for polynomially many entries, which will lead to an $O(\log n)$-bit influential algorithm for matrix powering.

For the above rounding scheme to work, it is critical that we know the value of s which depends on the value of approximation error which is at most $\epsilon 2^n$. However, in the relative error setting we do not a priori know the value of the approximation error, as it depends on $f(x)$. We could infer the of value of $\epsilon f(x)$ by looking at an output, and try to estimate s. However, since the approximation algorithm can produce multiple outputs, it is not possible to uniquely infer a value for s. We get around these problems by using additional (constant bits) randomness, and an adaptive rounding scheme. We now present a proof of Theorem 12.

Proof. Let f be a function from Σ^* to integers and let A be a (ϵ, δ)-approximation algorithm for f. For ease of presentation, we assume that $f(x) \leq 2^n$, where $n = |x|$. Consider the following algorithm. In this algorithm we set ℓ to 8 and p to $8 - \log(1/\epsilon)$. Note that p is negative. Given a number y in binary, and an integer z, we use $y\rfloor_z$ to denote the value obtained by replacing the last z bits of y with zeros. We use % for the modulo operator: $x\%y$ is the unique integer z in $\{0 \dots y-1\}$ satisfying $yr + z = x$ for some integer r. We say $x \equiv y\%z$ if $x\%y = z\%y$.

1. Input x of length n.
2. Choose m uniformly at random from $\{0 \dots 4\}$.
3. Choose r uniformly at random from $\{1 \dots 2^\ell\}$.
4. Run $A(x)$ and let y be the output.
5. Choose k_y so that $2^{k_y} \leq y < 2^{k_y+1}$.
6. Set $z_y = k_y + p + \{2, 1, 0, -1, -2\}$ so that $z_y \equiv m \ \% \ 5$.
7. Set $r_{z_y} = r \cdot 2^{(z_y - \ell)}$.
8. Output $(y - r_{z_y})\rfloor_{z_y}$.

We prove that this is a $O(1)$-bit influential algorithm for f, where the influential bits describe m and r. Fix an input x. The probability that A outputs a value in the range $[(1-\epsilon)f(x), (1+\epsilon)f(x)]$ is at least $1-\delta$. From now we assume that this event has happened. Let $\{y_1, y_2, \cdots, y_N\}$ be all possible outputs of $A(x)$ such that every $y_i \in (1 \pm \epsilon)f(x)$. Let y_{min} and y_{max} be the smallest and the largest of these values respectively. Since $\epsilon \leq 1/2$, there exists a $k \geq 0$ such that for every y_i, $2^k \leq y_i < 2^{k+2}$. Note that once the input x is fixed, k is also fixed.

We say $m \in \{0, \cdots, 4\}$ is *good* if all z_{y_i} $(1 \leq i \leq N)$ are the same (defined in Line 6). We will establish the following claim.

▶ **Claim 12.1.** *If we choose m randomly from $\{0, 1, 2, 3, 4\}$, m is good with probability $4/5$.*

Proof. Consider y_i and y_j, $i \neq j$. If both y_i and y_j lie between 2^k and 2^{k+1}, then clearly z_{y_i} equals z_{y_j}. Suppose that y_i lies between 2^k and 2^{k+1} and y_j lies between 2^{k+1} and 2^{k+2}. Thus $z_{y_i} \in \{k+p-2, k+p+1, k+p, k+p+1, k+p+2\}$ and $z_{y_j} \in \{k+p-1, k+p, k+p+1, k+p+2, k+p+3\}$. Note that both z_{y_i} and z_{y_j} are equal m modulo 5. If $(k+p-2) \equiv m\%5$, then the value of z_{y_i} equals $k+p-2$ and z_{y_j} equals $k+p+3$ and they differ. In all other cases, z_{y_i} equals z_{y_j}. The probability that randomly chosen $m \equiv (k+p-2)\%5$ is exactly $1/5$. Thus m is good with probability $4/5$. ◀

From now on we will assume that the event "m is good" has happened. Thus $z_{y_1} = z_{y_2} = \cdots = z_{y_N}$. For notational simplicity, we will denote this value by z. Note that r_z is formed by randomly picking $r \in \{1, \cdots 2^\ell\}$ and multiplying with $2^{(z-\ell)}$. We will prove the following claim that will be used later.

▶ **Claim 12.2.** *For every* $X \in [0 \dots 2^z]$,
1. $\frac{X}{2^z} - \frac{1}{2^\ell} \leq \Pr[r_z \leq X] \leq \frac{X}{2^z}$
2. $1 - \frac{X}{2^z} \leq \Pr[r_z > X] \leq 1 - (\frac{X}{2^z} - \frac{1}{2^\ell})$

Proof. Note that r_z is uniformly distributed in $\{\frac{2^z}{2^\ell}, 2 \cdot \frac{2^z}{2^\ell}, 3 \cdot \frac{2^z}{2^\ell}, \dots n^2 \cdot \frac{2^z}{2^\ell}\}$.
Thus, if $i\frac{2^z}{2^\ell} \leq X < (i+1)\frac{2^z}{2^\ell}$, there are i values of r_z that are at most X. We can write i as $\lfloor \frac{X \cdot n^\ell}{2^z} \rfloor \leq \frac{X \cdot 2^\ell}{2^z}$, giving us $\Pr[r_z \leq X] = \frac{i}{2^\ell} \leq \frac{X \cdot 2^\ell}{2^\ell \cdot 2^z} = \frac{X}{2^z}$ and $\Pr[r_z > X] = 1 - \frac{i}{2^\ell} \geq 1 - \frac{X}{2^z}$
Similarly, $i \geq \frac{X \cdot 2^\ell}{2^z} - 1$, so $\Pr[r_z \leq X] = \frac{i}{2^\ell} \geq \frac{X}{2^z} - \frac{1}{2^\ell}$ and $\Pr[r_z > X] = 1 - \frac{i}{2^\ell} \leq 1 - (\frac{X}{2^z} - \frac{1}{2^\ell})$ ◀

▶ **Lemma 13.** *For every good* m*, if we randomly choose* $r \in \{1, \cdots, 2^\ell\}$*, then for at least* $223/256$ *of the possible* r*, the following holds: For every* $i \neq j$ *(*$1 \leq i, j \leq r$*)* $(y_i - r_z)\rfloor_z = (y_j - r_z)\rfloor_z$.

Proof. The $\rfloor_z$ operation can be viewed as dividing the interval $[0, 2^n]$ into subintervals $[0, 2^z - 1], [2^z, 2 \cdot 2^z - 1], [2 \cdot 2^z, 3 \cdot 2^z - 1] \dots [2^n - 2^z, 2^n]$, and mapping the contents of each interval to its left endpoint. Note that $y_{max} - y_{min}$ is at most $2f(x)\epsilon$. Since $f(x) \leq 2^{k+2}$ and $2^z \geq 2^{k+p-2}$, $y_{max} - y_{min}$ is at most $2^{k+3}\epsilon$, which is at most $2^z/8$. Since the size of each interval is 2^z and the difference between y_{max} and y_{min} is less than the size of the interval, either both y_{min} and y_{max} lie in the same interval or lie in two contiguous intervals. We consider both cases.

Case 1. Both y_{max} and y_{min} are in the same interval, say $[i \cdot 2^z, (i+1) \cdot 2^z]$ for some i. If subtracting r_z causes them to be in different intervals, r_z must be large enough to move y_{min} to a new interval, but not so large that y_{max} moves too. That is, $r_z > (y_{min} - i \cdot 2^z)$ and $r_z \leq (y_{max} - i \cdot 2^z)$. They are in the same interval when $r_z \leq (y_{min} - i \cdot 2^z)$ or $r_z > (y_{max} - i \cdot 2^z)$.
So, the probability that y_{min} and y_{max} are in the same interval is

$$\begin{aligned} &\Pr[r_z > (y_{max} - i \cdot 2^z)] + \Pr[r_z \leq (y_{min} - i \cdot 2^z)] \\ &\quad\geq 1 - \frac{y_{max} - i \cdot 2^z}{2^z} + \frac{y_{min} - i \cdot 2^z}{2^z} - \frac{1}{2^\ell} \\ &\quad= 1 - \frac{y_{max} - y_{min}}{2^z} - \frac{1}{2^l} \\ &\quad\geq 223/256 \end{aligned}$$

The first inequality follows from Claim 12.2. The second follows from the conclusion that $y_{max} - y_{min}$ is at most $2^z/8$. The last inequality follows because $\ell = 8$.

Case 2. Now consider the other case, where y_{min} is in the interval $[i \cdot 2^z, (i+1) \cdot 2^z]$ and y_{max} is in $[(i+1) \cdot 2^z, (i+2) \cdot 2^z]$. If subtracting r_z causes y_{max} and y_{min} to be in different intervals, either r_z is not large enough to move y_{max} into the same interval as y_{min}, or r_z is so large that it moves y_{min} into a new interval as well. In this case, the condition is $r_z > (y_{min} - i \cdot 2^z)$ or $r_z \leq (y_{max} - (i+1) \cdot 2^z)$. Then, the probability that y_{min} and y_{max} end up in different intervals is

$$\begin{aligned} &\Pr[r_z > (y_{min} - i \cdot 2^z)] + \Pr[r_z \leq (y_{max} - (i+1) \cdot 2^z)] \\ &\quad\leq 1 - (\frac{y_{min} - i \cdot 2^z}{2^z} - \frac{1}{2^\ell}) + \frac{y_{max} - (i+1) \cdot 2^z}{2^z} \\ &\quad= 1 - \frac{y_{min}}{2^z} + i + \frac{1}{2^\ell} + \frac{y_{max}}{2^z} - i - 1 \\ &\quad= \frac{y_{max} - y_{min}}{2^z} + \frac{1}{2^\ell} \\ &\quad\leq 33/256 \end{aligned}$$

The first inequality follows from Claim 12.2. The second follows from the conclusion that $y_{max} - y_{min}$ is at most $2^z/8$. ◀

Our next claim shows that the operation $\rfloor z$ preserves the approximation (up to a constant factor).

▶ **Claim 13.1.** *If $y \in f(x)(1 \pm \epsilon)$, then $(y - r_z)\rfloor_z \in f(x)(1 \pm O(\epsilon))$.*

Proof. Subtracting r_z and taking the $\rfloor_z$ only decrease, so this is at most y, which is at most $f(x)(1+\epsilon)$. For a given y, the minimum value of $(y - r_z)\rfloor_z$ occurs when r is as large as possible and $\rfloor_z$ changes the last z bits from 1 to 0. r_z is at most 2^z. $\rfloor_z$ can decrease the input by at most 2^z. Thus, $(y - r_z)\rfloor_z$ is at least $y - 2^z - 2^z$. Consider the following inequalities.

$$\begin{aligned} 2^{z+1} &\leq 2^{k+2+p+2} \\ &= 2^{k+13-\log 1/\epsilon} \\ &\leq f(x)(2^{13}\epsilon)(\text{ as } f(x) \geq 2^k) \end{aligned}$$

Since $(y-r_z)\rfloor_z \geq y-2^{z+1}$, and $y \geq f(x)(1-\epsilon)$, we obtain that $(y-r_z)\rfloor_z \geq f(x)(1-2^{14}\epsilon)$. ◀

Now we can finish the proof of Theorem 12. The influential bits of the algorithm are m and r. Note that the number of influential bits is constant. Let us call the set of r for which the consequence of Lemma 13 holds as *good*. Note that for every fixing of good m and r, the output of the algorithm is unique and is a good approximation, with probability at least $1 - \delta$, by Lemma 13 and Claim 13.1. Finally the fraction of bad m and r is at most $1/5 + 33/256$ which is at most $1/3$. Thus the algorithm is a constant-bit influential algorithm.

The statement of Theorem 12 assumed that $\epsilon \in o(1)$. However, the proof goes through for any sufficiently small ϵ. We have the following as a corollary of Theorem 12. ◀

▶ **Theorem 14.** *For every problem in $\sharp$NP, there is a $O(1)$-bit influential, $(1/n, 1/2^n)$-approximation algorithm.*

References

1. S. Aaronson. Oracles are subtle but not malicious. In *IEEE Conference on Computational Complexity*, pages 340–354, 2006.
2. S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
3. M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of np-witnesses using an np-oracle. *Inf. Comput.*, 163(2):510–526, 2000.
4. N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996.
5. J-Y. Cai. $\mathrm{S}^{\mathrm{p}}_2$ subseteq $\mathrm{zpp}^{\mathrm{np}}$. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 620–629, 2001.
6. J. Y. Cai, R. Lipton, L. Longpré, M. Ogihara, K. Regan, and D. Sivakumar. Communication complexity of key agreement on small ranges. In *STACS*, pages 38–49, 1995.
7. E. Gat and S. Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:136, 2011.
8. O. Goldreich, S. Goldwasser, and D. Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 127–138, 2013.
9. S. Goldwasser and O. Grossman. Bipartite perfect matching in pseudo-deterministic NC. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 87:1–87:13, 2017.

10 S. Goldwasser, O. Grossman, and D. Holden. Pseudo-deterministic proofs. *CoRR*, abs/1706.04641, 2017.

11 O. Grossman. Finding primitive roots pseudo-deterministically. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:207, 2015.

12 O. Grossman and Y. Liu. Reproducibility and pseudo-determinism in log-space. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:48, 2018.

13 R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

14 A. Klivans and D. Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002.

15 J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998.

16 I. Oliveira and R. Santhanam. Pseudodeterministic constructions in subexponential time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 665–677, 2017.

17 M. Saks and S. Zhou. BP $_{\mathrm{h}}$space(s) subseteq dspace(s$^{3/2}$). *J. Comput. Syst. Sci.*, 58(2):376–403, 1999.

18 R. Santhanam. Circuit lower bounds for merlin–arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

19 L. Stockmeyer. The complexity of approximate counting (preliminary version). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 118–126, 1983.

20 L. Stockmeyer. On approximation algorithms for #p. *SIAM J. Comput.*, 14(4):849–861, 1985.

21 N. V. Vinodchandran. A note on the circuit complexity of PP. *Theor. Comput. Sci.*, 347(1-2):415–418, 2005.