The Computational Complexity Column

BY

Michal Koucký

Computer Science Institute, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
koucky@iuuk.mff.cuni.cz
https://iuuk.mff.cuni.cz/~koucky/

Range Avoidance and the Complexity of Explicit Constructions

Oliver Korten
Department of Computer Science
Columbia University, New York City
oliver.korten@columbia.edu

Abstract

A recent line of work has investigated the complexity of explicit construction problems through the study of a search problem known as *Range Avoidance*: given as input a boolean circuit $C : \{0,1\}^n \to \{0,1\}^{n+1}$, find an element $y \in \{0,1\}^{n+1}$ outside of its range. Analysis of this search problem and its variants has lead to several exciting new results in derandomization and circuit complexity. In this survey we give an overview of this nascent research direction and its connections to some old and fundamental questions in complexity theory.

1 Introduction

1.1 Motivation: Explicit Constructions

Throughout combinatorics and computer science, a fundamental tool used to demonstrate the existence of interesting combinatorial objects is the *probabilistic method*. Rather than constructing an explicit example of the desired object, a random distribution of objects is chosen, and the probability of the desired object arising from the distribution is shown to be strictly positive. The first appearance of this argument was in a classical paper of Erdős who used it to establish the existence of so-called *Ramsey graphs*: n vertex undirected graphs whose largest clique and largest independent set each have size at most $2 \log n$ [15]. A few years later, an essentially identical argument was used by Shannon [42] to demonstrate the existence of functions $f : \{0,1\}^n \to \{0,1\}$ requiring boolean circuits of size $\geq 2^n/n$. In each case, the probabilistic argument gives no indication as to a *particular example* of the object in question, and a signficant body of research over the past eight decades has been devoted to matching these nonconstructive bounds

with explicit arguments. Similar phenomenon have become ubiquitous throughout combinatorics and computer science, whereby a nonconstructive argument proving the existence of an interesting class of object has spawned a deep line of work dedicated to producing a concrete example of one such object. In some fortunate cases, such as Ramsey graphs, error correcting codes, and expanders, decades of hard work eventually lead to optimal or near-optimal solutions. In many other cases no significant progress has been made to date.

The pursuit of *explicit constructions* in place of nonconstructive arguments has several motivations, which may be divided into roughly two classes: *informal/philosophical* and *formal/computational*. The informal motivation can be summarized by a maxim asserting that we do not fully understand a concept until we can produce an example. When we have some combinatorial property, the mere existence of objects satisfying that property does not give us any deeper information about *why* a certain object achieves the property. If we succeed in the quest of finding an explicit object, and have a mathematical proof that this object *in particular* has the relevant property, this proof will usually contain within it a furtherance of our general understanding of the concept in question.

This informal motivation utilizes a rather vague definition of the term "explicit." In this framework, we would call an object, such as a Ramsey graph $G \subseteq \binom{[n]}{2}$, "explicit" provided we can present it by a comprehensible definition from which we can extract and compute various other properties of the object by direct mathematical argument. The class of *formal/computational* motivations for studying explicit constructions centers itself on a more concrete definition of the notion of "explicitness:" in this context we will say that an object is explicit if it can be produced by an efficient algorithm. In the example of a Ramsey graph $G \subseteq \binom{[n]}{2}$, we might say that it is explicit if there is a $\mathrm{poly}(n)$ time algorithm which outputs its adjacency matrix given n; embedded in this kind of definition is the necessity of reserving the notion of explicitness for *sequences of objects* rather than individuals.

The motivations for the study of *computationally explicit* constructions can be further subdivided into two categories, which turn out to be intimately connected: *derandomization* and *computational lower bounds*. Derandomization is a field dedicated to the study of when randomized algorithms can be replaced by efficient deterministic algorithms posessing similar behavior. At a high level, the connection to explicit constructions may be viewed as follows. Say we have a randomized algorithm $\mathcal{A}$ that flips n random coins, which we hope to simulate deterministically. We may think of $\mathcal{A}(r)$ as a deterministic algorithm which takes an n-bit string r, and has some desirable behavior with high probability when r is chosen uniformly. Peering into the proof of $\mathcal{A}$'s correctness, it is often possible to isolate a *particular pseudorandom property* of n-bit strings, call it $\Pi \subseteq \{0,1\}^n$, so that a random string r has the property Π with high probability, and whenever $r \in \Pi$, $\mathcal{A}(r)$ has the correct behavior. Hence, if we could somehow *deterministically* construct a fixed string

$r \in \{0,1\}^n$ which has the necessary property Π, then we could reap the benefits of our randomized algorithm while using this fixed string in place of true randomness. The problem of constructing strings with property Π is then the relevant explicit construction problem.

In the case of computational lower bounds, the connection to explicit constructions is most direct. A fundamental problem in complexity theory is to show separations of the form $\mathcal{A} \not\subseteq C$ where $\mathcal{A}$ is a uniform complexity class such as $\mathsf{NP}, \mathsf{PSPACE}, \mathsf{EXP}$, and C is a *nonuniform* class of circuits, such as $\mathsf{AC}^0, \mathsf{TC}^0, \mathsf{P/poly}$. By a counting argument, it is usually straightforward to show that there exists boolean functions $f : \{0,1\}^n \to \{0,1\}$ which are hard for the class C. We may then study the explicit construction problem of producing an example of such a function f; if the construction itself has bounded uniform complexity in some class $\mathcal{A}$, then this implies by definition that $\mathcal{A} \not\subseteq C$. While this connection may sound rather tautological, the study of computational lower bounds through the lens of explicit construction has proved quite useful, particularly in the case that the uniform classes $\mathcal{A}$ is at least as large as EXP.

1.2 The "Right" Complexity Class for Explicit Constructions

The main topic of this survey is the Range Avoidance problem: given a boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ with $m > n$, find a string $y \in \{0,1\}^{n+1}$ outside the range of C. The primary motivation for studying Range Avoidance stems from its ability to capture the computational complexity of the kinds of explicit construction problems discussed in the last section [29]. This connection is based on the simple (and well-known) observation that most probabilistic existence arguments can be strengthened to *encoding arguments*: to show that most objects have a desired property, we show how any "bad object" failing to satisfy the property can be coded by description of nontrivial length. In an information theoretic sense this claim has no content: obviously every set of size 2^k can have its elements coded by k-bit strings. However, we shall see that if an encoding can be defined for which the *decoder* has an efficient algorithm, this will yield a reduction to Range Avoidance.

It is easiest to work through an example, and perhaps the simplest is the case of Ramsey graphs mentioned above. We start with the standard probabilistic argument which shows the existence of a k-Ramsey graph on n vertices (no clique or independent set of size $\geq k$) whenever $k >> \log n$. Choose $G \sim \{0,1\}^{\binom{n}{2}}$ uniformly at random. For each set $V \subseteq [n]$ of size k, the probability that $G_{u,v} = 1$ for all $u, v \in V$ is $2^{-\binom{k}{2}}$; a symmetric argument gives the same bound in the case $G_{u,v} = 0$. Now, taking a union bound over all choices of V and the choice of whether V is a clique or an independent set, the probability any clique or independent set

exists is at most

$$2 \cdot \binom{n}{k} \cdot 2^{-\binom{k}{2}} \leq 2^{1+k\log n-\binom{k}{2}}$$

Hence setting $k \geq (2 + o(1)) \cdot \log n$, this probability is strictly below 1 for n sufficiently large.

Now, say that we had a graph G and a clique or independent set V of size k in G. Using the exact same analysis, we can form a concise encoding of G as follows:

1. 1 bit indicating whether V is a clique or independent set.
2. $k \log n$ bits describing the set V as a list of vertices.
3. $\binom{n}{2} - \binom{k}{2}$ bits describing G on all edges $\{u, v\}$ with $\{u, v\} \nsubseteq V$.

It is clear that from this description we could uniquely decode the graph G in polynomial time. The total number of bits is $1 + k \log n + \binom{n}{2} - \binom{k}{2}$, which is strictly less than $\binom{n}{2}$ provided $k \geq (2 + o(1)) \log n$. Hence if we define

$$C : \{0,1\}^{\binom{n}{2}+1+k\log n-\binom{k}{2}} \to \{0,1\}^{\binom{n}{2}}$$

as the function which takes such a concise graph description and outputs the graph G being described, we see that any graph which fails to be k-Ramsey will lie in the range of C; hence any $G \notin \text{range}(C)$ is a solution to our explicit construction problem. Since C is computable in $\text{poly}(n)$ time, we can efficiently construct a boolean circuit computing C in time $\text{poly}(n)$ given n. Since the number of input bits of C is strictly less then the number of outputs, C is a valid instance of the Avoid problem and we have succeeded in reducing our explicit construction problem to Range Avoidance.

1.3 Road Map

Since its introduction in [28] and its investigation in connection to explicit construction problems in [29], Range Avoidance has received considerable attention in the past few years [9–11, 18, 21, 23, 30, 31, 35, 41, 43]. At a very high level, some of the key takeaways from this line of work are:

1. Essentially all interesting and unsolved explicit construction problems found throughout theoretical computer science can be reduced in polynomial time to Avoid using a small toolkit of reduction techniques.
2. Several important explicit construction problems actually reduce to a special case of Avoid in which the input circuit $C : \{0,1\}^n \to \{0,1\}^m$ has a special

form. In other cases, nontrivial algorithms for special cases of Avoid have been found unconditionally. Unfortunately, the known results of this form do not yet "match up" to generate new explicit constructions.

3. There are nontrivial conditional and unconditional upper bounds for Avoid inside of the polynomial hierarchy, one of which has lead to a recent breakthrough circuit lower bound.

This survey will attempt to cover the current state of knowledge surrounding this intriguing search problem, and some of its broader implications in complexity theory. The content of this article is laid out as follows. In Section 2 we describe the basic formulation of explicit construction problems, the connection between explicit constructions and circuit lower bounds, and make some preliminary observations about the complexity of Avoid. In Section 3 we survey the main high level techniques used to reduce various explicit construction problems to Avoid. In Section 4 we discuss reductions between different variants of the Avoid problem. In Section 5 we discuss some more involved *upper bounds* for Range Avoidance, some of which have important implications in circuit complexity. In Section 6 we cover some results indicating the *hardness* of Avoid, both in the white-box model (cryptographic hardness) and in the black box model (oracle separations). In Section 7 we discuss work on an easier variant of Avoid lying in TFNP called *Lossy Code*. Finally in Section 8 we present some important open problems.

1.4 Background– Bounded Arithmetic

The investigations discussed in this survey are deeply connected to, and in some cases directly inspired by, important work in the field of *Bounded Arithmetic*. Bounded Arithmetic studies the strength of subtheories of Peano Arithmetic whose induction principle is restricted to formulas of bounded computational complexity; for formal definitions and a comprehensive introduction to the area see [32]. An important early question in the field was whether the theory $I\Delta_0$ could prove the existence of infinitely many primes; in complexity-theoretic terminology, we may think of $I\Delta_0$ as the fragment of Peano Arithmetic with induction for formulas decidable in the linear time hierarchy:

$$\mathsf{LH} = \bigcup_{i,c\in\mathbb{N}} \Sigma_i\text{-}\mathsf{Time}[c \cdot n]$$

To resolve this question, [39] took the following approach: first isolate an abstract combinatorial principle which suffices to prove the existence of infinitely many primes in $I\Delta_0$, then give an $I\Delta_0$ proof of this principle[1]. The combinatorial principle

[1] In fact they could only prove this principle in a mild extension of $I\Delta_0$ now known as T_2.

used in [39] was the *weak pigeonhole principle*, which states that no formula in the language of $I\Delta_0$ can define an injective map $2^{n+1} \mapsto 2^n$; in a theory as strong as $I\Delta_0$ this is equivalent to the statement that no formula defines a surjective map $2^n \mapsto 2^{n+1}$. This work was influential in at least two respects. First, the technique used to prove the weak pigeonhole principle in $I\Delta_0$ based on repeated composition of a supposed surjection $2^n \mapsto 2^{n+1}$ has been applied repeatedly in later work, as discussed in Section 5.1. Second, the work of [39] highlighted the importance of the weak pigeonhole principle as a key lemma from which highly nontrivial results in number theory and combinatorics could be derived via the use of counting arguments.

This latter direction was taken to its logical extreme in the highly influential thesis of Jeřábek [25]. In this work, Jeřábek showed that a theory of bounded arithmetic possessing induction for polynomial time computable predicates together with the *dual weak pigeonhole principle for polynomial time functions* could formalize a wide array of difficult complexity theoretic arguments that utilize randomness and probability. This variant of the pigeonhole principle essentially asserts that the problem Avoid is a *total search problem*: any instance $C : \{0,1\}^n \to \{0,1\}^{n+1}$ of Avoid defined by a boolean circuit must have a solution. In particular, Jeřábek was able to show this theory could define and analyze many key properties of *randomized algorithms*. Jeřábek also showed that the dual weak pigeonhole principle defining his theory could be replaced by a principle asserting the existence of boolean functions $f : \{0,1\}^n \to \{0,1\}$ of high circuit complexity; this result plays an important role in the study of Avoid, and was used in [29] to characterize the complexity of Avoid in terms of a computational hardness assumption; this is discussed further in Section 5.2.

2 Preliminaries

In this section we introduce basic definitions related to explicit construction problems and Avoid.

2.1 Formalizing Explicit Construction

We start with a formal definition of an explicit construction problem:

Definition 1. *An explicit construction problem is defined by a language* $\Pi \subseteq \{0,1\}^*$. *We use* Π_n *to denote* $\Pi \cap \{0,1\}^n$. *The explicit construction problem* EC_Π *associated to* Π *is: given* 1^n, *output* $x \in \Pi_n$ *(or determine that none exist).*

We say that Π *is total if* $\Pi_n \neq \emptyset$ *for all n; we say that it is dense if there is a fixed c so that* $|\Pi_n|/2^n \geq 1/n^c$ *for all n.*

We will sometimes abuse notation and refer to Π directly as both a language and its associated explicit construction problem. In all cases of interest, Π is total, since an explicit construction problem only presents itself once the question of existence has already been settled. The reason for the terminology of "totality" is that in such cases EC_Π is a *total search problem*: every instance has a solution. We define the input to the search problem as being encoded in unary (i.e. 1^n) so that the complexity of algorithms solving it is measured as a function of n rather than $\log n$: ideally, we seek algorithms with running time $\mathrm{poly}(n)$.

As mentioned in the introduction, essentially all of the major examples of unsolved explicit construction problems are dense. The density of solutions immediately implies the existence of a nontrivial randomized algorithm for the search problem:

Lemma 1. *If Π is dense, then EC_Π is in FZPP^Π; in other words there is a randomized algorithm using a Π-membership oracle which outputs a correct answer to EC_Π with probability 1, and has polynomial runtime in expectation.*

Proof. Sample a uniform string $x \sim \{0,1\}^n$. Use a Π membership oracle to test if $x \in \Pi_n$; if the test passes output x, else repeat. □

This basic upper bound leads us to the first *conditional derandomization* result for general dense explicit construction problems, based on a classical and powerful result in derandomization:

Theorem 1 ([24, 33, 37])**.** *Assume that there is a language L computable in time $2^{O(n)}$ with a Π oracle, such that L requires Π-oracle boolean circuits of size $2^{\Omega(n)}$. Then $\mathsf{FZPP}^\Pi \subseteq \mathsf{FP}^\Pi$. Consequently, under the same assumption there is an FP^Π algorithm for EC_Π whenever Π is dense.*

For this reason, if the property Π is testable in polynomial time, then EC_Π has a deterministic polynomial time algorithm under plausible assumptions; if it is testable with an NP-oracle, then EC_Π is solvable in $\mathsf{FP}^{\mathsf{NP}}$ under similar assumptions.

There is one additional result, not covered by the general hardness/randomness connection, that gives an unconditional and highly nontrivial upper bound for dense explicit construction problems recognizable in P:

Theorem 2 ([13])**.** *If $\Pi \in \mathsf{P}$, then EC_Π has a polynomial time pseudodeterministic algorithm that works infinitely often: there is a sequence $Y = (x_n \in \Pi_n)_{n\in\mathbb{N}}$ and a polynomial time randomized algorithm $\mathcal{A}$ with the following property: for infinitely many n, $\mathcal{A}(1^n)$ outputs x_n with probability $\frac{2}{3}$ over its internal randomness.*

Most of the important unsolved explicit construction problems are not recognizable in polynomial time, and hence not covered by the above result; as we shall see, the class of explicit construction problems primarily investigated in this work are only recognizable in the larger class coNP. The primary exception to this is the problem of constructing prime numbers: given 1^n, output a prime $p > 2^n$. Testing membership in the primes lies in P by [3], and the primes have density $\frac{1}{O(n)}$ in the range $[2^n, 2^{n+1}]$ by the prime number theorem; hence the above result gives a polynomial time pseudodeterministic algorithm for this important explicit construction problem.

2.2 Circuit Lower Bounds as Explicit Construction Problems

Although the ultimate goal of circuit complexity is to obtain superpolynomial lower bounds for functions in NP, it remains a fundamental and difficult open problem to accomplish this even for much larger complexity classes such as E and E^{NP}, which denote the sets of languages decidable by Turing machines running in time $2^{O(n)} = \text{poly}(2^n)$ (respectively, with an NP-oracle). This problem came to more prominence after the classical work of Nisan, Wigderson and Impagliazzo mentioned above, who showed that if E (resp. E^{NP}) contains a language requiring circuits of size $2^{\Omega(n)}$ for sufficiently large n, then BPP $\subseteq$ P (resp. BPP $\subseteq \mathsf{P}^{\mathsf{NP}}$).

An important observation is that for classes running in time $2^{O(n)}$, deciding a language L on one input of length n has essentially the same cost as deciding *all inputs of length n* and printing out the entire truth table $L \cap \{0,1\}^n$. In particular:

Observation 1. *The following are equivalent for a language $L \subseteq \{0,1\}^*$:*

1. *$L \in$ E (resp. E^{NP})*
2. *There is a polynomial time algorithm (resp. with an NP-oracle) which, given 1^{2^n}, outputs the truth table of $L \cap \{0,1\}^n$.*

As an immediate consequence we have:

Corollary 1. *The following are equivalent:*

1. *There is a language in E (resp. E^{NP}) which requires circuits of size $s(n)$ for all n.*
2. *There is a polynomial time algorithm (resp. with an NP-oracle) for the following explicit construction problem: given 1^{2^n}, output the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ requiring circuits of size $\geq s(n)$.*

Hence the complexity of explicit construction problems involving the production of hard boolean functions precisely captures the truth of corresponding circuit lower bounds. In this terminology, the main theorem of [24] can be rephrased as follows:

Theorem 3 ([24])**.** *Let $\epsilon > 0$ be any fixed constant. Every language in* BPP *is polynomial time reducible to the following explicit construction problem: given 1^{2^n}, output the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ with circuit complexity $\geq 2^{\epsilon n}$.*

In other words, the explicit construction problem of producing hard truth tables is at least as hard as the simulation of polynomial time randomized algorithms.

2.3 Range Avoidance: Definition and Variants

We start by recalling the formal definition of Avoid:

Definition 2 ([28])**.** *Range avoidance, denoted formally as* Avoid*, is the following search problem: given a boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ with $m > n$, output a string $y \in \{0,1\}^m$ such that $y \notin \text{range}(C)$.*

This problem was first defined in [28] and investigated more comprehensively in [29]; in both of these works it went by the name Empty, however since the work of [41] the name "range avoidance" has become more standard. In [28, 29], the name "APEPP" is used to describe the class of search problems polynomial time reducible to Avoid; again this terminology has mostly fallen out of use in subsequent work on the topic and we will not use it here.

We can make the following basic observations about the complexity of this search problem:

Lemma 2.

1. Avoid *is a total search problem – every instance has a solution.*
2. *A candidate solution $y \in \{0,1\}^m$ can be verified for correctness in* coNP.
3. Avoid *is solvable in* $\mathsf{FZPP}^{\mathsf{NP}}$.

Proof.

1. This follows from the pigeonhole principle since $m < n$ implies $|\{0,1\}^n| < |\{0,1\}^n|$.
2. To test that $y \notin \text{range}(C)$ it suffices to verify that *for all* $x \in \{0,1\}^n$, we have $C(x) \neq y$; observe that the condition $C(x) \neq y$ is checkable in polynomial time since C is a boolean circuit.

3. Sample $x \sim \{0,1\}^m$ uniformly; use an NP-oracle to test if $y \in \text{range}(C)$. Repeat until a solution is found.

□

An important observation is that the above $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm does not favor any one solution over another: on a given instance C, it outputs each element of $\{0,1\}^m \setminus \text{range}(C)$ with equal probability. In Section 5.3 we will see that there is an alternative randomized NP-oracle algorithm for Avoid which has the much stronger property that the set of possible solutions output has size exactly one.

Conditions (1) and (2) place Avoid in the class $\mathsf{TF\Sigma}_2^{\mathsf{P}}$ originally defined in [28]; indeed these two conditions may be taken as a definition of $\mathsf{TF\Sigma}_2^{\mathsf{P}}$. The class $\mathsf{TF\Sigma}_2^{\mathsf{P}}$ is rather new and largely unexplored; for a more in-depth coverage of the problems therein see [28, 31].

The last point in the above lemma, together with the general conditional derandomization result from Section 2.1, gives us the following conditional derandomization result for Avoid:

Corollary 2. *If there is a language in* E^{NP} *which requires* NP*-oracle circuits of size* $2^{\Omega(n)}$*, then* Avoid *is solvable in* $\mathsf{FP}^{\mathsf{NP}}$.

In Section 5 we will see that this theorem can be strengthened so that the circuits in question are oracle-free, and with this condition it becomes an equivalence. For now the important takeaway is that conventional complexity-theoretic wisdom says that the true upper bound for Avoid should be $\mathsf{FP}^{\mathsf{NP}}$; in Section 6 we will see some evidence that a better upper bound, for example Avoid $\in \mathsf{FP}$, is unlikely to hold.

2.3.1 Parameters

There are two main parameters through which we may restrict the problem Avoid – as we shall see later, the Avoid problem already becomes interesting in some highly restricted settings of these parameters. If our goal is to eventually obtain better Avoid algorithms in general, it is thus natural to study these weaker instances as a starting point.

Stretch: The *stretch* of an avoid instance refers to the relation between the number of output bits m and input bits n of the Avoid instance C. By definition we require $m > n$, however the problem becomes potentially easier when $m > 2n$ or $m > n^2$. In some cases it is relevant to study the ratio $\frac{m}{n}$, and in other cases the difference $m - n$. We will not fix one of these as the formal definition of a stretch parameter; instead we will simply say that an avoid instance has "stretch $n \mapsto m$" if it is of the form $C : \{0,1\}^n \to \{0,1\}^m$.

Circuit Complexity: For a circuit class C, e.g. $C \in \{\mathsf{AC}^0, \mathsf{NC}^1, \mathsf{P/poly}\}$, we define C-Avoid to be the special case of Avoid where the instance $C : \{0,1\}^n \to \{0,1\}^m$ has each output bit given by a circuit from the class C. By default Avoid = $\mathsf{P/poly}$-Avoid.

3 Reductions to Avoid

In this section we survey some of the techniques used to reduce problems to Avoid. We start with the most basic (and arguably most important) class of reductions, involving the production of hard boolean functions.

3.1 Hard Functions

The general scenario in the subject of non-uniform complexity lower bounds is the following: we have a finite set U, for example $U = \{0,1\}^n$, and a class of nonuniform "algorithms" $\mathcal{A}$ computing functions $U \to \{0,1\}$. $\mathcal{A}$ induces a set $\mathcal{F} \subseteq \{0,1\}^U$ of those functions which are computed by one of the algorithms $A \in \mathcal{A}$. We would like to exhibit an explicit function $f : U \to \{0,1\}$ which is not in $\mathcal{F}$, and is hence "hard" in the computational model $\mathcal{A}$. A crucial property of essentially all computational models of interest is that there are only a few "easy functions:" $|\mathcal{F}| << |\{0,1\}^U| = 2^{|U|}$. Hence the pigeonhole principle tells us that there exists $f \in \{0,1\}^U \setminus \mathcal{F}$, which must be a hard function.

Peering into the proof $|\mathcal{F}| << |\{0,1\}^U|$, the argument typically goes as follows: if $f \in \mathcal{F}$ then by definition there is some efficient computational device $A \in \mathcal{A}$ which computes it. If A computes f on every input, then from the description of A we may fully recover the function f. The fact that the algorithms in $\mathcal{A}$ are "simple" directly implies that the device $A \in \mathcal{A}$ may be explicitly described using some number of bits $m << |U|$. Hence we may define the function Eval : $\{0,1\}^m \to \{0,1\}^U$, which given the description of some $A \in \mathcal{A}$, outputs the function f which A computes. Then we have $\mathcal{F} \subseteq$ range(Eval), and hence any point outside the range of Eval is a hard function. This directly yields a reduction from finding a hard function to Range Avoidance by interpreting Eval as an Avoid instance; the only details to be checked are: (1) the function Eval can be computed efficiently, and (2) the encoding is sufficiently succinct so that $m < |U|$ while keeping the "sizes" of machines under consideration in $\mathcal{A}$ as large as possible (so as to prove the best quantitative lower bounds).

Essentially the strongest models[2] satisfying the first condition are boolean

[2]Large communication complexity classes such as $\mathsf{PSPACE}^{\mathsf{cc}}$ are (conjecturally) incomparable to $\mathsf{P/poly}$ and provide other examples of computational lower bounds reducible to Avoid which do not follow directly from Theorem 4, see [29].

circuits and Turing machines, where we obtain:

Theorem 4 ([10, 29])**.**

1. *Producing* $f : \{0,1\}^n \to \{0,1\}$ *requiring boolean circuits of size* $2^n/n$ *is reducible in* $\text{poly}(2^n)$ *time to* Avoid.

2. *For any fixed polynomial p, producing* $x \in \{0,1\}^n$ *which cannot be printed by any Turing machine of length* $n-2$ *running in* $p(n)$ *steps is reducible in* $\text{poly}(n)$ *time to* Avoid.

Note that every boolean function has circuits of size $(1 + o(1))2^n/n$, hence reductions to Avoid can achieve close to maximally hard boolean functions. Obtaining the exact bound $2^n/n$ is good example of the sometimes nontrivial work involved in choosing the right encoding of machines in our complexity class; an easy analysis in [29] yields the bound $2^n/2n$ by encoding circuits naively as DAGs; to obtain the bound $2^n/n$ in [10] requires a more careful encoding.

This reduction has some important consequences in light of the discussion in Section 2.2: since explicit constructions of hard truth tables correspond to circuit lower bounds for exponential time classes, we have:

Corollary 3 ([29])**.** *If* Avoid $\in \mathsf{FP}$ *(resp.* Avoid $\in \mathsf{FP}^{\mathsf{NP}}$*) then* E *(resp.* E^{NP}*) contains a language requiring circuits of size* $2^n/n$ *for all n.*

In Section 5 we will see that this can be strengthened to an "if and only if" in the case of $\mathsf{FP}^{\mathsf{NP}}, \mathsf{E}^{\mathsf{NP}}$. Combining this with the hardness/randomness connection of [24, 37] mentioned in Section 2.1, we can also observe:

Corollary 4 ([29])**.** *Every language in* BPP *is polynomial time reducible to* Avoid.

This corollary can be proven in a more direct fashion without appealing to the powerful results of [24, 37], by instead giving a direct reduction from the problem of constructing *pseudorandom generators* to Avoid. This argument is carried out in [29].

Fine-Grained Considerations: Recall that the two main parameters of interest for an Avoid instance $C : \{0,1\}^n \to \{0,1\}^m$ are the stretch $n \mapsto m$ and the circuit complexity of C. When reducing hard function construction to Avoid, there is a natural tradeoff between the quantitative strength of the lower bound and stretch of Avoid instance; when we consider "smaller" algorithms in a class $\mathcal{A}$ we typically get shorter descriptions. For example if we wish to produce $f : \{0,1\}^n \to \{0,1\}$ hard for circuits of size s, we can reduce to an Avoid instance with stretch $\text{poly}(s(n)) \mapsto 2^n$. When it comes to the circuit complexity of the Avoid instances produced by

the reductions, the relevant question is the complexity of computing Eval, each of whose output bits is essentially an instance of the "Circuit Value Problem" for the class of algorithms $\mathcal{A}$. Roughly speaking, if $\mathcal{A}$ corresponds to a "typical" class of circuits such as $\mathsf{AC}^0, \mathsf{NC}^1$, etc., the circuit value problem for $\mathcal{A}$ can be computed by circuits in the same class $\mathcal{A}$; sometimes this is referred to as a circuit class having the "universal property," see [41] for a further discussion. As a consequence we have:

Theorem 5 ([41])**.** *Let C be a circuit class with the universal property, for example $C \in \{\mathsf{AC}^0, \mathsf{ACC}^0, \mathsf{TC}^0, \mathsf{NC}^1\}$. Then producing $f : \{0,1\}^n \to \{0,1\}$ requiring C-circuits of size $s(n)$ is polynomial time reducible to a C-*Avoid *instance with stretch* $\mathrm{poly}(s(n)) \mapsto 2^n$.

*As a corollary (recall Corollary 1), if C-*Avoid *is solvable in polynomial time (resp. with an* NP*-oracle) then* E *(resp.* E^{NP}*) requires C-circuits of size $2^{\Omega(n)}$.*

Note that, for circuit classes such as TC^0 and NC^1, it is an important open problem to show that E^{NP} does not have $\mathrm{poly}(n)$ size circuits from the class. By the above connection, this means that we would already achieve a breakthrough from, for example, a $2^{O(n)}$ time NP-oracle algorithm for TC^0-Avoid in the stretch regime $n^{\omega(1)} \mapsto 2^n$, i.e. when the number of outputs is almost exponential in the number of inputs.

3.2 Sparse String Encodings

Say we have a map $C : \{0,1\}^m \to \{0,1\}^n$, and an explicit construction problem $\Pi \subseteq \{0,1\}^n$ we want to solve which has the following property: if $x \notin \Pi$ then $\Delta(x, \mathrm{range}(C)) \leq d$, where $\Delta(x, A)$ denotes the Hamming distance of x to its closest point in a set $A \subseteq \{0,1\}^n$. In other words, while we have not quite completed a reduction from Π-construction to Avoid, we have exhibited an Avoid instance whose range has small hamming distance to all the "bad strings" failing to have property Π. This implies that Π is reducible to the following range avoidance variant: given C, output a point which has Hamming distance $> d$ from every element of $\mathrm{range}(C)$; this variant has been referred to as Remote Point in the literature [9].

In the case that m is sufficiently smaller than n, remote point can be reduced back to the standard variant of range avoidance. For this it suffices to show that sparse boolean strings can be given compressed representations decodable in polynomial time, which is standard:

Lemma 3. *For any $k \leq n$, there exists a polynomial time computable map $S : \{0,1\}^{\lceil \log \binom{n}{k} \rceil} \to \{0,1\}^n$ such that for every string $x \in \{0,1\}^n$ of hamming weight exactly k, $y \in \mathrm{range}(S)$.*

A simple proof of this result can be found in [19]. Clearly we may extend this scheme to encode a string of hamming weight at most k while incurring an additive cost of at most $\log k$ bits to specify the weight. This gives:

Corollary 5. *If $C : \{0,1\}^m \to \{0,1\}^n$ is given, and $\log \binom{n}{k} + \log k + m < n$, then finding a string of hamming distance $> k$ to* range(C) *is reducible in polynomial time to* Avoid.

This construction occurs repeatedly in reduction to range avoidance; many important explicit construction problems can be reduced most naturally to remote point, because their definition involves being far in hamming distance from a set of simple objects. Some examples where this has been applied include:

1. A boolean function $f : \{0,1\}^n \to \{0,1\}$ is hard on average for small circuits provided it is far in hamming distance from the set of low complexity functions (used in [29]).

2. A matrix is rigid provided it is far in hamming distance from the low-rank matrices (used in [29]).

3. A matrix generates a good linear error correcting code if each row is far in hamming distance from the span of the others (used in [21]).

4. A list of strings $(x^j \in \{0,1\}^n)_{j \in J}$ is a good pseudorandom generator for size-s circuits if the string $(x^j_i)_{j \in J} \in \{0,1\}^J$ is far in hamming distance from $(D(x^j_1, \ldots, x^j_{i-1}))_{j \in J}$ for all next bit predictors $D : \{0,1\}^{i-1} \to \{0,1\}$ of circuit size $O(s)$ and all $i < n$ (used in [29]).

In each case, combining these facts with an application of Corollary 5 yields the reduction to Avoid in a rather straightforward way.

In [21], it is shown that two of the above problems (construction of rigid matrices and strong error correcting codes) can be reduced to Avoid for restricted circuit classes, in particular NC^1-Avoid. It turns out that the main difficulty is in obtaining a sparse-string encoding lemma along the lines of Lemma 3, where moreover the decoder has restricted circuit complexity. The authors of [21] accomplish this using a construction from static data structure complexity:

Lemma 4 ([21, 38])**.** *For each $k \leq n$ there is a map $S^n_k : \{0,1\}^m \to \{0,1\}^n$ with the following properties:*

1. $m \leq \log \binom{n}{k} + O(\frac{n}{\log^2 n})$

2. *Every string $x \in \{0,1\}^n$ of hamming weight at most k lies in the range of S^n_k.*

3. *Each output bit of S^n_k is computable by a $O(\log n)$-depth decision tree over the inputs, and moreover there is a* $\mathrm{poly}(n)$*-time algorithm which outputs this list of decision trees.*

In particular, the last bullet means that the map S^n_k is computable in $\mathsf{AC}^0 \subseteq \mathsf{NC}^1$. Finally we mention an alternative approach used in [9] to reduce remote point to range avoidance in restricted circuit classes based on error correcting codes. Roughly, given a range avoidance instance C, we replace it with $\mathrm{Dec} \circ C$ where Dec, Enc are the decoder/encoder for a suitable error correcting code. If x is outside the range of $\mathrm{Dec} \circ C$ then $\mathrm{Enc}(x)$ will be far in hamming distance from $\mathrm{range}(C)$. This construction enjoys a superior stretch parameter at the cost of higher circuit complexity (note that Lemma 4 cannot obtain $n \mapsto n^2$ stretch in any sparsity regime because of the additive $n/\log n$ penalty).

3.3 Strongly-Explicit Extractors and Related Constructions

Here we describe a technique which allows us to reduce to Avoid the construction of circuits computing functions $f : \{0,1\}^n \to \{0,1\}^n$ with certain strong pseudorandom properties. The reductions here run in time $\mathrm{poly}(n)$, and produce a circuit for f of size $\mathrm{poly}(n)$; this is in contrast to the reductions in Section 3.1 which produce n-bit boolean functions in time $2^{O(n)}$ given as truth tables.

The reductions here hinge on a simple high-level construction involving k-wise independent generators. To explain the method in sufficient generality we require the following definition:

Definition 3. *Let $\mathcal{A} \subseteq \binom{\{0,1\}^n}{k}$, $\mathcal{B} \subseteq (\{0,1\}^n)^k$ be given. We say that $f : \{0,1\}^n \to \{0,1\}^n$ is a (A, B) extractor if whenever $(x_1, \ldots, x_k) \in \mathcal{A}$, $(f(x_1), \ldots, f(x_k)) \notin \mathcal{B}$.*

The idea behind the definition is as follows. We think of $\mathcal{A} \subseteq \{S \subseteq \{0,1\}^n, |S| = k\}$ as defining a class of *structured* subsets of k inputs, with $|\mathcal{A}| << \binom{2^n}{k}$. For any fixed set of k inputs $(x_1, \ldots, x_k)$ and a function $f : \{0,1\}^n \to \{0,1\}^n$, we may think of $(f(x_1), \ldots, f(x_k)) \in \mathcal{B}$ as some unlikely event, i.e. $|\mathcal{B}| << 2^{nk}$. We then have that f is an (A, B) extractor iff $f(S)$ avoids the event $f(S) \in \mathcal{B}$ whenever $S \in \mathcal{A}$. If $\Pr[(y_1, \ldots, y_k) \in \mathcal{B}]$ is very small for a random sequence $y_1, \ldots, y_k \sim \{0,1\}^n$, then for any fixed $S = \{x_1, \ldots, x_k\} \in \mathcal{A}$ of size k, if we choose f uniformly at random we are very likely to have $f(S) = \{f(x_1), \ldots, f(x_k)\} \notin \mathcal{B}$. If $|\mathcal{A}|$ is also sufficiently small then we can union bound over $S \in \mathcal{A}$ and argue that a random f will be an $(\mathcal{A}, \mathcal{B})$ extractor with high probability.

This existence argument can be improved in the following sense: say that instead of choosing the function $f \sim \{0,1\}^n \to \{0,1\}^n$ uniformly at random, we sample it from a k-wise independent distribution $\mathcal{F}$: for every fixed distinct $x_1, \ldots, x_k \in \{0,1\}^n$, $(f(x_1), \ldots, f(x_k))$ is distributed uniformly on $(\{0,1\}^n)^k$ when

$f \sim \mathcal{F}$. Then in this case we have $f(S) \notin \mathcal{B}$ with the same high probability over $f \sim \mathcal{F}$, and the rest of the argument goes through as before. Thus we can argue for the existence of an $(\mathcal{A}, \mathcal{B})$ extractor inside of any k-wise independent function family $\mathcal{F}$.

The following reduction formalizes this using the standard construction of k-wise independent families by univariate polynomials over a finite field. The reduction works provided that the "smallness" of $\mathcal{A}, \mathcal{B}$ can also be exhibitted by reductions to Range Avoidance, i.e. we can cover these sets by the range of some explicit length-expanding functions $A : \{0,1\}^\ell \to \binom{\{0,1\}^n}{k}$, $B : \{0,1\}^r \to \{0,1\}^{nk}$. The technique here is quite similar to an old result of Razborov [40].

Lemma 5. *Let $A : \{0,1\}^\ell \to \binom{\{0,1\}^n}{k}$, $B : \{0,1\}^r \to \{0,1\}^{nk}$ be given as circuits. Provided $\ell + r < nk$, the following problem is reducible in polynomial time to range avoidance: output the description of a polynomial size circuit $f : \{0,1\}^n \to \{0,1\}^n$ which is an* $(\mathrm{range}(A), \mathrm{range}(B))$ *extractor. Moreover:*

1. *The circuit f produced by the reduction lies in $\mathsf{AC}^0[2]$.*
2. *If $A, B \in \mathsf{NC}^2$ the Range Avoidance instance produced by the reduction also lies in NC^2.*

Proof. We construct an instance of range avoidance $C : \{0,1\}^{\ell+r} \to \{0,1\}^{nk}$. Elements of $\{0,1\}^{nk}$ are interpreted as vectors $\bar{\alpha} = (\alpha_1, \ldots, \alpha_k) \in \mathbb{F}_{2^n}^k$, and to each $\bar{\alpha}$ we associate the function $f_{\bar{\alpha}} : \{0,1\}^n \to \{0,1\}^n$ given by:

$$f_{\bar{\alpha}}(x) = \sum_{i=1}^{k} \alpha_i x^{i-1}$$

where $\{0,1\}^n$ and $\mathbb{F}_{2^n}$ are associated in some standard way. We will construct C so that if $f_{\bar{\alpha}}$ fails to be a $(\mathrm{range}(A), \mathrm{range}(B))$ extractor, then $\bar{\alpha} \in \mathrm{range}(C)$ which will give the theorem. The fact that $f_{\bar{\alpha}}$ lies in $\mathsf{AC}^0[2]$ for any choice of $\bar{\alpha}$ follows from results of [22] on the complexity of arithmetic over $\mathbb{F}_{2^n}$.

C is constructed as follows. We interpret the input as encoding (a, b) with $a \in \{0,1\}^\ell$ and $b \in \{0,1\}^r$. Let $(x_1, \ldots, x_k) = A(a)$ and let $(v_1, \ldots, v_k) = B(b)$. We then compute the unique degree $k-1$ polynomial $p \in \mathbb{F}_{2^n}[x]$ such that $p(x_i) = v_i$ for all i; this can be accomplished in polynomial time using Gaussian elimination on the corresponding Vandermonde matrix (by assumption $A(a)$ outputs a list of k distinct strings). Finally C outputs the coefficients $(\alpha_1, \ldots, \alpha_k)$ of p. Clearly we have $\bar{\alpha} \in \mathrm{range}(C)$ whenever $f_{\bar{\alpha}}$ fails to be a $(\mathrm{range}(A), \mathrm{range}(B))$ extractor. The complexity of C is upper bounded by the complexity of computing A, B, together with that of Gaussian elimination over $\mathbb{F}_{2^n}$ which lies in NC^2. □

The above lemma can be used to reduce the construction of various "structured seed extractors" with near-optimal parameters to Range Avoidance. An important case of a structured seed extractor is a 2-source extractor: in this case the domain of the function f is $\{0,1\}^n \times \{0,1\}^n$, and the family of simple sets $\mathcal{A}$ is the set of "combinatorial rectangles:" sets of the form $L \times R$ for some $L, R \subseteq \{0,1\}^n$, $|L| = |R| = k$. Such a combinatorial rectangle has size k^2 but can be encoded using $2kn$ bits, which yields the appropriate circuit A for the reduction. The "bad event" $\mathcal{B}$ is that the first bit of the function f is biased towards 1 or 0 over the rectangle; here we can use the sparse string encodings from Section 3.2 to form the circuit B. Details of this reduction can be found in [29]. Recent advances have yielded 2-source extractors with parameters very close to optimal, but still not meeting the bounds attainable here [8, 34]. Other kinds of structured-seed extractors studied in the literature include extractors for varieties and for affine spaces, see [7, 14].

4 Reducibilities Between Avoid Variants

In this section we cover some known reductions between variants of Avoid which are restricted in one of the two main parameters: stretch and circuit complexity. Most of the complexity classes C for which we typically study C-Avoid are restricted in terms of *circuit depth*; when we refer to the depth of an Avoid instance $C : \{0,1\}^n \to \{0,1\}^m$, we mean the depth of C as a boolean circuit.

4.1 Increasing the Stretch, and its Effect on Circuit Depth

The first natural question to ask is whether Avoid is robust to changes in the stretch parameter: can we reduce a general Avoid instance in polynomial time to an instance with stretch $n \mapsto n^{100}$? The following lemma is essentially the only known result along these lines, which says that such reductions are possible, provided we are afforded the use of an NP-oracle. The proof of this lemma is simple and parallels classical results in proof complexity and cryptography concerning reducibility amongst pigeonhole principles and amongst pseudorandom generators respectively [17, 39].

Before explaining the reduction, we mention an important parameter to keep track of, which we'll call the *depth complexity* of the reduction: the reduction will take a circuit $C : \{0,1\}^n \to \{0,1\}^{n_1}$ and produce a circuit $C' : \{0,1\}^n \to \{0,1\}^{n_2}$ which makes some oracle calls to C and otherwise performs some basic (NC^0) computations inbetween; we refer to the depth of the reduction as the depth of the longest chain of oracle calls C' makes to C. The relevance is that if C is given as a circuit of depth d_0, and we construct C' by a reduction with depth complexity d, then C' will be computed by an (explicitly given) circuit of depth $d_0 \cdot d$. By keeping

track of this depth complexity we can analyze how our stretching reductions affect the depth of the circuits defining our Avoid instances. To state the quantitative condition in the tightest way we need the following function:

Definition 4. *For $n' > n$ and $d \geq 1$, define $\Delta_d(n' \mid n)$ by induction on d as follows:*

1. $\Delta_1(n' \mid n) = n'$
2. $\Delta_{d+1}(n' \mid n) = \lfloor \frac{\Delta_d(n'|n)}{n} \rfloor \cdot (n' - n) + \Delta_d(n' \mid n)$

The function $\Delta_d(n' \mid n)$ is a kind of integral approximation to $(n'/n)^d n$. An operational interpretation is as follows: we start with n items in a basket, and get to perform the following update for d steps: choose any grouping of the current items into bundles, and replace each bundle of size exactly n by a larger bundle of size n'. The maximum number of items obtainable in this way after d steps equals $\Delta_d(n' \mid n)$.

The relevant fact used in the following lemma is that, if $f = g \circ h$, $y \notin \text{range}(f)$, then either $y \notin \text{range}(g)$, or else for any preimage $g(z) = y$ we must have $z \notin \text{range}(h)$.

Lemma 6. *Let $n_2 \geq n_1 > n$ be given. If $\Delta_d(n_1 \mid n) \geq n_2$, then* Avoid *with stretch $n \mapsto n_1$ can be reduced to stretch $n \mapsto n_2$ in* $\text{poly}(n_2)$ *time and depth complexity d with an* NP*-oracle.*

Proof. We prove the lemma by induction on d. Note that Avoid can always be reduced downward in stretch by ignoring some output bits. In the case $d = 1$, $n_2 \leq n_1$, and so by this observation there is nothing to prove. If the lemma holds up to d, then we can construct an instance $C' : \{0,1\}^n \to \{0,1\}^m$ computable with depth d calls to C, so that $m \leq \Delta_d(n_1 \mid n)$, and given any solution to C' Avoid we can find one for C in polynomial time with an NP-oracle. We now write m as $m = k \cdot n + \ell$ for maximal k subject to $\ell \geq 0$; we can then define C'' which, given $x \in \{0,1\}^n$, first applies C' to get an m bit string z. We can write z as $z_1, \ldots, z_k, z_{k+1}$ where $|z_j| = n$ for $j \leq k$, $|z_{k+1}| = \ell$. Now C'' outputs $C(z_1), \ldots, C(z_k), z_{k+1}$ which has length $n_1 \cdot k + \ell \leq \Delta_{d+1}(n_1 \mid n)$. Cleary the depth complexity of C'' is $d + 1$. Given $y_1, \ldots, y_k, y_{k+1}$ outside the range of C'', we can use an NP-oracle to search for a preimage of each of y_j, $j \leq k$ under C; if one has no preimage we solve Avoid for C, otherwise we find $z_1, \ldots, z_k, z_{k+1}$ which must lie outside the range of C'; by induction we have an NP-oracle algorithm which then maps this to an Avoid solution for C. □

Some important parameter regimes are highlighted below:

Reducing stretch $n \mapsto n_1$ to $n \mapsto n_2$		
n_1	n_2	Depth Cost
$n+1$	$2n$	n
$2n$	n^2	$\log n$
$1.01n$	$100n$	$O(1)$
$n^{1.01}$	n^{100}	$O(1)$

As mentioned previously, the methods here are almost identical to those used for a related problem: given a candidate cryptographic pseudorandom generator (PRG) $G : \{0,1\}^n \to \{0,1\}^m$, how and when can we construct a generator G' with better stretch parameters whose security can be based on that of G? On the positive side we can achieve the same stretching reductions with identical depth complexities for PRGs as in Lemma 6 for Avoid. Despite these strong similarities, no results are currently known which connect these problems in some more formal framework.

4.2 Reducing the Circuit Complexity

With respect to the circuit complexity of the range avoidance instance, there is one very powerful result known which allows us to reduce C-Avoid to C'-Avoid for C' much smaller than C. The result is due to [41] and utilizes the method of randomized encodings from low-depth cryptography [1]. To state it we need the following definition:

Definition 5. *We say that a circuit $C : \{0,1\}^n \to \{0,1\}^m$ is in NC^0_k if each output of C depends on at most k inputs of C.*

We then have:

Theorem 6 ([41])**.** NC^1-Avoid *is polynomial time reducible to* NC^0_4-Avoid.

Since we have the circuit class inclusions $\mathsf{NC}^0_4 \subseteq \mathsf{AC}^0 \subseteq \mathsf{ACC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1$, this result implies that for all circuit classes $C, C' \in \{\mathsf{NC}^0_4, \mathsf{AC}^0, \mathsf{ACC}^0, \mathsf{TC}^0, \mathsf{NC}^1\}$, the problems C-Avoid and C'-Avoid are polynomial time equivalent.

An important caveat is that this reduction completely destroys the stretch of the Avoid instance we start with: if the original instance of NC^1-Avoid has stretch $n \mapsto n^{100}$, the NC^0_4-Avoid instance we obtain from the reduction in Theorem 6 will have stretch roughly $m + n \mapsto m + n^{100}$, where m is the size of the formula defining the original NC^1-Avoid instance. In particular if m is a polynomial larger than n^{100}, this will be in the stretch regime $n \mapsto n + n^{1-\Omega(1)}$ (up to a reparameterization $n := m+n$) which cannot be boosted to a regime $n \mapsto (1+\Omega(1))n$ without incurring an $n^{\Omega(1)}$ blowup in depth if we rely on Lemma 6. This distinction becomes crucial in

light of some later results we will see in Section 5, which show that for NC^0_k-Avoid and even ACC^0-Avoid, there are unconditional FP and $\mathsf{FP}^{\mathsf{NP}}$ algorithms when the stretch is sufficiently large.

5 Upper Bounds for Range Avoidance

In this section we explore some unconditional upper-bounds on the Range Avoidance problem which significantly improve those mentioned in Section 2.3. We have already seen that suitably strong derandomization assumptions imply that Avoid lies in $\mathsf{FP}^{\mathsf{NP}}$. However an unconditional proof of this would immediately imply $\mathsf{BPP} \subseteq \mathsf{P}^{\mathsf{NP}}$ and more generally that E^{NP} requires $2^{\Omega(n)}$ size circuits almost everywhere, and thus seems currently out of reach.

The results covered here will allow us to obtain the derandomization Avoid $\in \mathsf{FP}^{\mathsf{NP}}$ under weaker assumptions, as well as place it unconditionally in a complexity class smaller than $\mathsf{FZPP}^{\mathsf{NP}}$. For restricted variants of Avoid, we will discuss two other sets of results which give unconditional $\mathsf{FP}^{\mathsf{NP}}$ and FP algorithms.

5.1 The Tree Construction

The first two results we discuss in this section, in addition to the result discussed in Section 7.3, all hinge on a single construction dating back to [17, 39] and used again in [25], which also bears a strong resemblance to the stretching reduction from Section 4. The application of this construction to the reduction in Section 5.2 is essentially identical to its use in [25]. The application to the algorithm in Section 5.3 bears a much stronger resemblance to [39] (and to a lesser extent [17]), however it requires a more careful analysis.

In the following, we think of the circuit $C : \{0,1\}^n \to \{0,1\}^{2n}$ as an instance of Avoid which we aim to solve. As mentioned in Section 4 we can reduce the general case of Avoid to the case of doubling-stretch, so this is without loss of generality.

Definition 6. *Let $C : \{0,1\}^n \to \{0,1\}^{2n}$. C_0 (resp. C_1) is the function obtained by restricting C to the first (resp. last) n bits of output. For any binary string s, define $C_s : \{0,1\}^n \to \{0,1\}^n$ inductively as follows:*

1. *C_ϵ is the identity map where ϵ is the empty string.*
2. *$C_{bs}(x) = C_s(C_b(x))$ for any $b \in \{0,1\}$, $s \in \{0,1\}^*$.*

If $S \subseteq \{0,1\}^*$ is a family of strings it is perhaps most natural to think of $(C_s)_{s\in S}$ as a family of functions $\{0,1\}^n \to \{0,1\}^n$. However for our purposes it will be most natural to invert our perspective:

Definition 7. *Let C be as above. For any set of strings $S \subseteq \{0,1\}^*$, consider the following function*

$$C^S : \{0,1\}^n \to \{S \to \{0,1\}^n\}$$

given by $C^S(x) = \{s \mapsto C_s(x)\}$.

The function C^S maps elements of $\{0,1\}^n$ to *functions* $S \to \{0,1\}^n$. We will concern ourselves only with those $S \subseteq \{0,1\}^*$ which are prefix-free: no string in S is the prefix of another. In this case we may identify S with a binary tree whose leaves are S. In the important special case $S = \{0,1\}^d$, this yields a perfect binary tree of depth d.

In this terminology we may think of C^S itself a succinctly represented instance of Avoid: it takes strings of length n and outputs functions $S \to \{0,1\}^n$, which we may in turn interpret as strings of length $|S|n$. This succinct Avoid instance has two useful properties:

Lemma 7. *Let C be as above, $S \subseteq \{0,1\}^*$ prefix free. Then $C^S : \{0,1\}^n \to \{S \to \{0,1\}^n\}$ has the following properties:*

1. *Given C, x and $s \in S$ we may compute $C^S(x)(s)$ uniformly in time* $\mathrm{poly}(|C|, |s|)$.
2. *Say $s_0, s_1 \in S$ have a common parent r, i.e. $s_0 = r0$ and $s_1 = r1$. Let $S' = S \cup \{r\} \setminus \{s_0, s_1\}$. Let $f : S \to \{0,1\}^n$ be given, let $v_0 = f(s_0), v_1 = f(s_1)$, and suppose v is a preimage of (v_0, v_1) under C. Then the for the function f' given by*

$$f'(s) = \begin{cases} f'(s) = v \text{ if } s = r \\ f'(s) = f(s) \text{ otherwise} \end{cases}$$

we have that $f \notin \mathrm{range}(C^S) \to f' \notin \mathrm{range}(C^{S'})$.

Proof. For the first part, $C^S(x)(s)$ is equal to $C_s(x)$ in our original notation. To compute this value we expand $s = (b_1, \ldots, b_d)$ where $d = |s|$ and output $C_{b_d}(\cdots C_{b_1}(x)\cdots)$ which requires $|s|$ evaluations of the circuit C.

For the second part, say that $f' \in \mathrm{range}(C^{S'})$ and let $x \in \{0,1\}^n$ be its preimage. Then for all $s \in S \setminus \{s_0, s_1\}$, $C^S(x)(s) = C^{S'}(x)(s) = f'(s) = f(s)$. On the other hand for r we have $C^{S'}(x)(r) = v$, and hence $C^S(x)(s_b) = C_b(C^{S'}(v)) = v_b$ for both $b \in \{0,1\}$ by the assumption that v is the preimage of (v_0, v_1) under C. Hence overall we have $C^S(x)(s) = f(s)$ for all s, and hence $C^S(x) = f$ and so $f \in \mathrm{range}(C^S)$. □

Interpreting S as the leaves of a binary tree T, we may visualize the lemma as follows: each function $f : S \to \{0,1\}^n$ is a labeling of the leaves of T by elements of $\{0,1\}^n$. Now, C^S is a map whose domain is $\{0,1\}^n$ and whose range is the set of functions $f : S \to \{0,1\}^n$; each $f \in \mathrm{range}(C^S)$ is thus a certain "highly compressible" leaf-labeling of T, with the following properties:

1. Given the "compressed representation" of f, namely the n-bit string x such that $C^S(x) = f$, we may compute any label $f(\ell)$ of any leave $\ell \in T$ in time polynomial in the *depth* of the leave ℓ.

2. Given a tree T labeled by a "compressible" f as above, if we prune the tree T to T' by removing two leaves with a common parent, we can attempt to relabel T' by a compressible f' in the following way: take the labels v, v' assigned to these leaves, find some r such that $C(r) = (v, v')$, and label the new leaf in T' by r. If a preimage is found we succesfully generate f', otherwise we solve Avoid for C by finding a string (v, v') outside its range.

5.2 Avoid Reduces to Hard Truth Tables

The first important upper bound for Avoid is the following:

Theorem 7 ([25, 29])**.** *There is an* $\mathsf{FP}^{\mathsf{NP}}$ *reduction from* Avoid *to the following problem* ϵ-Hard*: given* 1^{2^n}*, output the truth table of a boolean function* $f : \{0,1\}^n \to \{0,1\}$ *which requires boolean circuits of size* $\geq 2^{\epsilon n}$*.*

This is a rather immediate consequence of Lemma 7:

Proof. Let $C : \{0,1\}^n \to \{0,1\}^{2n}$ be a given Avoid instance. Set $d = K \log |C|$ for an appropriate constant K to be determined later. Consider the tree $C^{\{0,1\}^d}$. This is a map of the form $C^{\{0,1\}^d} : \{0,1\}^n \to \{\{0,1\}^d \to \{0,1\}^n\}$ and can be interpreted an instance of Avoid with input length n and output length $2^d n = \text{poly}(|C|)$. By Lemma 7 each $g : \{0,1\}^d \to \{0,1\}^n$ in the range of $C^{\{0,1\}^d}$ may be computed by a circuit of size $O(|C|d)$. Setting K sufficiently large w.r.t. ϵ, if $g : \{0,1\}^d \to \{0,1\}$ requires boolean circuits of size $2^{\epsilon d}$ then the function $f : \{0,1\}^d \to \{0,1\}^n$ given by $f(x) = (g(x), \ldots, g(x))$ cannot lie in the range of $C^{\{0,1\}^d}$. Hence any solution to ϵ-Hard will supply us with some $f \notin \text{range}(C^{\{0,1\}^d})$.

It remains to show that we can use f to find $y \notin \text{range}(C)$. For this we use the second part of Lemma 7. Initializing $S = \{0,1\}^d$ we have a string $g \notin \text{range}(C^S)$. Choose two leaves s_0, s_1 of S with a common parent r and let $v_0 = g(s_0), v_1 = g(s_1)$. Use an NP-oracle to search for the lexicographically first $v \in \{0,1\}^n$ which is a preimage of (v_0, v_1); if none exist we have solved Avoid for C. If we find a preimage v then set:

$$S' = S \cup \{r\} \setminus \{s_0, s_1\}$$

$$g'(s) = \begin{cases} f'(s) = v \text{ if } s = r \\ f'(s) = f(s) \text{ otherwise} \end{cases}$$

By Lemma 7 we have $g' \notin \text{range}(C^{S'})$. We keep repeating this procedure, at each step generating a new set S' and some $g' \notin \text{range}(C^{S'})$, with the property that each

successive S' is a subtree of the last. If we have not found a solution by the time we reach the point $S = \{0,1\}$ and $g = \{0 \mapsto v_0, 1 \mapsto v_1\}$ then we have $g \notin \text{range}(C^{\{0,1\}})$ which by definition means $(v_0, v_1) \notin \text{range}(C)$ and we are done. □

Recalling Corollary 1 and Theorem 4, we observe the following equivalence between circuit lower bounds for E^{NP} and $\mathsf{FP}^{\mathsf{NP}}$ Avoid algorithms:

Corollary 6 ([29])**.** *The following are equivalent:*

1. *There is a language in* E^{NP} *which requires circuits of size* $2^n/n$ *for sufficiently large n.*

2. *There is a language in* E^{NP} *which requires circuits of size* $2^{\Omega(n)}$ *for all n.*

3. Avoid $\in \mathsf{FP}^{\mathsf{NP}}$

Proof. (1)→(2) is immediate. Recall that by Corollary 1 the existence of a language in E^{NP} requiring $2^{\Omega(n)}$ sized circuits is equivalent to the existence of a constant $\epsilon > 0$ and an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for the problem of constructing truth tables of hardness $2^{\epsilon n}$. Combining this with the above lemma yields (2)→(3). Finally, (3)→(1) is given by Theorem 4. □

This places the question Avoid $\in \mathsf{FP}^{\mathsf{NP}}$ in rare company as a derandomization question which is *exactly equivalent* to a computational hardness assumption; classical hardness/randomness connections, such as those yielding $\mathsf{BPP} = \mathsf{P}$ [24, 37], are only known to hold in one direction.

5.3 Pseudodeterministic Algorithms for Avoid

So far, the best unconditional upper bound we have seen for Avoid is $\mathsf{FZPP}^{\mathsf{NP}}$, the class of search problems solvable with zero error by a randomized NP-oracle algorithm running in expected polynomial time. We have also seen that under plausible assumptions, this can be derandomized to Avoid $\in \mathsf{FP}^{\mathsf{NP}}$, but proving this unconditionally would resolve several longstanding open problems in complexity theory and seems currently out of reach. In a pair of recent breakthroughs [10, 35], it was shown that Avoid can be placed unconditionally inside a complexity class lying between $\mathsf{FP}^{\mathsf{NP}}$ and $\mathsf{FZPP}^{\mathsf{NP}}$, which is called $\mathsf{psZPP}^{\mathsf{NP}}$:

Definition 8. *We say that a search problem* $\mathcal{S} \subseteq \{0,1\}^* \times \{0,1\}^*$ *is in* $\mathsf{psZPP}^{\mathsf{NP}}$ *if there is a choice function* $F : \{0,1\}^* \to \{0,1\}^*$ *assigning each instance x of* $\mathcal{S}$ *to a solution* $F(x)$ *with* $(x, F(x)) \in \mathcal{S}$*, such that F is computable in* $\mathsf{FZPP}^{\mathsf{NP}}$.

The prefix ps- stands for *pseudodeterministic*. Note that the computational model associated with $\mathsf{psFZPP}^{\mathsf{NP}}$ and $\mathsf{FZPP}^{\mathsf{NP}}$ are identical: both are randomized algorithms running in expected polynomial time with an NP-oracle, which are required to output a valid answer with probability 1. The distinction is that a general $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm is allowed to output different solutions to a given instance depending on its internal randomness, while a pseudodeterministic algorithm must output a fixed solution that depends only on the instance. As mentioned in Section 2.3, the obvious $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm for Avoid is not pseudodeterministic: its output is a uniformly random solution. The following was shown in [35], strengthening a slightly weaker result shown immediately prior by [10]:

Theorem 8 ([10, 35])**.** Avoid $\in \mathsf{psZPP}^{\mathsf{NP}}$

The most important corollary of this new upper bound is that it in turn implies a new *lower bound*, via the connection established in Section 2.2:

Corollary 7 ([10, 35])**.** *There is a language in* $\mathsf{ZPE}^{\mathsf{NP}}$ *which requires circuits of size* $2^n/n$ *for sufficiently large n.*

Proof assuming Theorem 8. This is the same as Corollary 1. Since Avoid has a pseudodeterministic $\mathsf{FZPP}^{\mathsf{NP}}$ algorithm, and hard-truth table construction reduces in deterministic polynomial time to Avoid (Theorem 4), there is a pseudodeterministic $2^{O(n)}$ time randomized NP-oracle algorithm which, given n, produces $f_n : \{0,1\}^n \to \{0,1\}$ requiring circuits of size $2^n/n$. Note that the pseudodeterministic guarantee ensures that for each n, there is a *unique* boolean function f_n produced by this algorithm. This implies that the language

$$L = \{x \mid f_{|x|}(x) = 1\}$$

requires $2^n/n$ size circuits, and it can be decided in $\mathsf{ZPE}^{\mathsf{NP}}$ by running the explicit construction algorithm to produce $f_{|x|}$ as a truth table and outputting $f_{|x|}(x)$. □

The main result of [10, 35] is actually a bit stronger, giving an inclusion of Avoid in a single-valued, functional variant of the class $\mathsf{S}_2^{\mathsf{P}}$ which we do not define formally here. The pseudodeterministic algorithm is then obtained by combining this with an older result of Cai [6] which shows how to simulate $\mathsf{S}_2^{\mathsf{P}}$ in $\mathsf{ZPP}^{\mathsf{NP}}$.

The proof methods in [10, 35] also rely on the "tree construction" described in the previous section. The crucial distinction between these methods and those in the previous section is that [10, 35] consider the function C^S in the case where the prefix tree for S has depth linear in n, and hence $|S|$ can have *exponential* size. In this regime the tree cannot be written down explicitly by a polynomial time algorithm, however we can still perform certain operations on the tree implicitly. The first observation needed in [10, 35] is the following[3]:

[3] In [10, 35] they set $S = \{0,1\}^{2n}$ and use a slightly different construction. The variant described here is used in [31] and originates from [39].

Observation 2. *There is an explicit function* $f_{\text{diag}} : \{0,1\}^n \to \{0,1\}^n$*, computable uniformly in* $\text{poly}(|C|)$ *time on every input, so that* $f_{\text{diag}} \notin \text{range}(C^{\{0,1\}^n})$.

Proof. The proof is via the standard Cantor diagonalization. For an n-bit string v let $\neg v$ denote the string obtained by flipping all its bits. For $s \in S$ set $f_{\text{diag}}(s) = \neg C_s(s)$. Then for any $x \in \{0,1\}^n$, $C^{\{0,1\}^n}(x)$ differs from f_{diag} on input x; hence f_{diag} cannot lie in $\text{range}(C^{\{0,1\}^n})$. □

Hence, unlike the case of the shallower Avoid-tree in Section 5.2, we can *explicitly describe* an element outside the range of $C^{\{0,1\}^n}$. Unfortunately $|\{0,1\}^n|$ is prohibitively large and so we cannot explicitly expand the tree and perform the reduction procedure described in the proof of Theorem 7. Nonetheless we can consider the function C^S, for certain *succinctly describable* S which are subtrees of $\{0,1\}^n$, and reason about the functions $f : S \to \{0,1\}^n$ inside and outside of $\text{range}(C^S)$. This line of analysis is applied in an ingenious way in both of [10,35] to isolate certain *special solutions* to an Avoid instance which have unique certificates of correctness. The original method in [10] used a so called *win-win* analysis over a family of Avoid instances of various input lengths, and is shown to work correctly for *some instance* in this family. Li was able to modify (and significantly simplify) this original construction so that it works unconditionally on all Avoid instances.

5.4 Algorithmic Methods

As we saw in Section 3.1, algorithms for C-Avoid imply C-circuit lower bounds for exponential time classes, however for classes C below $\mathsf{P/poly}$ the reverse implication is not known. It is then natural to ask whether, for the circuit classes C which are unconditionally known not to contain large uniform classes such as $\mathsf{EXP}, \mathsf{NEXP}, \mathsf{EXP}^{\mathsf{NP}}$, we can strengthen these lower bounds to achieve range avoidance algorithms. Two important special cases are $C = \mathsf{AC}^0$, where lower bounds for very simple functions (parity) have been known for decades [2, 16], and $C = \mathsf{ACC}^0$, where lower bounds inside NEXP were more recently obtained by Williams [45].

Williams' breakthrough result was established by general method laid out in [45] and denoted "the Algorithmic Method." The main theorem of [45] says that for any "sufficiently nice" class of circuits C, any algorithm for testing the satisfiability of C-circuits which has sufficiently nontrivial advantage over brute-force search implies that NEXP does not have polynomial size C-circuits. Williams' lower bound for ACC^0 is then obtained by designing the necessary satisfiability algorithms for ACC^0. In a series of works [9,41], Williams' algorithmic method is adapted to the context of Avoid, where results of the following form are given: if a certain computational task associated with C-circuits has a slightly nontrivial algo-

rithm, then C-Avoid (with a suitable stretch parameter) collapses into polynomial time (perhaps with an NP-oracle).

The results of [9, 41] are quite involved and beyond the scope of this survey; the main technical ingredient in both papers is the design of highly specialized PCPs. We mention only a major result of the second paper, which achieves the goal of extending the best known lower bounds techniques for ACC^0 to achieve matching ACC^0-Avoid algorithms:

Theorem 9 ([9]). ACC^0-Avoid *with stretch* $n \mapsto 2^{\log^{\omega(1)} n}$ *is solvable in* $\mathsf{FP}^{\mathsf{NP}}$. *More specifically, for each* $d, m \in \mathbb{N}$ *there exists* $c \in \mathbb{N}$ *and an* $\mathsf{FP}^{\mathsf{NP}}$ *algorithm* $\mathcal{A}$ *which solves* Avoid *on instances* $C : \{0,1\}^n \to \{0,1\}^m$ *in which each output is computed by a depth* d $\mathrm{poly}(m)$ *size circuit over the basis* $\{\vee, \wedge, \mathrm{MOD}_m\}$, *provided* $m \geq 2^{\log^c n}$.

It should be noted that this theorem is "optimal" in terms of the stretch in the sense that it recovers the best known quantitative lower bounds against ACC^0: any Avoid algorithm of the above kind immediately implies that E^{NP} requires $2^{n^{\Omega(1)}}$ size ACC^0 circuits by Theorem 5, and obtaining an $\mathsf{FP}^{\mathsf{NP}}$ algorithm in a stronger stretch regime would imply a better lower bound then is currently known.

5.5 NC^0 Avoid With Large Stretch

We mention here briefly the only other case of Avoid for which nontrivial algorithms are known:

Theorem 10 ([18, 21])**.**

1. NC^0_2-Avoid *has a polynomial time algorithm.*
2. NC^0_k-Avoid *has a polynomial time algorithm in the stretch regime* $n \mapsto n^{k-1}/\log n$.

Unlike the previous upper bounds for Avoid, these algorithms are based on direct combinatorial properties of the special Avoid instances. The first is from [21] and is a rather direct application of known polynomial time algorithms for 2-Sat. The second result was first proven in a weaker form in [21], utilizing the theory of k-wise independent distributions. This original algorithm required a stretch regime $n \mapsto n^{k-1}$ and only worked with an NP-oracle; both aspects were improved in [18], which uses an iterative algorithm to maintain a prefix $(y_1, \ldots, y_i) \in \{0,1\}^i$, so that many of its extensions to a complete string $y \in \{0,1\}^m$ lie outside the range of the Avoid instance C.

6 Lower Bounds for Avoid

In this section we cover some results which indicate the hardness of solving Avoid in certain complexity classes. Of course any unconditional lower bound for polynomial time algorithms would imply $\mathsf{P} \neq \mathsf{NP}$, hence we can only hope for results here which are conditional or which hold against severely restricted models of computation.

We have seen in Section 3 that many difficult and well-studied explicit construction problems reduce to Avoid. Certainly this qualifies as evidence of the *mathematical difficulty* of discovering improved algorithms for Avoid, however it would only qualify as evidence of *computational hardness* if we had reasons to believe that some of these problems were truly outside of polynomial time. While the situation is certainly unclear, the prevailing wisdom is that most (if not all) of the explicit construction problems discussed in Section 3 ultimately lie in P. The intuition seems to be that, once a suitable mathematical understanding of the relevant pseudorandom concept (e.g. circuit complexity or matrix rigidity) is obtained, explicit constructions will follow; the only barrier is our (humanity's) mathematical immaturity. Hence to get "evidence" of the computational hardness of Avoid, it seems we need to look for reductions to Avoid from problems of a different flavor.

Even if the intuition in the previous paragraph is unconvincing, there is a second reason why reductions from explicit construction problems fail to resolve the hardness of Avoid: since explicit construction problems have unary input, they are all solvable in $\mathsf{FP/poly}$ by hardcoding a solution of each length as advice. For this reason, the results in Section 3 seem to have no bearing whatsoever on the question of whether Avoid is solvable by polynomial-size circuits.

6.1 Cryptographic Hardness

The first evidence of strong computational hardness for Avoid, namely Avoid $\notin \mathsf{FP/poly}$, was obtained in [23] using a strong cryptographic assumption. This assumption, known as *indistinguishability obfuscation* or $\mathcal{IO}$ for short, posits the existence of a universal compiler which can take any program (circuit) C and produce an equivalent one $\mathcal{IO}(C)$ which has the same functionality as the original, but which hides all implementation details: for any pair C, C' with the same functionality, $\mathcal{IO}(C)$ and $\mathcal{IO}(C')$ should look the same to a computationally-bounded observer. This concept was originally introduced in [5] (where a more formal definition can be found) and has received considerable attention in the years since. In [23], it is shown that assuming the existence of $\mathcal{IO}$ we can obtain conditional hardness for Avoid:

Theorem 11 ([23]). *Assume that there exists $\mathcal{IO}$ scheme secure against subexponential time nonuniform algorithms and that* $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$. *Then* Avoid $\notin \mathsf{FP/poly}$.

For a considerable span of time after the original definition of $\mathcal{IO}$ in [5] there was little public consensus on whether or not secure $\mathcal{IO}$ was a plausible assumption. This changed after a recent breakthrough of Jain, Lin and Sahai [27], who demonstrated how to construct $\mathcal{IO}$ from a few concrete cryptographic hardness assumptions which have each withstood years of attacks and are considered highly plausible. As a result, the existence of $\mathcal{IO}$ is now considered by many as "likely true" and hence the results of [23] give a rather convincing argument that Avoid is not solvable by polynomial size circuits.

A second line of work by [11] obtains a similar kind of cryptographic hardness for Avoid, which moreover applies to C-Avoid for very simple classes of circuits C. In particular, assuming the hardness of certain nonstandard variants of LWE and LPN, [11] obtain hardness for TC^0-Avoid, and hence for NC^0_4-Avoid by Theorem 6. The cryptographic assumptions used in [11] are introduced in the paper itself and have an unclear relationship with the more standard variants of LWE and LPN studied widely in the literature.

6.2 Hardness in the Black Box Model

A second kind of "hardness" result for Avoid is known in a restricted computational model, called the "black box" or "decision tree" model. Here it can be shown that no $\mathsf{FP}^{\mathsf{NP}}$ algorithm can solve Avoid given that it treats the instance $C : \{0,1\}^n \to \{0,1\}^m$ as a black box and ignores its implementation as a boolean circuit. By standard techniques, this can be used to build an oracle relative to which Avoid is not in $\mathsf{FP}^{\mathsf{NP}}$ (see [43] for an explanation). This lower bound was essentially proven by Wilson [44], who gave an oracle relative to which $\mathsf{E}^{\mathsf{NP}} \in \mathsf{P/poly}$; the interpretation of this result in terms of Avoid was first noted by [43]. Since the proof is quite simple and Wilson states it in a rather different terminology, we reproduce it here in the language of range avoidance. We first need the following definition of a black-box $\mathsf{FP}^{\mathsf{NP}}$ query algorithm:

Definition 9. *Let N, M be given, $M \geq 2N$ and both powers of two. Each assignmet $\alpha \in \{0,1\}^{N \log M}$ can be interpreted uniquely as defining a function $f_\alpha : [N] \to [M]$, i.e. for $x \in [N]$ and $j \in [\log M]$ we interpret $\alpha_{x,j}$ as the j^{th} bit of $f_\alpha(x)$.*

Let Q be a query algorithm which, given an assignment α, may choose at each step a DNF D over the variables $\{\alpha_i \mid i \in N \log M\}$, and receive its value. After all queries have been made, Q must output a value $y \in [M]$. We say that Q solves the Avoid *problem if for all assignments α, $Q(\alpha)$ outputs a value $y \notin \mathrm{range}(f_\alpha)$. The "width complexity" of Q is the maximum width of a DNF it queries, and the*

"query complexity" is the maximum number of queries it makes before outputting an assignment.

If we imagine $N = \{0,1\}^n$ and $M = \{0,1\}^m$ for some $m > n$, then each $f_\alpha : [N] \to [M]$ corresponds to a potential "oracle instance" of Avoid. Now, from the perspective of an $\mathsf{FP}^{\mathsf{NP}}$ algorithm, each NP query may make a nondeterministic guess, followed by a polynomial time verification. This verification can only read $\mathrm{poly}(n) = \mathrm{poly}\log(N)$ bits of the oracle assignment α. The NP query outputs 1 if at least one of these verifications pass. Hence the value of this query, as a function of the oracle assignment, corresponds to a DNF of width $\mathrm{poly}\log(N)$ over the variables of α, and since the $\mathsf{FP}^{\mathsf{NP}}$ algorithm must run in $\mathrm{poly}(n) = \mathrm{poly}\log(N)$ time, it can only make $\mathrm{poly}\log(N)$ many such queries to the oracle before terminating. Hence if we can prove that any query algorithm Q of the above form must have either its query complexity or width complexity exceeding $\mathrm{poly}\log(N)$ then this will show that no such $\mathsf{FP}^{\mathsf{NP}}$ algorithm can work relative to every oracle.

Theorem 12 ([44])**.** *If Q is as above, has query complexity q and width complexity w and solves the $[N] \to [M]$* Avoid *problem, then $qw \geq N$. Hence* Avoid $\notin \mathsf{FP}^{\mathsf{NP}}$ *relative to some oracle.*

Proof. We will design an adversary which can answer enough DNF queries so that Q is forced to output an answer, but does not give enough information about the purported assignment α so as to fix any y to be outside the range of f_α. At each step we maintain a partial assignment $\rho : [N] \to [M]$, so that after t steps we have $|\mathrm{domain}(\rho)| \leq tw$. When a DNF $D = \bigvee_j \tau_j$ is presented, we search for a term τ_j such that there exists a total assignment consistent with both τ_j and the current partial assignment ρ. If so, then since the width of τ_j is at most w, we may extend ρ to ρ' by defining it on at most w additional inputs so that already $\tau_j(\alpha) = 1$ for all total assignments α extending ρ'; the adversary then answers "true" for the DNF query D. If it cannot find such an extension the adversary answers "false;" in this case we know that for any extension of ρ we will always falsify all terms in D. Now, if $qw < N$ then the adversary may continue this process q times and reach a termination point of Q, after which Q must output some candidate solution y. Since at this point $|\rho| \leq qw < N$, there is some $x \in [N]$ so that $\rho(x)$ is undefined; the adversary then sets $\rho(x) = y$ and completes ρ everywhere else to a total assignment α. At this point all the query responses supplied by the adversary are true of α, however $y \in \mathrm{range}(f_\alpha)$, hence $Q(\alpha)$ has the wrong behavior. □

In [31] a much stronger oracle separation is shown for a variant of the Avoid problem, called Strong-Avoid, in which we are given a boolean circuit of the form $C : \{0,1\}^n \setminus \{0^n\} \to \{0,1\}^n$, and must find an n-bit string outside its range

(in other words we are excluding 0^n from consideration in the domain). It is shown in [31] that (relative to an oracle), Strong-Avoid is not solvable in $\mathsf{FP}_{\|}^{\Sigma_2^{\mathsf{P}}}$, the class of search problems solvable in polynomial time with *non-adaptive* access to a Σ_2^{P} oracle. This implies as a special case that the (relativizing) upper bound Avoid $\in \mathsf{psZPP}^{\mathsf{NP}}$ mentioned in Section 5.3 cannot be applied to the more general problem Strong-Avoid. The proof of this separation involves techniques developed to analyze bounded-depth proof systems and AC^0 circuits, and in particular relies on a specialized variant of the switching lemma.

7 Lossy Code

So far we have focused exclusively on the general Avoid problem, which is a total search problem lying in the second level of the polynomial hierarchy. In this section we will explore a "younger cousin" of Avoid which we refer to as the Lossy Code Problem or Lossy for short. This problem lies one level down in the polynomial hierarchy and is a total search problem in the heavily-studied class TFNP. Similar to Avoid, obtaining better algorithms for Lossy can be viewed as a derandomization problem. However, the motivation to study it is admittedly weaker than in the case of Avoid: while obtaining derandomized algorithms for Avoid would have tremendous consequences for circuit complexity and derandomization, we will see only one unconditional interesting consequence that follows from a derandomized algorithm for Lossy. On the other hand, since Lossy is an inherently easier problem than Avoid, and indeed easier even than derandomizing BPP and ZPP[4], it is possible that we can obtain positive results for this problem which are too difficult to obtain in the case of Avoid.

7.1 Definition and Basic Inclusions

We start with the formal definition of the problem:

Definition 10 (Lossy code, [30])**.** *Lossy Code, denoted* Lossy*, is the following problem: given circuits* $E : \{0,1\}^n \to \{0,1\}^{n-1}$*,* $D : \{0,1\}^{n-1} \to \{0,1\}^n$*, find an input* $x \in \{0,1\}^n$ *such that* $D(E(x)) \neq x$*.*

It is useful to think of E as an encoding function which compresses an n-bit string down to a shorter string, and D as a decoding function which attempts to recover the original string from its encoding. Here the pigeonhole principle tells us that E, D cannot define a true lossless compression scheme: there must be some x

[4]Technically speaking this only holds if we define BPP, ZPP as promise classes, but we will not concern ourselves with this distinction here.

which is not recovered form its encoding. The basic upper bounds for this problem are as follows:

Observation 3. Lossy $\in \mathsf{TFNP} \cap \mathsf{FZPP}$.

Proof. Lossy is an NP search problem since the validity of a solution $D(E(x)) = x$ can be checked in polynomial time. It is total by the pigeonhole principle. A random candidate solution is correct with probability $\geq \frac{1}{2}$. □

As per Lemma 1, the inclusion in FZPP tells us the following:

Corollary 8. *Under standard derandomization assumptions, namely that* E *requires* $2^{\Omega(n)}$ *size circuits,* Lossy *is solvable in deterministic polynomial time.*

Hence, unlike Avoid where "conventional wisdom" at best gives an upper bound of $\mathsf{FP}^{\mathsf{NP}}$, for Lossy it tells us the problem should completely collapse into polynomial time. We can also easily observe that in a more direct sense, Avoid is at least as hard as Lossy:

Observation 4. *Lossy Code is polynomial time reducible to Avoid.*

Proof. Let (E, D) be a Lossy Code instance. Note that D by itself can be considered as an instance of Avoid. If $x \notin \text{range}(D)$ is an Avoid solution for D, then clearly $D(E(x)) \neq x$ and so it is also a solution for the Lossy Code instance. □

Perhaps more enlightening is the following more informal connection to Avoid. Say that $C : \{0,1\}^{n-1} \to \{0,1\}^n$ is an arbitrary Avoid instance, and consider the lexicographical-pseudoinverse $C^{-1} : \{0,1\}^n \to \{0,1\}^{n-1}$ defined as follows: $C^{-1}(x)$ outputs the lexicographically first preimage of x if one exists, else it outputs 0^n. Then any solution to the Lossy "instance" (C^{-1}, C) is a solution to Avoid. Clearly (C^{-1}, C) is not actually a valid instance of Lossy, since given C we cannot efficiently construct a boolean circuit computing C^{-1} (indeed assuming $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ there are some cases in which no small circuit for C^{-1} exists). However, if we have access to an NP-oracle, we can effectively use it to evaluate C^{-1}; hence an $\mathsf{FP}^{\mathsf{NP}}$ algorithm can, in some cases, treat an Avoid instance as an implicitly defined Lossy instance. In this sense we may view Lossy as a kind of "polynomial-time model" of the more general Avoid problem, where certain techniques that worked for Avoid only in the presence of an NP-oracle are now available in polynomial time for Lossy. For example:

Lemma 8. *For any constant* $c \in \mathbb{N}$*, a general instance of* Lossy *is polynomial time reducible to an instance* $E : \{0,1\}^{n^c} \to \{0,1\}^n, D : \{0,1\}^n \to \{0,1\}^{n^c}$.

Proof. Identical to Lemma 6, replacing the NP-oracle with E. □

Similar to the direct reduction of Lossy to Avoid, there is also an immediate reduction of Lossy to the more standard pigeonhole problems in TFNP:

Definition 11 ([26])**.** WeakPigeon *is the following search problem: given a circuit* $E : \{0,1\}^n \to \{0,1\}^{n-1}$*, find a pair* $x \neq x' \in \{0,1\}^n$ *such that* $E(x) = E(x')$*.*

WeakPigeon is the complete problem for the TFNP class PWPP, which itself is a subclass of the more famous PPP. We can then observe:

Observation 5. Lossy *is polynomial time reducible to* WeakPigeon*.*

Proof. Let (E, D) be an instance of Lossy. Interpret E as an instance of WeakPigeon and let $x \neq x'$ be a solution. Then either x or x' must be a Lossy solution since for $y = E(x) = E(x')$ it cannot be that $D(y) = x$ and $D(y) = x'$. □

7.2 Problems Reducible to Lossy Code

The primary motivation to study Avoid stems from the plethora of important problems which reduce to it. In contrast, the set of known reductions to Lossy is rather sparse. The only well-studied problem for which an *unconditional* reduction to Lossy is known is the derandomization of *Catalytic Logspace*, a complexity class defined in [4]:

Definition 12 ([4])**.** *A language L is in Catalytic Logspace, denoted* CL*, if it is computable by a Turing machine of the following form. The machine has three tapes:*

1. *Input tape: this tape has length n and is read-only.*
2. *Work tape: this tape has length* $O(\log n)$ *and is read-write.*
3. *Catalytic tape: this tape has length* $\mathrm{poly}(n)$ *and is read-write.*

At the beginning of the computation, the input x is written on the input tape, the work tape is initialized to all-zeroes, and the catalytic tape is intialized to an arbitrary value z. The machine must have the following behavior on all such initial configurations (x, z)*:*

1. *At the end of the computation, the machine successfully decides whether or not* $x \in L$*.*
2. *At the end of the computation, the catalytic tape is returned to its original state z.*

It was shown in [4] that $\mathsf{CL} \subseteq \mathsf{ZPP}$; hence under suitable derandomization assumptions $\mathsf{CL} \subseteq \mathsf{P}$, however it has remained an open problem to prove this unconditionally. The following upper bound, improving the bound $\mathsf{CL} \subseteq \mathsf{ZPP}$, is established in [12]

Theorem 13 ([12])**.** *Any language $L \in \mathsf{CL}$ is (deterministic) polynomial-time reducible to* Lossy

Two more reductions to Lossy are shown in [30]; unlike the previous they only give a reduction to a nonstandard variant of Lossy

Definition 13. *Let $A, B \subseteq \{0,1\}^*$ be languages. The problem* $\text{Lossy}^{A,B}$ *is defined as follows: given an A-oracle circuit $E : \{0,1\}^n \to \{0,1\}^{n-1}$ and a B-oracle circuit $D : \{0,1\}^{n-1} \to \{0,1\}^n$, solve the* Lossy *problem for (E, D): find a string x so that $D(E(x)) \neq x$. We abbreviate* $\text{Lossy}^A := \text{Lossy}^{A,\emptyset}$.

As hinted above, we can reduce a general Avoid instance C to $\text{Lossy}^{\mathsf{NP}}$ by letting $D = C$ and $E = C^{-1}$ (the lexicographical pseudoinverse); since we can always ignore the encoder and solve Avoid for the decoder, we see that the problems Avoid and $\text{Lossy}^{\mathsf{NP}}$ are completely equivalent. Hence any of the important explicit construction problems covered in Section 3 can be reduced directly to $\text{Lossy}^{\mathsf{NP}}$ and this reduction tells us nothing new. In [30], two examples are given where an interesting explicit construction problem is reduced to Lossy^A for a language $A \in \mathsf{NP}$ which is not known to be NP-hard, or to $\text{Lossy}^{A,B}$ where A, B lie in $\mathsf{NP} \cap \mathsf{coNP}$. Such a reduction tells us more than a generic reduction to $\text{Avoid} = \text{Lossy}^{\mathsf{NP}}$, however it is still much weaker than a full reduction to Lossy. The most interesting of these reductions is the following:

Theorem 14 ([30])**.** *Consider the problem: given 1^n, produce a prime number $p > 2^n$. This problem is reducible to* $\text{Lossy}^{\mathsf{FAC},\mathsf{FAC}}$*, where* FAC *is the integer-factorization problem. In particular if* $\mathsf{FAC} \in \mathsf{P}$ *then prime construction is reducible to* Lossy.

This theorem follows from a careful analysis of the result of [39] in Bounded Arithmetic mentioned in the introduction, who showed that the existence of infinitely many primes can be proved in $I\Delta_0$ if we add as an axiom the weak pigeonhole schema for Δ_0 formulas. The reduction in Theorem 14 produces a prime in the range $[2^n, 2^{32n}]$.

7.3 Conditional and Unconditional Upper Bounds for Lossy Code

The main positive result in [30] is a conditional derandomization for Lossy under an unusual hardness assumption for uniform deterministic algorithms. This deran-

domization result does not quite apply to the generic search problem Lossy, but rather to *uniform instance sequences* of the search problem:

Definition 14. *We say that a sequence of strings $X = (x_n)_{n\in\mathbb{N}}$ is a "uniform sequence" if there is a fixed deterministic Turing machine M_X which prints x_n in* poly(n) *time given* 1^n.

Let $\mathcal{S} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a search problem. We say that a search problem $\mathcal{S}$ is polynomial time solvable on uniform instances if for every uniform sequence $X = (x_n)_{n\in\mathbb{N}}$, there is a second uniform sequence $Y = (y_n)_{n\in\mathbb{N}}$ so that for all n, $(x_n, y_n) \in \mathcal{S}$, i.e. y_n is a solution to instance x_n of the search problem $\mathcal{S}$.

The motivation for this definition is as follows: for explicit construction problems EC_Π, there is only one relevant "instance" of each length that we care about solving. The instance is the string 1^n, and the goal is to produce a string $y_n \in \Pi_n$ given this "instance." This means that solving the explicit construction problem EC_Π in polynomial time is equivalent to solving it on uniform instance sequences. Thus for the search problems Lossy and Avoid, which we primarily care about because of the explicit construction problems that reduce to them, we would already be satisfied with novel algorithms which only worked on uniform input sequences.

The main result in [30] says that, assuming certain *uniform* time-space tradeoffs for Turing machines, we can solve Lossy on all uniform sequences in polynomial time. The specific tradeoff assumption needed is the following:

Hypothesis 1. *There is a constant $\epsilon > 0$ and a language L so that the following holds:*

1. *L is decidable by a Turing machine running in time 2^n.*
2. *Any Turing machine M running in time $2^{(1+\epsilon)n}$ and space $2^{\epsilon n}$ fails to decide L on some n-bit input, for all sufficiently large n.*

We then have:

Theorem 15. *[30] Assuming Hypothesis 1,* Lossy *can be solved in deterministic polynomial time on all uniform instances.*

Recall (Corollary 8) that standard derandomization assumptions already collapse Lossy into polynomial time, with no caveats about uniform input sequences. These standard derandomization assumptions posit the existence of languages of bounded uniform complexity which are hard for boolean circuits, i.e. *nonuniform algorithms*. The important distinction between these results and the above theorem is that Theorem 15 uses a hardness assumption against *uniform* algorithms, i.e. Turing machines with no advice.

The proof of Theorem 15 utilizes the tree-construction from Section 5.1. The idea is roughly the following: say there is a uniform instance of Lossy which is hard for all polynomial time algorithms. This defines a compression scheme, computable uniformly in polynomial time, whereby any n-bit string may be compressed to $n - 1$ bits and then recovered later; while the compression must fail for some strings, no polynomial time observer will be able to construct an example of such a string. Using a modification of the tree construction, this scheme can be iterated to construct a "virtual RAM," whereby a large memory of size s can be compressed to s^ϵ bits, and each bit can be read/written at cost s^ϵ; any read or write operation causing the virtual RAM to fail will indicate a failure point of the compression scheme. We can then use this to simulate any T-time computation in T^ϵ space, with an overhead of T^ϵ time to simulate each step of the original computation. If there was any uniform machine where the low-space simulation failed, a poly(T) time algorithm could use it to to find a string on which the compression procedure failed, and hence solve the uniform instance of Lossy.

Finally, we mention one additional positive result which is relevant to solving uniform instances of Lossy. This stems from the powerful result of [13] mentioned in Section 2.1, which gives a polynomial time pseudodeterministic algorithm for EC_Π whenever Π is a dense property recognizable in P. Observe that if $(x_n)_{n\in\mathbb{N}}$ is a uniform instance of Lossy, the set of y which form a solution for the instance x_n is a dense, polynomial time recognizable set. As a direct corollary we get:

Corollary 9. *For every uniform sequence of* Lossy *instances $X = (x_n)_{n\in\mathbb{N}}$, there is a solution sequence $Y = (y_n)_{n\in\mathbb{N}}$ and a polynomial time randomized algorithm which outputs y_n with high probability for infinitely many n.*

8 Open Problems

In this section we highlight some important open problems related to the topics discussed in this survey.

8.1 Better Stretching Reductions?

The first problem we highlight is to determine whether the stretching reduction in Section 4 is optimal in terms of depth:

Problem 1. *Is there an $\mathsf{FP}^{\mathsf{NP}}$ reduction from stretch $n \mapsto \alpha n$ to $m \mapsto \beta m$ with depth complexity significantly better than $\log_\alpha \beta$?*

Along the same lines we may ask:

Problem 2. *Can we reduce C-Avoid to constructing C-hard truth tables for any class C smaller than* $\mathsf{P/poly}$*?*

These two problems seem closely related: the reduction of Avoid to hard truth tables is essentially an extension of the stretching reduction, and the reason that it fails for depth-restricted circuit classes is precisely because of the depth cost inherent in the repeated composition of the avoid instance.

Perhaps the most exciting possibility is that the answer to one or both of these questions is positive, and that it can be proven unconditionally by exhibiting the reduction. Recall that the reduction from Avoid to hard truth tables allows us to prove a kind a hardness amplification result for the circuit complexity of E^{NP}: if this class requires $2^{\Omega(n)}$ size circuits then it also requires $2^n/n$ sized circuits. It can be shown using the same argument that if $\mathsf{EXP}^{\mathsf{NP}}$ requires $2^{n^{\Omega(1)}}$ sized circuits, then it also requires $2^n/n$ sized-circuits. If such an amplification result could be shown for the case $C = \mathsf{AC}^0$ using a superior stretching reduction, and we could boost the known $2^{n^{\Omega(1)}}$ AC^0 lower bounds for parity to $2^{\Omega(n)}$ lower bounds for a function in $\mathsf{EXP}^{\mathsf{NP}}$, this would imply the breakthrough $\mathsf{EXP}^{\mathsf{NP}} \not\subseteq \mathsf{NC}^1$ (this observation was made in [41]).

However it is equally natural to conjecture that the answer is negative, in which case we cannot hope to refute it unconditionally without proving $\mathsf{P} \neq \mathsf{NP}$. In this case, it would be interesting to show a negative result in a restricted *black box* model of reducibility: note that the stretching reduction in Lemma 6 treats the Avoid instance in a completely black box way, and the *depth* of the reduction can be measured in a black box sense with no reference to boolean circuit depth. The analogous question for PRG stretching reductions has been asked before (in a suitable formalization of the black box model), and remains almost completely unsolved [36].

8.2 Stronger Hardness for Avoid?

The second problem we highlight is to better understand the computational hardness of Avoid. The results of [11, 23] highlighted in Section 6 tell us that Avoid is hard under very strong cryptographic assumptions. It is still unclear whether we can base the hardness of Avoid on something more standard. In particular the following remains completely open:

Problem 3. *Is* Avoid NP*-hard under polynomial time Turing reductions? Would such* NP*-hardness contradict any standard complexity theoretic assumptions? Is there an oracle relative to which* Avoid $\in \mathsf{FP}$ *but* $\mathsf{P} \neq \mathsf{NP}$*?*

Note that if severe restrictions are placed on the reduction, it is possible to show that Avoid cannot be NP-hard unless $\mathsf{NP} = \mathsf{BPP}$ [23]. In particular if there is

a reduction from SAT making a small number of queries to AVOID instances with stretch $n \mapsto n^2$, then we may replace the oracle response with a random string, and the behavior of the reduction will still be correct with high probability. However, if a reduction adaptively queries poly(n) many instances each with stretch $n \mapsto n + 1$ and is only guaranteed to be correct when supplied with correct answers to *all* queries, replacing the oracle with random bits appears useless.

8.3 Lossy Code: More Reductions and Better Upper Bounds?

Lastly we discuss some directions for future work on the less explored problem LOSSY. The first direction is to try to demonstrate the *power* of the problem LOSSY by exhibiting more reductions from explicit construction problems to it. As discussed at the end of Section 7.3, any explicit construction problem reducible to LOSSY must also be reducible to the explicit construction problem EC_Π for a dense language $\Pi \in \mathsf{P}$. Currently the only interesting example of such a problem we are aware of is the construction of prime numbers, where a reduction to LOSSY is only known to hold under the (unlikely) assumption that integer factoring is in P. It would add significant motivation to the study of LOSSY if this assumption could be removed:

Problem 4. *Can the problem of constructing a prime $p > 2^n$ be reduced in* poly(n) *time to* LOSSY *unconditionally? More generally, are there any other interesting unsolved explicit construction problems which can be reduced unconditionally to* LOSSY*?*

A primary motivation for studying LOSSY is that it is essentially the easiest *generic derandomization problem* we are aware of, and thus we may hope to get unconditional upper bounds for it before we are able to achieve more lofty goals such as AVOID $\in \mathsf{FP}^{\mathsf{NP}}$ or $\mathsf{BPP} = \mathsf{P}$.

Problem 5. *Does Lossy admit nontrivial upper bounds similar to those given for* AVOID *in [10, 35], other than the infinitely-often pseudodeterministic algorithm on uniform instances given by [13]?*

For this problem, we note an obstacle which is that LOSSY is a TFNP search problem, and it is known that essentially all nontrivial algorithmic subclasses of TFNP collapse to P in the black box model, while LOSSY does not. This presents an obstacle for a result similar to [35] (which relativizes) to apply to LOSSY; see the introduction of [31] for a discussion of this issue. However, the pseudodeterministic construction in [13] is also non-relativizing in this same sense (see [20]). For this reason it seems possible that something could be done along the lines of [13], which uses specific properties of uniform instances of LOSSY that do not hold for general dense P explicit construction problems. For example:

Conjecture 1. *Let $\Pi \in \{0,1\}^*$ be an explicit construction problem reducible to Lossy Code. Then there is an* NP *language L so that for infinitely many n, $L_n \subseteq \Pi_n$ and $|L_n| = 1$.*

Of course under strong enough derandomization assumptions we expect significantly stronger assumptions to hold; the question is whether a result of this form, which is not immediately implied by [13], could be established unconditionally using specific properties of the search problem Lossy.

References

[1] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc0. *SIAM Journal on Computing*, 36(4):845–888, 2006.

[2] Miklós Ajtai. $\sum^1_1$-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983.

[3] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Ann. of Math*, 2:781–793, 2002.

[4] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC 2014, pages 857–866, 2014.

[5] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), may 2012.

[6] Jin-Yi Cai. S_2^P is subset of ZPP^{NP}. *Journal of Computer and System Sciences*, 73(1):25–35, 2007.

[7] Eshan Chattopadhyay, Jesse Goodman, and Jyun-Jie Liao. Affine extractors for almost logarithmic entropy. In *Proceedings of the IEEE 62nd Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 622–633, 2022.

[8] Eshan Chattopadhyay. Guest column: A recipe for constructing two-source extractors. *SIGACT News*, 51(2):38–57, June 2020.

[9] Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1058–1066, 2023.

[10] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 1990–1999, 2024.

[11] Yilei Chen and Jiatu Li. Hardness of range avoidance and remote point for restricted circuits via cryptography. Cryptology ePrint Archive, Paper 2023/1894, 2023. https://eprint.iacr.org/2023/1894.

[12] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. *Preprint*, 2024.

[13] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *Proceedings of the IEEE 64th Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 1261–1270, nov 2023.

[14] Zeev Dvir. Extractors for varieties. In *2009 24th Annual IEEE Conference on Computational Complexity*, CCC 2009, pages 102–113, 2009.

[15] Paul Erdös. Some remarks on the theory of graphs. *Bulletin of the American Mathematical Society*, 53:292–294, 1947.

[16] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984.

[17] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[18] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range Avoidance for Constant Depth Circuits: Hardness and Algorithms. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2023)*, volume 275 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 65:1–65:18, 2023.

[19] Alexander Golovnev, Rahul Ilango, R. Impagliazzo, Valentine Kabanets, A. Kolokolova, and A. Tal. AC0[p] lower bounds against MCSP via the coin problem. *Electron. Colloquium Comput. Complex.*, 26:18, 2019.

[20] Shafi Goldwasser, Russell Impagliazzo, Toniann Pitassi, and Rahul Santhanam. On the pseudo-deterministic query complexity of NP search problems. In *Proceedings of the 36th Computational Complexity Conference*, CCC 2021, 2021.

[21] Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range Avoidance for Low-Depth Circuits and Connections to Pseudorandomness. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*, volume 245 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:21, 2022.

[22] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2006*, pages 672–683, 2006.

[23] Rahul Ilango, Jiatu Li, and R. Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1076–1089, 2023.

[24] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC 1997, pages 220–229, 1997.

[25] Emil Jeřábek. Dual weak pigeonhole principle, boolean complexity, and derandomization. *Annals of Pure and Applied Logic*, 129(1):1–37, 2004.

[26] Emil Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016.

[27] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 60–73, 2021.

[28] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total Functions in the Polynomial Hierarchy. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:18, 2021.

[29] Oliver Korten. The hardest explicit construction. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 433–444. IEEE, 2021.

[30] Oliver Korten. Derandomization from time-space tradeoffs. In *Proceedings of the 37th Computational Complexity Conference*, CCC 2022, 2022.

[31] Oliver Korten and Toniann Pitassi. Strong vs. weak range avoidance and the linear ordering principle. In *Proceedings of the 65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024*, pages 1388–1407, 2024.

[32] Jan Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Encyclopedia of Mathematics and its Applications. 1995.

[33] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[34] Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 1144–1156, 2017.

[35] Zeyong Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 2000–2007, 2024.

[36] Eric Miles and Emanuele Viola. On the complexity of non-adaptively increasing the stretch of pseudorandom generators. In *Theory of Cryptography*, pages 522–539, 2011.

[37] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[38] Mihai Patrascu. Succincter. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008*, pages 305–313, 2008.

[39] Jeff B. Paris, A. J. Wilkie, and Alan R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *J. Symb. Log.*, 53:1235–1244, 1988.

[40] Alexander Razborov. Bounded-depth formula over {AND,XOR} and some combinatorial problems (russian). *Problems of Cybernetics. Complexity Theory and Applied Mathematical Logic*, pages 149–166, 1988.

[41] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *Proceedings of the IEEE 63rd Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 640–650, 2022.

[42] Claude E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.

[43] Nikhil Vyas and Ryan Williams. On Oracles and Algorithmic Methods for Proving Lower Bounds. In *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 99:1–99:26, 2023.

[44] Christopher B. Wilson. Relativized circuit complexity. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science, FOCS 1983*, pages 329–334, 1983.

[45] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC 2010, pages 231–240, 2010.