Derandomization and Distinguishing Complexity

Eric Allender*
Rutgers University
allender@cs.rutgers.edu

Michal Koucký*
Rutgers University
mkoucky@paul.rutgers.edu

Detlef Ronneburger*
Rutgers University
detlef@paul.rutgers.edu

Sambuddha Roy*
Rutgers University
samroy@paul.rutgers.edu

Abstract

We continue an investigation of resource-bounded Kolmogorov complexity and derandomization techniques begun in [2, 3].

*We introduce nondeterministic time-bounded Kolmogorov complexity measures (*KNt *and* KNT*) and examine the properties of these measures using constructions of hitting set generators for nondeterministic circuits [22, 26].*

We observe that KNt *bears many similarities to the nondeterministic distinguishing complexity CND of [8]. This motivates the definition of a new notion of time-bounded distinguishing complexity* KDt*, as an intermediate notion with connections to the class* FewEXP*. The set of* KDt*-random strings is complete for* EXP *under* P/poly *reductions.*

Most of the notions of resource-bounded Kolmogorov complexity discussed here and in the earlier papers [2, 3] have close connections to circuit size (on different types of circuits). We extend this framework to define notions of Kolmogorov complexity KB *and* KF *that are related to branching program size and formula size, respectively. The sets of* KB*- and* KF*-random strings lie in* coNP*; we show that oracle access to these sets enables one to factor Blum integers. We obtain related intractability results for approximating minimum formula size, branching program size, and circuit size.*

The NEXP $\subseteq$ NC^1 *and* NEXP $\subseteq$ L/poly *questions are shown to be equivalent to conditions about the* KF *and* KB *complexity of sets in* P.

*Partially supported by NSF grant CCR-0104823.

1 Introduction

This paper continues a line of research begun in [2, 3], in which recent progress in derandomization techniques is employed to provide new insights about resource-bounded Kolmogorov complexity. One topic that was *not* discussed in the earlier papers is derandomization techniques for *nondeterministic* circuits, as provided by [22] and [26]. We will turn our attention to that task in Section 2, after first laying some groundwork by presenting our basic definitions.

In the earlier papers [2, 3] we focused on notions of resource-bounded Kolmogorov complexity, including Kt and KT. The first of these was originally defined and studied by Levin [19]. In this paper we will be introducing several more notions of resource-bounded Kolmogorov complexity. In order to have a uniform framework for these new definitions, we need to modify the definitions of Kt and KT in minor ways that affect none of the theorems proved in the earlier work.

Definition 1 *Let U be a deterministic Turing machine.*

$$\begin{aligned}\mathrm{Kt}_U(x) &= \min\{|d| + \log t \,:\, \forall b \in \{0,1,*\} \\ &\qquad \forall i \le n+1\ U(d,i,b) \text{ runs in} \\ &\qquad \text{time } t \text{ and accepts iff } x_i = b\} \\ \mathrm{KT}_U(x) &= \min\{|d| + t \,:\, \forall b \in \{0,1,*\} \\ &\qquad \forall i \le n+1\ U(d,i,b) \text{ runs in} \\ &\qquad \text{time } t \text{ and accepts iff } x_i = b\}\end{aligned}$$

Here, we say that $x_i = *$ if $i > |x|$.

As usual, we will choose a fixed "optimal" Turing machine U and use the notation Kt and KT to refer to Kt_U and KT_U. However, the definition of "optimal" Turing machine depends on the measure which is under considera-

tion. For instance, U is Kt-*optimal* if for any Turing machine U' there exists a constant $c \geq 0$ such that for all x, $\mathrm{Kt}_U(x) \leq \mathrm{Kt}_{U'}(x) + c \log |x|$. Notice that there is an additive logarithmic term instead of the "usual" additive constant. This comes from the slight slow-down that is incurred in the simulation of U' by U. Similarly, U is KT-*optimal* if for any Turing machine U' there exists a constant $c > 0$ such that for all x, $\mathrm{KT}_U(x) \leq c\mathrm{KT}_{U'}(x) \log \mathrm{KT}_{U'}(x)$. The existence of optimal machines for Kt and KT complexity follows via standard arguments. Definitions of Kt and KT can be relativized to yield measures Kt^A and KT^A by providing U with access to oracle A.

There is not space here to survey all of the earlier work on resource-bounded Kolmogorov complexity. Briefly, the considerations that cause us to focus on these particular definitions are

- The KT^A-complexity of a string of length 2^n (the truth table of a Boolean function f) is closely related to the size of A-oracle circuits computing f.
- Levin's Kt complexity is essentially the same thing as KT^A for any set A complete for E.
- These definitions facilitate the presentation of results relating new techniques in derandomization to traditional notions of Kolmogorov complexity.

In [3] the sets of strings with "high" complexity under these measures were shown to be complete for various complexity classes. Although these completeness results hold for a wide range of possible choices of "high", for convenience we settle on the following definition.

Definition 2 *For any Kolmogorov complexity measure* $\mathrm{K}\mu$*, define* $R_{\mathrm{K}\mu}$ *to be the set* $\{x : \mathrm{K}\mu(x) \geq |x|/2\}$.

A useful measure of the complexity of a language L (studied in [1, 2]) is the measure of the simplest strings in L.

Definition 3 *Let* L *be a language and let* $\mathrm{K}\mu$ *be a Kolmogorov complexity measure. We define the Kolmogorov complexity of* L *for length* n *as*

$$\mathrm{K}\mu_L(n) = \min\{\mathrm{K}\mu(x) \,:\, |x| = n \text{ and } x \in L\}$$

If $L \cap \Sigma^n = \emptyset$ *then* $\mathrm{K}\mu_L(n)$ *is undefined.*

2 Nondeterministic Kolmogorov Complexity

Miltersen and Vinodchandran [22] proved that if there is a set in NE $\cap$ coNE that does not have "strong" nondeterministic circuits of subexponential size, then there is a hitting-set generator computable in NP for co-nondeterministic circuits. Shaltiel and Umans [26] subsequently presented a better construction of a hitting-set generator that hits co-nondeterministic as well as nondeterministic circuits.

We recall some standard definitions:

Definition 4 *A* SNP-procedure (Strong NP procedure) *computing a function* f *is a polynomial time nondeterministic procedure, so that every computation path on input* x *either produces* $f(x)$ *or rejects. Furthermore, at least one computation path must produce* $f(x)$.

We will also refer to functions computable in SNP/log. For this, we assume that there is an advice function $h(n)$ providing a string of length $O(\log n)$, and a nondeterministic machine as above that produces $f(x)$ on every non-rejecting computation path on input $(x, h(|x|))$. We place no restrictions on the behavior of the nondeterministic machine on inputs (x, z) where $z \neq h(|x|)$.

Definition 5 *A* nondeterministic Boolean circuit C *contains, in addition to* AND, OR, *and* NOT *gates, choice gates of fan-in* 0. *The circuit evaluates to* 1 *on an input* x*, and we say that* $C(x) = 1$*, if there is some assignment of truth values to the choice-gates that makes the circuit evaluate to* 1. *A co-nondeterministic circuit* C *is defined similarly: the circuit evaluates to* 1 *on an input* x*, and we say that* $C(x) = 1$*, if every assignment of truth values to the choice-gates makes the circuit evaluate to* 1. *Otherwise* $C(x) = 0$.

Similarly, a strong nondeterministic circuit C *computing a function* f *has, in addition to its usual output, an extra output bit, called the flag. For any input* x*, and any setting of the choice-gates, if the flag is on, the circuit should output the correct value of* $f(x)$*. Furthermore, for any* x*, there should be some setting of the choice-gates that turns the flag on. It is easy to see that a Boolean function* f *has a strong nondeterministic circuit of size* $O(s(n))$ *if and only if* f *has a nondeterministic circuit of size* $O(s(n))$ *and a co-nondeterministic circuit of size* $O(s(n))$.

Definition 6 *A* hitting set generator *for a class of circuits* $\mathcal{C}$ *and threshold* α *is a procedure* G *that maps strings of length* n *to a set* H_n *of polynomial size with the property that, for every circuit in* $\mathcal{C}$ *on* n *inputs that accepts at least* $\alpha 2^n$ *strings in* Σ^n*,* C *accepts an element of* H_n.

In order to see what the techniques of [22, 26] tell us about Kolmogorov complexity, it is necessary to present nondeterministic analogs of Kt and KT. We call the new notions KNt and KNT.

Definition 7 *Let U be a fixed nondeterministic Turing machine.*

$$\begin{aligned}\mathrm{KNt}_U(x) &= \min\{|d| + \log t : \forall b \in \{0,1,*\} \\ &\qquad \forall i \le n+1\ U(d,i,b) \textit{ runs in} \\ &\qquad \textit{time } t \textit{ and accepts iff } x_i = b\} \\ \mathrm{KNT}_U(x) &= \min\{|d| + t : \forall b \in \{0,1,*\} \\ &\qquad \forall i \le n+1\ U(d,i,b) \textit{ runs in} \\ &\qquad \textit{time } t \textit{ and accepts iff } x_i = b\}\end{aligned}$$

As in the definition for Kt and KT, we have to be careful of the properties we require of the optimal Turing machine. We define KNt as KNt_U where the optimal machine U has the property that for all U', we have $\mathrm{KNt}_U(x) \le \mathrm{KNt}_{U'}(x) + c$ We define KNT as KNT_U, such that for all U', we have $\mathrm{KNT}_U(x) \le c \cdot \mathrm{KNT}_{U'}(x)$ for some constant c.

Remark: In precisely the same way that $\mathrm{KT}(x)$ is polynomially related to the size of (deterministic) circuits computing the function whose truth table is given by x, it is easy to see that KNT is polynomially related to *strong nondeterministic* circuit size.

Theorem 8 *The following are equivalent:*

1. $\exists \epsilon > 0 \forall n \exists x \in \Sigma^n\ \mathrm{KNT}(x) + \log|x| > 2^{\epsilon \mathrm{KNt}(x)}$. *(That is,* KNT *and* KNt *are nearly as far apart as possible.)*
2. $\exists A \in$ NE/lin $\cap$ co-NE/lin, $\exists a$ *such that* A *requires strong nondeterministic circuits of size* 2^{an}.
3. $\exists A \in$ NE/lin $\cap$ co-NE/lin, $\exists a$ *such that* A *requires nondeterministic circuits of size* 2^{an}.
4. *For all dense*[1] A *in* coNP/poly, $\mathrm{KNt}_A(n) = O(\log n)$.
5. *For all dense* A *in* NP/poly, $\mathrm{KNt}_A(n) = O(\log n)$.
6. *For all dense* A *in* NP/poly $\cap$ coNP/poly, $\mathrm{KNt}_A(n) = O(\log n)$.
7. *There exist* SNP/log *computable hitting set generators for nondeterministic linear-size circuits and threshold* $\frac{1}{2}$ *(and similar conditions for co-nondeterministic and strong circuits).*

Remark: We wish to call attention to the equivalence of conditions 4 and 5. For some notions of complexity such as KT, there are dense sets in coNP with essentially maximal KT complexity (such as R_{KT}), whereas there are good reasons to believe that every dense language in NP/poly has low KT-complexity. (Rudich gives evidence for this conjecture in [25].)

[1] A language is *dense* if it contains at least a polynomially-large fraction of the strings of each length.

Proof. (1 $\Leftrightarrow$ 2) This equivalence is proved similarly to related statements in [2]. Given any sequence of strings $x_1, x_2, \ldots$ with $|x_m| = n = 2^m$, where $\mathrm{KNT}(x_m)$ is large and $\mathrm{KNt}(x_m)$ is small (and must in fact be logarithmic, since KNT is always linear at most) by concatenation one can construct the characteristic sequence of a language A in NE/lin $\cap$ co-NE/lin that requires large strong nondeterministic circuits. For the converse, given any such language A, the prefices of its characteristic sequence have logarithmic KNt complexity and large KNT complexity.

(2 $\Rightarrow$ 3) We prove the contrapositive, $\neg 3 \Rightarrow \neg 2$. Thus every $A \in$ NE/lin $\cap$ co-NE/lin has "small" nondeterministic circuits (that is, of size less than 2^{an} for any $a > 0$). Thus $\overline{A} \in$ NE/lin $\cap$ co-NE/lin, and hence by hypothesis has "small" nondeterministic circuits. This yields co-nondeterministic circuits for A; we can combine the two circuits to get strong nondeterministic circuits for A. This proves $\neg 2$. (Similar observations are made by Shaltiel and Umans [26].)

(3 $\Rightarrow$ 2) This is trivial; a strong nondeterministic circuit yields a nondeterministic circuit of roughly the same size.

(2 $\Rightarrow$ 4, 5, 6, and 7) In Corollaries 10 and 12 of [26], Shaltiel and Umans show that there is a constant c and a function $G(x,n)$ computable in deterministic polynomial time with the property that if x is a string of length n^c such that $\mathrm{KNT}(x) > |x|/2$ (i.e., if x is the truth table of a function requiring large size on strong nondeterministic circuits) then $G(x,n)$ produces a set $H_{x,n}$ that is a hitting set for both nondeterministic and co-nondeterministic linear-size circuits with threshold $\frac{1}{2}$.

It is now straightforward to obtain a hitting set generator in SNP/log; with logarithmic advice we can nondeterministically guess and verify a string x that is a truth table for a particular language in NE/lin $\cap$ co-NE/lin, and then run the generator G.

It is easy to see that any string in the hitting set output by a SNP/log computable hitting set generator has low KNt complexity; this shows that any set of density one-half or greater accepted by linear-size nondeterministic or co-nondeterministic circuits contains some strings of low KNt complexity. The more general statements now follow by an easy padding argument.

The implications (7 $\Rightarrow$ 6) (4 $\Rightarrow$ 6), and (5 $\Rightarrow$ 6) are either trivial or follow via the argument above. Thus it suffices to prove (6 $\Rightarrow$ 2).

(6 $\Rightarrow$ 2). Define $A = \{x : |x| = 5m \text{ and } \mathrm{KNt}(x) > m\}$. We claim that A is in NE/lin $\cap$ co-NE/lin. To see this, recall that for a string x of length $5m$, $\mathrm{KNt}(x) \le m$ implies $\exists d, |d| \le m, \forall i\ U(d,i,b)$ has an accepting path iff $x_i = b$, where U is a universal nondeterministic Turing

machine running for 2^m steps. In order to enumerate all x's of length $5m$ that have $\mathrm{KNt}(x) \leq m$, we will exclude from consideration those d's that are not valid descriptions of strings. We define α to be the number of d's that are indeed valid descriptions of strings of length $5m$, (i.e., there exists an x for which $\forall i\, U(d,i,b)$ has an accepting path iff $x_i = b$), and we define β to be the number of "recognizably bad" descriptions, that is, those for which $\forall i \leq 5m+1 \exists b \in \{0,1,*\}, U$ accepts (d,i,b) and for some i and some $b' \neq b \in \{0,1,*\}, U$ accepts both (d,i,b) and (d,i,b'). Our SNP machine takes in α and β as advice (each of length $O(m)$). First it guesses β "recognizably bad" descriptions and verifies that they are indeed bad by guessing accepting paths for both (d,i,b) and (d,i,b'). Then it guesses α other strings (corresponding to candidate "good" d's), and guesses accepting paths for all of them and prints out the corresponding strings. All of this takes time exponential in m. Now we can accept x if and only if it is not in the list that has been generated.

Now we need to show that A requires large strong nondeterministic circuits. Assume otherwise, so that for every c there is some n such that there is a strong nondeterministic circuit of size $2^{n/c}$ deciding A for inputs of length n. Then we can construct a dense language $B \in$ NP/poly $\cap$ coNP/poly of the form $B = \{y : |y| = n$ and the prefix of y of length $c_n \cdot \log n$ is in $A\}$ where c_n is chosen (nonuniformly) to be as large as possible, so that the membership test for A can be implemented in size n via a strong nondeterministic circuit. By assumption, the sequence of numbers (c_n) is unbounded. It follows that $\mathrm{KNt}_B(n) \neq O(\log n)$. □

Most work on derandomizing nondeterministic circuits has been done with the aim of providing weak hypotheses that imply AM = NP. The conditions of the preceding theorem are not known yield this conclusion (although it is obvious that they imply AM $\subseteq$ NP/log); in order to imply AM = NP it is sufficient for the language A in item 3 to be in NE $\cap$ co-NE instead of NE/lin $\cap$ co-NE/lin [22]. It is worth mentioning that we also obtain another partial derandomization.

Theorem 9 *If there exists* $A \in$NE/lin $\cap$ co-NE/lin, *such that* A *requires strong nondeterministic circuits of size* 2^{an}, *for some* $a > 0$, *then* AM $\in \mathrm{P}^{\mathrm{NP}[\log n]}$.

Proof. As in [22], to determine if x is in a set $B \in$ AM, we model the Arthur-Merlin game using a nondeterministic circuit with input x and some probabilistic inputs y. Let C_x be the result of hardwiring the bits of x into this circuit; then $x \in B \Rightarrow C_x$ accepts every y, and $x \notin B \Rightarrow C_x$ rejects at least half of the strings y. Thus it suffices to use our NP oracle to determine if there is a string y that is rejected by C_x. By parts 7 and 4 of the preceding theorem, if such a string y exists, then there is such a string with $\mathrm{KNt}(y) = O(\log n)$.

Thus it suffices to design a $\mathrm{P}^{\mathrm{NP}[\log n]}$ procedure to determine if there is a string y with $\mathrm{KNt}(y) \leq c \log n$ such that the nondeterministic circuit C_x rejects y.

As in the proof of (6 $\Rightarrow$ 2) of the previous theorem, let α be the number of good descriptions of length at most $c \log n$ and let β be the number of "recognizably bad" descriptions d of length at most $c \log n$. The numbers α and β can be computed in $O(\log n)$ queries to an NP oracle of the form "do there exist $\geq j$ strings $(d_1, d_2, \cdots, d_j)$ of length at most $c \log n$ such that for all m and all $i \leq |y|+1$ there is a $b \in \{0,1,*\}$ such that $U(d_m, i, b)$ has an accepting path?" and "do there exist $\geq j$ strings $(d_1, d_2, \cdots, d_j)$ of length at most $c \log n$ such that for all m, i there is a b such that $U(d_m, i, b)$ accepts and there is some $i \leq |y|+1$ for which there are $b \neq b' \in \{0,1,*\}$ such that $U(d_m, i, b)$ and $U(d_m, i, b')$ each have an accepting path?" Having computed α and β we can ask one more query to an NP oracle to determine if there are β bad descriptions and α good descriptions such that C_x accepts all of the strings y described by the α good descriptions. □

2.1 Nondeterministic Derandomization

It is frequently the case that hardness assumptions are in fact *equivalent* to the existence of derandomization constructions; for a survey, see [11]. To the best of our knowledge, it has not been stated explicitly that the hitting set constructions of [22, 26] are in fact equivalent to the hardness assumptions they use, although it follows easily from well-known techniques [16].

Theorem 10 *The following are equivalent:*

1. $\exists A \in$ NE $\cap$ co-NE, $\exists a > 0$ *such that* A *requires strong nondeterministic circuits of size* 2^{an}.
2. $\exists A \in$ NE $\cap$ co-NE, $\exists a > 0$ *such that* A *requires nondeterministic circuits of size* 2^{an}.
3. *There is a* SNP*-computable hitting set generator for linear-size nondeterministic circuits and threshold* $\frac{1}{2}$.
4. *There is a* SNP*-computable hitting set generator for linear-size co-nondeterministic circuits and threshold* $\frac{1}{2}$.
5. *There is a* SNP*-computable hitting set generator for linear-size strong nondeterministic circuits and threshold* $\frac{1}{2}$.

3 Distinguishing Complexity

One of the first types of resource-bounded Kolmogorov complexity to be studied was "distinguishing" complexity. For more on the history of this notion, see [8], where the following notion of nondeterministic distinguishing complexity was introduced.

Definition 11 *Let p be a polynomial, and let U be a universal nondeterministic Turing machine. $CND^p(x)$ is defined to be the minimum $|d|$ such that $U(d, y)$ accepts in time $p(|x|)$ if and only if $y = x$.*

KNt and CND complexity are closely related, in that they agree on strings of logarithmic complexity, in the following sense.

Proposition 12 *Let c be given.*

- *There is a polynomial p and a $d \in \mathbb{N}$ such that if* $\mathrm{KNt}(x) < c \log |x|$ *then* $CND^p(x) < d \log |x|$.
- *For all polynomials p there is a $d \in \mathbb{N}$ such that if* $CND^p(x) < c \log |x|$ *then* $\mathrm{KNt}(x) < d \log |x|$.

(One consequence of this proposition is that the proof of Theorem 9 could have been presented in terms of CND complexity, instead of KNt complexity.) This motivates the definition of notions of distinguishing complexity, having the flavor of KNt and Kt.

Definition 13 *Let U_1 be a fixed nondeterministic Turing machine, and let U_2 be a fixed deterministic Turing machine.*

$$\begin{aligned} \mathrm{KNDt}_{U_1}(x) &= \min\{|d| + \log t : \forall y \in \Sigma^{|x|}\ U_1(d, y) \\ &\qquad \textit{runs in time } t \textit{ and accepts iff } x = y\} \\ \mathrm{KDt}_{U_2}(x) &= \min\{|d| + \log t : \forall y \in \Sigma^{|x|}\ U_2(d, y) \\ &\qquad \textit{runs in time } t \textit{ and accepts iff } x = y\} \end{aligned}$$

Again, we have to be careful about the properties we require of the optimal Turing machine. We define KNDt as KNDt_U where the optimal machine U has the property that for all U', we have $\mathrm{KNDt}_U(x) \leq \mathrm{KNDt}_{U'}(x) + c$ We define KDt as KDt_U, such that for all U', we have $\mathrm{KDt}_U(x) \leq \mathrm{KDt}_{U'}(x) + c \log |x|$ for some constant c.

We observe that KNDt is essentially the same thing as KNt, up to logarithmic terms. Showing that $\mathrm{KNDt}(x) \leq \mathrm{KNt}(x) + O(\log |x|)$ is an easy exercise. Conversely, if $\mathrm{KNDt}(x)$ is small (using description d), then a nondeterministic machine, given (d', i, b) where $d' = (d, |x|)$, can guess $x \in \Sigma^{|x|}$ and if $U(d, x)$ accepts, then accept iff the ith bit of x is b. Analysis of the run times easily yields that $\mathrm{KNDt}(x) \leq \mathrm{KNt}(x) + O(\log |x|)$. Since KNDt is indistinguishable from KNt from our standpoint, we will not refer to KNDt any further.

Theorem 14 $\mathrm{KNDt}(x) = \mathrm{KNt}(x) + \Theta(\log |x|)$

This leads us to ask if KDt is similarly related to Kt. At first, it might seem that they *are* closely related.

Proposition 15 *R_{Kt} and R_{KDt} are both complete for* EXP *under* P/poly *reductions.*

Proof. For R_{Kt} this is proved in [3], and in fact hardness holds for any dense set containing no strings of low Kt-complexity. Since $\mathrm{Kt}(x) > \mathrm{KDt}(x) - O(\log |x|)$ it follows that R_{KDt} is also hard for EXP. Membership in EXP is easy to show. □

Nonetheless, if Kt and KDt are polynomially related, it implies that FewEXP is equal to EXP. In order to state the connection more precisely, we need the following definition.

Definition 16 *We say that* FewEXP search instances are EXP-solvable *if, for* every NEXP *machine N and every k there is an* EXP *machine M with the property that if N has fewer than $2^{|x|^k}$ accepting paths on input x, then $M(x)$ produces one of these accepting paths as output if there is one. We say that* FewEXP decision instances are EXP-solvable *if, for* every NEXP *machine N and every k there is an* EXP *machine M with the property that if N has fewer than $2^{|x|^k}$ accepting paths on input x, then $M(x)$ accepts if and only if $N(x)$ accepts.*

Remark: Note that we do not require that N is a FewEXP machine; it need not have a small number of accepting paths on *every* input.

Theorem 17 *The following statements are equivalent.*

1. $\forall x, \mathrm{Kt}(x) = \mathrm{KDt}(x)^{O(1)}$
2. FewEXP *search instances are* EXP-*solvable.*
3. FewEXP *decision instances are* EXP/poly-*solvable.*
4. $\forall\, L \in \mathrm{P}, \mathrm{Kt}_L(n) = (\log |L^{=n}| + \log n)^{O(1)}$
5. $\forall x, \forall y, \mathrm{KDt}(x) = (\mathrm{KDt}(xy) + \log |xy|)^{O(1)}$

Remark: This theorem has a similar flavor to a theorem of [12] concerning C^p and CD^p complexity. However, although we make use of the techniques of [12] we do not see a way to formulate an equivalent condition using the C^p and CD^p measures.

Remark: The fifth condition of the preceding theorem deserves some comment. For all of the other resource-bounded Kolmogorov complexity measures $\mathrm{K}\mu$ studied in this paper (other than KDt) it is easy to see that the following three conditions are equivalent:

- For all $A \in$ NP $\mathrm{K}\mu_A(n) = \log^{O(1)} n$.
- For all $A \in$ P $\mathrm{K}\mu_A(n) = \log^{O(1)} n$.
- For all $A \in$ DLOGTIME-uniform AC^0 $\mathrm{K}\mu_A(n) = \log^{O(1)} n$.

(For example, see Theorem 3 in [2].) The simple observation that forms the main part of the proof of this equivalence is the fact that for all x and y, $\mathrm{K}\mu(x)$ can be bounded by $\mathrm{K}\mu(xy) + \log|xy|$.

Proof of Theorem 17. It is immediate that $2 \Rightarrow 3$ and $1 \Rightarrow 5$. We will now prove $3 \Rightarrow 1$, $1 \Rightarrow 4$, $4 \Rightarrow 2$, and $5 \Rightarrow 1$.

($3 \Rightarrow 1$) Consider a NEXP machine M that on input $(d, 1^t, i, b, n)$ guesses a string $y \in \{0,1\}^n$, runs $U(d, y)$ for 2^t steps and then accepts iff $y_i = b$ and $U(d, y)$ accepts. If d is a distinguishing description for a string $x \in \{0,1\}^n$ and t is sufficiently large, then there is exactly one accepting path of M on input $(d, 1^t, i, x_i, |x|)$; there is no accepting path of M on $(d, 1^t, i, \overline{x_i}, |x|)$, for all $1 \leq i \leq |x|$. By our assumption, there is a deterministic machine N running in exponential time, that on input $(d, 1^t, i, b, |x|)$, given some polynomial advice h, can decide whether M accepts $(d, 1^t, i, b, |x|)$ or not.[2] Thus, given d, t, $|x|$ and the advice h, we can generate x bit by bit in time exponential in $(|d| + t + \log|x| + |h|)^{O(1)}$. Since $\mathrm{KDt}(x) \geq \log|x|$, $\mathrm{Kt}(x) \leq \mathrm{KDt}(x)^{O(1)}$.

($1 \Rightarrow 4$) Using hashing the authors of [8] show that for any set L there is a polynomial time algorithm with oracle access to L, such that for every $x \in L$ there is a description d_x of length $2\log\left|L^{=|x|}\right| + O(\log|x|)$, such that the algorithm accepts (z, d_x) if and only if $z = x$. Consider $L \in$ P. Then the oracle access to L is not necessary and for every $x \in L$ we know $\mathrm{KDt}(x) = 2\log\left|L^{=|x|}\right| + O(\log|x|)$. Assuming that KDt and Kt are polynomially related we obtain $\mathrm{Kt}(x) \leq (\log\left|L^{=|x|}\right| + \log|x|)^{O(1)}$.

[2]Note that it would have been sufficient to use a formally weaker assumption, dealing only with the case where there is a single accepting path.

($4 \Rightarrow 2$) Let L be decidable by a nondeterministic machine N running in time 2^{n^k}, for $k \geq 1$. Define the set $C = \{w10^x$: where $w \in \{0,1\}^{2^{|x|^k}}$ is a witness that $N(x)$ accepts $\}$. (Here, we identify x with the integer having binary representation $1x$.) Clearly $C \in P$. Let x be a string, such that $N(x)$ has few accepting paths, i.e., $|C^{=n_x}| \leq 2^{|x|^{O(1)}}$, where $n_x = 2^{|x|^k} + x + 1$. By assumption, there is a witness w with $\mathrm{Kt}(w10^x) \leq |x|^{O(1)}$. So in order to find a witness for $x \in L$ we just need to search through all strings y with $\mathrm{Kt}(y) \leq |x|^{O(1)}$. That can be done in exponential time.

($5 \Rightarrow 1$) Assume that there is a constant c, such that for every string x and every prefix y of x, $\mathrm{KDt}(y) \leq (\mathrm{KDt}(x) + \log|x|)^c$. Let x be a string of length n. If $\mathrm{KDt}(x) \geq (n/2)^c$ then the claim is true for x. Assume that $\mathrm{KDt}(x) < (n/2)^c$. Let $a = \max\{\mathrm{KDt}(y);\ y \text{ is a prefix of } x\}$. Clearly, $a \leq (\mathrm{KDt}(x) + \log|x|)^c < n$. We construct a sequence $S_a, \ldots, S_n$ of sets with $|S_i| \leq 2^a$, such that for each i, if $z \in \{0,1\}^i$ and z and every prefix y of z has $\mathrm{KDt}(y) \leq a$, then $z \in S_i$. We initially start with $S_a = \{0,1\}^a$ and then proceed iteratively as follows.

$$S_{i+1} := \{s \in S_i \circ \{0,1\} : \exists d_s \in \{0,1\}^a \; U(d_s, s) \text{ accepts , and if } s' \in S_i \circ \{0,1\} \text{ and } s \neq s' \text{ then } U(d_s, s') \text{ rejects } \}$$

It is fairly straightforward to verify that these sets have the property mentioned above, namely that they are not too big and that they contain all the simple strings having simple prefixes. Thus $x \in S_n$. Thus there is a machine M that in time $n2^{O(a)}$ on input (n, a, i) generates x, where i is the index of x in the set S_n. Hence, $\mathrm{Kt}(x) = |(n, a, i)| + \log n 2^{O(a)} + O(\log n) = O(a + \log n) = \mathrm{KDt}(x)^{O(1)}$. □

Since it seems unlikely that KDt is polynomially-related to Kt, one might ask if KDt is polynomially-related to KNt. Here again, we can use the techniques of [8] to show:

Corollary 18 *The following are equivalent:*

1. $\mathrm{Kt}(x) = \mathrm{KNt}(x)^{O(1)}$
2. $\mathrm{KDt}(x) = \mathrm{KNt}(x)^{O(1)}$

Proof. ($1 \Rightarrow 2$) This is trivial.

($2 \Rightarrow 1$) If $\mathrm{KDt}(x)$ is always polynomially bounded by $\mathrm{KNt}(x)$, then we know that for every x and y we have $\mathrm{KDt}(x) \leq \mathrm{KDt}(xy) + \log(|xy|)^{O(1)}$. Hence Theorem 17 yields $\mathrm{Kt}(x) = \mathrm{KDt}(x)^{O(1)}$, and we obtain the desired conclusion. □

Remark: The conditions of this theorem clearly imply the conditions of Theorem 17. They also imply that EXP/poly = NEXP/poly ∩ coNEXP/poly. (To see this, observe that $A \in$ NEXP/poly ∩ coNEXP/poly if and only if $\chi_A = n$ has polylogarithmic KNt complexity, and $A \in$ EXP/poly if and only if χ_A has polylogarithmic Kt complexity.) We do not know of an easy-to-state condition involving complexity classes that is equivalent to these statements.

4 Branching Program and Formula Size

The definition of KT complexity is motivated in large part by the fact that $\mathrm{KT}(x)$ is a good estimate of the circuit size required to compute the function f that has x as its truth table. But circuit size is only one of many possible interesting measures of the "complexity" of f. There is also great interest in knowing the size of the smallest branching programs computing f, as well as the size of the smallest Boolean formula representing f. Do these notions of complexity also give rise to a natural notion of Kolmogorov complexity?

In this section we answer this question by presenting two more notions of resource-bounded Kolmogorov complexity.

Definition 19 *Let U_1 be a deterministic Turing machine, and let U_2 be an alternating Turing machine.*

$$\begin{aligned}\mathrm{KB}_{U_1}(x) &= \min\{|d| + 2^s : \forall b \in \{0,1,*\} \\ &\qquad \forall i \le n+1\ U_1(d,i,b) \text{ runs in} \\ &\qquad \text{space } s \text{ and accepts iff } x_i = b\} \\ \mathrm{KF}_{U_2}(x) &= \min\{|d| + 2^t : \forall b \in \{0,1,*\} \\ &\qquad \forall i \le n+1\ U_2(d,i,b) \text{ runs in} \\ &\qquad \text{time } t \text{ and accepts iff } x_i = b\}\end{aligned}$$

As usual, we will choose a fixed "optimal" Turing machines U_1 and U_2 and use the notation KB and KF to refer to KB_{U_1} and KF_{U_2}. A deterministic Turing machine U_1 is KB-*optimal* if for any deterministic Turing machine U_1' there exists a constant $c \ge 0$ such that for all x, $\mathrm{KB}_{U_1}(x) \le (\mathrm{KB}_{U_1'}(x))^c$. Similarly, an alternating Turing machine U_2 is KF-*optimal* if for any alternating Turing machine U_2' there exists a constant $c > 0$ such that for all x, $\mathrm{KF}_{U_2}(x) \le (\mathrm{KF}_{U_2'}(x))^c$. The existence of optimal machines for KB and KF complexity follows via standard arguments. We get the following simple proposition.

Proposition 20 *For any string x of length 2^n representing the truth table of a function f, let* BPSIZE*(x) denote the size of the smallest branching program computing f, and let* FSIZE*(x) denote the size of the smallest Boolean formula representing f. Then* $(\mathrm{KB}(x) + \log|x|)^{O(1)} = (\mathrm{BPSIZE}(x) + \log|x|)^{O(1)}$*, and* $(\mathrm{KF}(x) + \log|x|)^{O(1)} = (\mathrm{FSIZE}(x) + \log|x|)^{O(1)}$.

For each of these two new measures, the sets of random strings R_{KB} and R_{KF} lie in coNP. Can we prove better upper bounds on their complexity? Can we prove any intractability results?

In [3] these questions were posed for the set R_{KT}, and Kabanets and Cai and posed similar questions earlier for the related Minimum Circuit Size Problem (MCSP) [17]. Although we are not able to reduce the factorization problem to R_{KB} and R_{KF} (as was accomplished for R_{KT} in [3]), we can come close.

In this section we prove that factoring Blum Integers can be done in $\mathrm{ZPP}^{R_{\mathrm{KF}}}$ and $\mathrm{ZPP}^{R_{\mathrm{KB}}}$. (For an oracle A, a function f is in ZPP^A if there exists a procedure computed by a probabilistic oracle machine with oracle A that on input x, on every halting path, produces $f(x)$, and the expected running time is polynomial.) We use results of [23] and [7] in order to accomplish this. We define the following computational problem.

Blum Integer Factorization: Given a Blum Integer $N \in \mathbb{N}$, find the primes P and Q such that $1 < P \le Q$ and $N = PQ$. (A $2n$-bit integer N is called a *Blum Integer* if $N = PQ$, where P and Q are two primes such that $P \equiv Q \equiv 3 \bmod 4$.)

Theorem 21 *Blum Integer Factorization is in* $\mathrm{ZPP}^{R_{\mathrm{KF}}} \cap \mathrm{ZPP}^{R_{\mathrm{KB}}}$*, i.e., there are* $\mathrm{ZPP}^{R_{\mathrm{KF}}}$ *and* $\mathrm{ZPP}^{R_{\mathrm{KB}}}$ *procedures that on input N that is a Blum Integer produce factors P and Q of N.*

Proof. In [23], they construct a pseudo-random function ensemble $\{f_{N,r}(x) : \{0,1\}^n \to \{0,1\}\}_{N,r}$ with the following two properties (Construction 5.2 and Corollary 5.6 of [23]):

1. There is a TC^0 circuit computing $f_{N,r}(x)$, given $2n$-bit integer N, $4n^2 + 2n$-bit string r and n-bit string x.
2. For every probabilistic oracle Turing machine $\mathcal{M}$, that on its $2n$-bit input asks queries of length only n, and any constant $\alpha > 0$, there is a probabilistic Turing machine $\mathcal{A}$, such that for any $2n$-bit Blum Integer $N = PQ$, if
$$|\Pr[\mathcal{M}^{f_{N,r}}(N) = 1] - \Pr[\mathcal{M}^{R_n}(N) = 1]| > 1/n^\alpha$$
where $R_n = \{g : \{0,1\}^n \to \{0,1\}\}_n$ is a uniformly distributed random function ensemble and the probability is taken over the random string r and the random bits of $\mathcal{M}$, then $\Pr[\mathcal{A}(N) \in \{P,Q\}] > 1/n$.

Their factoring construction relativizes, i.e., the properties of $\{f_{N,r}(x)\}_{N,r}$ hold even if $\mathcal{M}$ and $\mathcal{A}$ have an access to the same auxiliary oracle.

Let $f_{N,r}(x)$ be computable by a TC^0 circuit of size $n^{c'}$, and hence, by an NC^1 circuit of size $n^{c''}$, for some constants $c', c'' > 1$. Let $x_1, x_2, \ldots, x_{2^n}$ denote strings in $\{0,1\}^n$ under lexicographical ordering. Clearly, there is a constant $c > 1$, such that for all large enough n, all $2n$-bit integers N and all $4n^2 + 2n$-bit strings r, the string obtained by concatenating $f_{N,r}(x_1), f_{N,r}(x_2), \ldots, f_{N,r}(x_{n^c})$ has KF-complexity less than $n^c/2$. Fix such a c and consider the following oracle Turing machine $\mathcal{M}$ with oracles R_{KF} and a function g:

- On $2n$-bit input N, $\mathcal{M}$ asks oracle g queries $x_1, x_2, \ldots, x_{n^c}$ to get answers $y_1, y_2, \ldots, y_{n^c}$. Then, $\mathcal{M}$ accepts if $y_1 y_2 \cdots y_{n^c} \in R_{\mathrm{KF}}$ and rejects otherwise.

It is easy to see that if $g \in \{f_{N,r}(x)\}_{N,r}$ then M always rejects, for n large enough. On the other hand, if g is taken uniformly at random from R_n, then $y_1 y_2 \cdots y_{n^c}$ is a random string and the probability that M accepts is at least $1 - 2^{-n/2}$. Hence, $|\Pr[\mathcal{M}^{f_{N,r}(x)}(N) = 1] - \Pr[\mathcal{M}^{R_n}(N) = 1]| > 1/2$, for n large enough. By the properties of $f_{N,r}(x)$ we can conclude that there is a probabilistic Turing machine $\mathcal{A}$ with oracle R_{KF} that factors N with non-negligible probability. We can reduce the error to zero by verifying the output of $\mathcal{A}$.

Since any function that is computable by NC^1 circuits is computable by polynomial size branching programs, by considering branching programs instead of NC^1 circuits we get that Blum Integer Factorization is in $\mathrm{ZPP}^{R_{\mathrm{KB}}}$. □

4.1 Hardness of Approximation

Many computational problems that complexity theory studies are decision problems for which an answer is always either "yes" or "no". Other problems that are of interest in computational complexity are optimization problems. Examples of optimization problems are the Maximum Clique — what is the size of the largest clique in G — and the Minimum Circuit Size Problem — what is the size of the smallest circuit computing a Boolean function f given by its truth table?

For some of these optimization problems efficient (polynomial time) algorithms are known. For others, no efficient algorithm is known. Moreover, it is known that these optimization problems are hard for NP. Given that the exact solution of such an optimization problem may be hard to find one can try to find at least an approximation to the solution. Many optimization problems are known for which even finding an approximation cannot be done efficiently, unless something unlikely is true, such as $\mathrm{P} = \mathrm{NP}$. For example, [13] shows that the Maximum Clique cannot be approximated up to factor $n^{1-\epsilon}$ in polynomial time, unless $\mathrm{P} = \mathrm{NP}$.

In this section we study the following optimization problems — given a truth table of a function f, what is the smallest size of a circuit, a branching program or a formula, respectively, that computes f. We show that under certain plausible complexity assumptions these optimization problems are hard to approximate.

A related problem was already studied by [10]. In his Master's Thesis he gives for some $\epsilon > 0$ an n^ϵ non-approximability result for a variant of the Minimum DNF Formula problem, under the assumption that $\mathrm{P} \neq \mathrm{NP}$. His result builds upon probabilistically checkable proofs. We obtain our results using completely different technique. Tools for our non-approximability results are related to hardness results for the sets R_{KT}, R_{KB} and R_{KF}.

For a minimization problem $f : \Sigma^* \to \mathbb{N}$ we say that $g : \Sigma^* \to \mathbb{N}$ *approximates f up to factor $r : \mathbb{N} \to \mathbb{N}$* if for all $x \in \Sigma^*$, $1 \leq g(x)/f(x) \leq r(|x|)$. For a complexity class $\mathcal{C}$ we say that *f cannot be approximated up to factor r in $\mathcal{C}$* if no $g \in \mathcal{C}$ approximates f up to factor r.

We recall definitions of two more problems that are believed to be computationally difficult.

Integer Factorization: Given a composite integer $N \in \mathbb{N}$, find two integers P and Q such that $1 < P \leq Q$ and $N = PQ$.

Discrete Logarithm: Given three integers x, z, N, $1 \leq x, z < N$, find an i such that $x = z^i \bmod N$ if such i exists.

The following result is implicit in [3]:

Theorem 22 *Let $0 < \gamma < 1$ be a constant and B be a set of at least polynomial density such that for any $x \in B$, $\mathrm{SIZE}(x) > |x|^\gamma$. Then Integer Factorization and Discrete Logarithm are in BPP^B.*

This theorem implies the non-approximability of circuit size.

Theorem 23 *For any $0 < \epsilon < 1$, $\mathrm{SIZE}(x)$ cannot be approximated up to factor $n^{1-\epsilon}$ in* BPP*, unless Integer Factorization and Discrete Logarithm is in* BPP.

Proof. Assume that for some $0 < \epsilon < 1$, there is a function $g \in \mathrm{BPP}$ that approximates $\mathrm{SIZE}(x)$ up to factor $n^{1-\epsilon}$. We will show that this implies that Integer Factorization and Discrete Logarithm are in BPP.

Consider the set $B = \{x \in \{0,1\}^*;\ g(x) > |x|^{1-\epsilon/2}\}$. Clearly, $B \in$ BPP. Since for all $x \in \{0,1\}^*$, $1 \leq g(x)/\text{SIZE}(x) \leq n^{1-\epsilon}$, we have that for all $x \in B$, $\text{SIZE}(x) > |x|^{\epsilon/2}$ and also for all $x \in \{0,1\}^*$, if $\text{SIZE}(x) \geq |x|^{1-\epsilon/2}$ then $x \in B$. By [20], almost all truth tables $x \in \{0,1\}^*$ require circuits of size at least $O(n/\log n)$. Hence, B is of at least polynomial density. By Theorem 22, Integer Factorization and Discrete Logarithm are in $\text{BPP}^{\text{BPP}} \subseteq$ BPP. (In the case of Integer Factorization we can actually verify correctness of the result to get ZPP computation instead of BPP.) □

Similar non-approximability results can be obtained for formula and branching program sizes. A proof similar to the proof of Theorem 21 yields the following claim.

Theorem 24 *Let $0 < \gamma < 1$ be a constant and B be a set of at least polynomial density such that for any $x \in B$, $\text{BPSIZE}(x) > |x|^{\gamma}$ or for any $x \in B$, $\text{FSIZE}(x) > |x|^{\gamma}$. Then there is a ZPP^B procedure that on input N that is a Blum Integer produces factors P and Q of N.*

As a corollary to this theorem we obtain:

Theorem 25 *For any $0 < \epsilon < 1$, $\text{BPSIZE}(x)$ and $\text{FSIZE}(x)$ cannot be approximated up to factor $n^{1-\epsilon}$ in* BPP, *unless Blum Integer Factorization is in* ZPP.

In Theorems 23 and 25, a function f is computable in BPP if there is a polynomial time probabilistic machine M such that for any x, $\Pr[M(x) = f(x)] \geq 2/3$. However, the results hold for an even stronger notion of non-approximability: For any $0 < \epsilon < 1$, if there is a polynomial time probabilistic machine M such that for all x, $\Pr[1 \leq M(x)/\text{BPSIZE}(x) \leq n^{1-\epsilon}] \geq 2/3$ or $\Pr[1 \leq M(x)/\text{FSIZE}(x) \leq n^{1-\epsilon}] \geq 2/3$ then Blum Integer Factorization is in ZPP. Similarly, if there is a polynomial time probabilistic machine M such that for all x, $\Pr[1 \leq M(x)/\text{SIZE}(x) \leq n^{1-\epsilon}] \geq 2/3$ then Integer Factorization and Discrete Logarithm are in BPP. These results follow by exactly the same proofs as the weaker one where one has to observe that the derandomization results that we use hold not only relative to oracles that distinguish between random and pseudorandom strings but also relative to probabilistic procedures that distinguish between random and pseudorandom strings with non-negligible probability.

4.2 KF Complexity and the NEXP $\subseteq$ NC^1 Question

Derandomization techniques were used in [15] to show that NEXP $\subseteq$ P/poly if and only if NEXP = MA; it was observed in [2] that this is also equivalent to conditions concerning the Kt-complexity of sets in P. In this section we conduct a similar investigation of the question of whether or not NEXP is contained in non-uniform NC^1.

Before we state the main result of this section, it will be helpful to present a technical definition. We begin by recalling the definition of IP[P/poly].

Definition 26 *[5]* IP[P/poly] *is the class of languages having an interactive proof system where the strategy of the prover can be computed by a polynomial sized circuit (also see [4] where the multiple prover class* MIP[P/poly] *is observed to be the same as* IP[P/poly]*).*

Clearly IP[P/poly] $\subseteq$ MA $\cap$ P/poly (because Merlin can guess the circuit that implements the Prover's strategy and send send it to Arthur); it appears to be a proper subclass of MA (since otherwise NP $\subseteq$ P/poly). If NEXP $\subseteq$ P/poly, the proof of [15] actually shows that NEXP = IP[P/poly]. We now define an analogous subclass of MA$\cap$ non-uniform NC^1.

Definition 27 MIPNC^1 *refers to the class of languages for which there is a 2-prover one-round interactive proof protocol where the strategy of each prover can be implemented by a (non-uniform)* NC^1 *circuit family and the computation of the verifier is computable by a uniform (probabilistic)* NC^1 *circuit family. (Although it is important that the verifier's circuits be uniform, our results do not depend crucially on the exact notion of uniformity. They hold for* P*-uniformity and for* DLOGTIME*-uniformity.)*

We could likewise define IPNC^1 as the class of languages similar to the above for a single-prover constant-round interactive proof protocol, but we can easily see that MIPNC^1 and IPNC^1 coincide.

Theorem 28 *The following are equivalent:*

1. *For all $A \in$* NP, $\text{KF}_A(n) = \log^{O(1)} n$.
2. *For all $A \in$* DLOGTIME*-uniform* AC^0, $\text{KF}_A(n) = \log^{O(1)} n$.
3. *All* NEXP *search problems are solvable in* NC^1.
4. NEXP $\subseteq$ *non-uniform* NC^1.
5. NEXP = MIPNC^1.

Proof. Items (1) and (2) are easily seen to be equivalent, as in the remark at the end of Section 3. That is, (1) trivially implies (2), and if the set of (string,witness) pairs in AC^0 for an NP language A has low KF-complexity, then so does

the set of strings in A. Obviously both of these conditions are equivalent to the corresponding condition for languages in P.

The proof that (2 $\Rightarrow$ 3) is immediate, once the following two assertions are established:

- (2) $\Rightarrow$ EXP $\subseteq$ NC^1.
- (2) $\Rightarrow$ NEXP search problems are solvable in EXP.

(Assume both of these assertions hold. Then for a given NEXP search problem solved in exponential time by machine M, the language $\{(x, i, b) :$ the ith bit output by M on input x is $b\}$ is in NC^1. The existence of such circuit families for NEXP search problems is precisely what is meant by condition (3).) Let us examine each assertion in turn.

Let $A \in$ EXP. Let $B = \{x : x$ is a prefix of $\chi_A\}$. B is clearly in P and (since we have already observed that (2 $\Rightarrow$ 1)) our assumption tells us that $\mathrm{KF}_B n = \log^{O(1)}(n)$. Now Proposition 20 allows us to conclude that $A \in$ NC^1.

For the second assertion, let M be any NEXP machine, and consider the language $C = \{y10^m : \text{where } y \in \{0,1\}^{2^{|m|^k}}$ is a witness that M accepts $m\}$. C is in DLOGTIME-uniform AC^0 and by (2) if there is any string in $C^{=n}$ then there is a string in $C^{=n}$ with small KF complexity. The exponential-time algorithm solving this search problem involves taking input m and searching through all short descriptions and seeing if any of the strings thus described encodes an accepting computation path of M on input m.

The implication (3 $\Rightarrow$ 4) requires proof. Certainly (3) implies that NP search problems are solvable in NC^1. Let $A \in$ NP be accepted by NP-machine M, and let C be a circuit solving the search problem defined by M. Thus $x \in A$ if and only if $C(x,1)C(x,2)\cdots C(x,n^k)$ encodes an accepting computation of M. This latter condition can also be checked in NC^1, which implies NP $\subseteq$ (non-uniform) NC^1. NP being contained in NC^1 easily implies that Σ_2^p is contained in NC^1. On the other hand, by [15], if NEXP search problems are solvable in P/poly, then NEXP is in Σ_2^p.

To prove that (4 $\Rightarrow$ 5), observe that by [15] if NEXP $\subseteq$ P/poly then NEXP = MA = PSPACE. By [9], we know that PSPACE has 2-prover, 1-round interactive proof systems, where the honest provers are in PSPACE. Also we note that the verifier's protocol is very easy to compute; it sends random sequences to each prover and receives from the provers sequences of polynomials on which it performs (in parallel) some consistency checks. The consistency checks involve field operations, which are computable by DLOGTIME-uniform TC^0 circuits [14]. All the queries to the provers are made in one round (and hence are non-adaptive). Since by assumption, PSPACE $\subseteq$ NC^1, we have that every language in NEXP is also in MIPNC^1.

Now we prove the implication (5 $\Rightarrow$ 2). We largely follow [15], where it is shown that if NEXP $\subset$ P/poly, then NEXP-search can be performed by P/poly circuits. More precisely, we will show that if there is a set in P with large KF-complexity, then for every $\epsilon > 0$, MIPNC^1 $\subset$ $io-$[NTime$(2^{n^\epsilon})/n^\epsilon$]. As in [15] this latter condition implies either that MIPNC^1 is a proper subset of NEXP (which is to say that condition (5) is false) or else EXP $\neq$ NEXP (which also easily implies that condition (5) is false).

Let $A \in$ MIPNC^1, where the verifier's strategy is computable by a P-uniform family of probabilistic NC^1 circuits $\{C_n\}$. Let p be a polynomial, such that C_n uses at most $p(n)$ probabilistic bits. Our strategy to determine if $x \in A$ is

1. Construct the circuit $C = C_{|x|}$.
2. Nondeterministically guess NC^1 circuits D, D' that might implement the strategies of the provers in the MIPNC^1 protocol for A.
3. Construct a circuit B that, given an input y of length $p(n)$
   (a) Uses C to compute the query that gets posed to each prover in the MIPNC^1 protocol for A on input x and probabilistic sequence y.
   (b) Uses D and D' to answer the queries.
   (c) Uses C to compute the actions of the verifier.
4. Estimate the probability that B accepts a randomly-chosen string y.

By the definition of MIPNC^1, if $x \in A$ then there are fan-in two circuits D and D' implementing the strategy of the provers (where the depth of D and D' is bounded by $d \log n$ for some constant d depending only on A) such that the circuit B accepts *all* of the inputs y, whereas if $x \notin A$, then *no* provers (and hence also no provers computed by small circuits D and D') can cause B to accept more than one-third of the inputs y.

All of the steps in this algorithm are easily computable in NP except for the final step 4. In order to complete the argument that MIPNC^1 $\subset$ $io-$[NTime$(2^{n^\epsilon})/n^\epsilon$], it suffices to show that for infinitely many input lengths n, there is an advice string of length n^ϵ such that a nondeterministic machine running in time 2^{n^ϵ} can estimate the probability that a circuit with fan-in two and depth $b \log p(n)$ accepts a randomly-chosen input of length $p(n)$ (where the constant b and the polynomial p depend only on our language A, and do not depend on ϵ).

As in [3], we will make use of the hardness-versus-randomness techniques of [24, 6]. In particular, some of the results of [24, 6, 18] are summarized in [3] in the following form.

Definition 29 *For all large n, any $\epsilon > 0$ and any Boolean function $f : \{0,1\}^{n^{\epsilon/3}} \to \{0,1\}$ there is a pseudorandom generator $G^{\mathrm{BFNW}}_{f,\epsilon} : \{0,1\}^{n^{\epsilon}} \mapsto \{0,1\}^{p(n)}$ with the property that the function $G^{\mathrm{BFNW}}_{f,\epsilon}$ is computable in space $O(n^{\epsilon})$ given access to the Boolean function f, and such that the following theorem holds.*

Theorem 30 ([6, 18]) *There is a constant k' depending on ϵ such that if T is a set such that $|\Pr_{r \in U_{p(n)}}[r \in T] - \Pr_{x \in U_{n^{\epsilon}}}[G^{\mathrm{BFNW}}_{f,\epsilon}(x) \in T]| \geq 1/3$, then there exists an oracle circuit C of size $n^{k'}$ with oracle T that computes f and queries T non-adaptively.*

Closer examination of the proof techniques that are used in [6, 18] shows that the circuit C computing the reduction can actually be implemented as a *constant depth* circuit of MAJORITY gates and oracle gates. Thus it can be implemented as a circuit of depth $k \log n$ for some constant k, consisting of oracle gates (where there is no path in the circuit from one oracle gate to another) and AND and OR gates of fan-in two.

Now we can state our $io - [\mathrm{NTime}(2^{n^{\epsilon}})/n^{\epsilon}]$ algorithm to estimate the probability that an NC^1 circuit accepts. Let L be a language in $\mathrm{DTime}(n^k)$ such that for every ℓ there exist infinitely many m such that $\mathrm{KF}_L(m) > \log^{\ell} m$. By our assumption that condition (2) fails, such a set L exists.

On input x of length n, our advice string will be a number m with approximately n^{δ} bits with $\delta = \epsilon/3$, such that L contains strings of length m, and all strings of length m in L have high KF complexity. Our nondeterministic algorithm will guess a string z of length m and verify that $z \in L$. This takes time $2^{O(n^{\epsilon})}$. Let f be the Boolean function on inputs of length $\lceil \log m \rceil$ (roughly n^{ϵ}) whose truth table has z as a prefix (and is zero elsewhere). By our assumption on L (combined with Proposition 20), there exist infinitely many m such that function f requires Boolean formulae of size greater than $p(n)^{k+b}$. For any input length n for which a corresponding $m = 2^{O(n^{\epsilon})}$ exists, the probability that circuit B accepts can be estimated by counting the fraction of strings y of length n^{ϵ} such that B accepts $G^{\mathrm{BFNW}}_{f,\epsilon}(y)$. This fraction must be within one-third of the true probability (since otherwise f is computed by a formula of size $p(n)^{k+b}$, by Theorem 30).

Since $G^{\mathrm{BFNW}}_{f,\epsilon}(y)$ is computable in space n^{ϵ}, the entire computation to estimate the acceptance probability of the NC^1 circuit B (and to recognize language A) is $2^{O(n^{\epsilon})}$.

This completes the proof. □

The following definition of MIPL combined with an analogous proof yields Theorem 32

Definition 31 MIPL *corresponds to the class of languages for which there is a 2-prover one-round interactive proof protocol where the strategy of each prover can be implemented in* L/poly *and the verifier is in* L.

Theorem 32 *The following are equivalent :*

1. NEXP $\subseteq$ L/poly
2. *All* NEXP *search problems are solvable in* L/poly
3. *For all* $A \in$ P $\mathrm{KB}_A(n) = \log^{O(1)} n$.
4. NEXP = MIPL

5 Concluding Comments

One could consider placing more restrictions on the universal alternating machine in the definition for KB complexity, for instance by restricting the number of alternations, or by making it deterministic. At first glance, it seems that one might obtain a measure that is related to depth k AC^0 circuit size for fixed k – but it seems that such machines cannot do much interesting computation on input (d, i, b) without looking at all of i, which means that their running time is so high that the framework developed here does not yield a very interesting measure. Is there a useful definition that can be developed to capture this notion?

For the more "limited" notions of Kolmogorov complexity KB and KF, we are not able to prove as strong intractability results as were proved for KT in [3]. However, it is not clear that this needs to be the case. For instance, although it is not known if the minimum circuit size problem is NP-complete, it *is* complete when restricted to DNF circuits [10, 21]. Is there a natural, restricted notion of Kolmogorov complexity, for which the "random" strings do indeed provide a complete set for coNP?

References

[1] E. Allender. Some consequences of the existence of pseudorandom generators. *Journal of Computer and System Sciences*, 39:101–124, 1989.

[2] E. Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *Proc. Conf. on Found. of Software Technology and Theo. Comp. Sci. (FST&TCS)*, volume 2245 of *Lecture Notes in Computer Science*, pages 1–15, 2001.

[3] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 669–678, 2002.

[4] V. Arvind and J. Kobler. Graph isomorphism is low for $\mathrm{ZPP}^{\mathrm{NP}}$ and other lowness results. Technical Report TR99-033, Electronic Colloquium on Computational Complexity, 1999.

[5] V. Arvind, J. Koebler, and R. Schuler. On helping and interactive proof systems. *International Journal of Foundations of Computer Science (IJFCS)*, 6(2):137–153, 1995.

[6] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[7] E. Biham, D. Boneh, and O. Reingold. Breaking generalized Diffie-Hellmann modulo a composite is no easier than factoring. In *Information Processing Letters 70(2)*, pages 83–87, 1999.

[8] H. Buhrman, L. Fortnow, and S. Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM Journal on Computing*, 31(3):887–905, 2002.

[9] J. Cai, A. Condon, and R. J. Lipton. PSPACE is provable by two provers in one round. *Journal of Computer and System Sciences*, 48:183–193, 1994.

[10] S. Czort. The complexity of minimizing disjunctive normal form fomulas. Master's thesis, University of Aarhus, 1999.

[11] L. Fortnow. Comparing notions of full derandomization. In *Proc. IEEE Conf. on Computational Complexity '01*, pages 28–34, 2001.

[12] L. Fortnow and M. Kummer. On resource-bounded instance complexity. *Theoretical Computer Science*, 161(1–2):123–140, 1996.

[13] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[14] W. Hesse, E. Allender, and D. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[15] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. In *Proc. IEEE Conf. on Computational Complexity*, pages 2–12, 2001.

[16] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 181–190, 1999.

[17] V. Kabanets and J.-Y. Cai. Circuit minimization problem. In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 73–79, 2000.

[18] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 2002. To appear; a preliminary version appeared in STOC '99.

[19] L. A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[20] O. B. Lupanov. A method of circuit synthesis. *Izvestiya VUZ, Radiofizika*, 1(1):120–140, 1959.

[21] W. Masek. Some NP-complete set covering problems. Unpublished manuscript, 1979.

[22] P. B. Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 71–80, 1999.

[23] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudorandom functions. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 458–467, 1997.

[24] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[25] S. Rudich. Super-bits, demi-bits, and $\tilde{\mathrm{NP}}$/qpoly-natural proofs. In *Proceedings of RANDOM*, volume 1269 of *Lecture Notes in Computer Science*, 1997.

[26] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proc. IEEE Symp. on Found. of Comp. Sci. (FOCS)*, pages 648–657, 2001.