

Total Search Problems in ZPP

Noah Fleming
Lund & Columbia

Stefan Grosser
McGill

Siddhartha Jain
UT Austin

Jiawei Li
UT Austin

Hanlin Ren
IAS

Morgan Shirley
Lund

Weiqiang Yuan
EPFL

December 1, 2025

Abstract

We initiate a systematic study of TFZPP, the class of total NP search problems solvable by polynomial time randomized algorithms. TFZPP contains a variety of important search problems such as BERTRAND-CHEBYSHEV (finding a prime between N and $2N$), refuter problems for many circuit lower bounds, and LOSSY-CODE. The LOSSY-CODE problem has found prominence due to its fundamental connections to derandomization, catalytic computing, and the metamathematics of complexity theory, among other areas.

While TFZPP collapses to FP under standard derandomization assumptions in the white-box setting, we are able to separate TFZPP from the major TFNP subclasses in the black-box setting. In fact, we are able to separate it from every uniform TFNP class assuming that NP is not in quasi-polynomial time. To do so, we extend the connection between proof complexity and black-box TFNP to randomized proof systems and randomized reductions.

Next, we turn to developing a taxonomy of TFZPP problems. We highlight a problem called NEPHEW, originating from an infinity axiom in set theory. We show that NEPHEW is in $PWPP \cap TFZPP$ and conjecture that it is not reducible to LOSSY-CODE. Intriguingly, except for some artificial examples, most other black-box TFZPP problems that we are aware of reduce to LOSSY-CODE:

- We define a problem called EMPTY-CHILD capturing finding a leaf in a rooted (binary) tree, and show that this problem is *equivalent* to LOSSY-CODE. We also show that a variant of EMPTY-CHILD with “heights” is complete for the intersection of SOPL and LOSSY-CODE.
- We strengthen LOSSY-CODE with several combinatorial inequalities such as the AM-GM inequality. Somewhat surprisingly, we show the resulting new problems are still reducible to LOSSY-CODE. A technical highlight of this result is that they are proved by *formalizations in bounded arithmetic*, specifically in Jeřábek’s theory APC_1 (JSL 2007).
- Finally, we show that the DENSE-LINEAR-ORDERING problem reduces to LOSSY-CODE.

Contents

Contents	2
1 Introduction	3
2 Preliminaries	9
2.1 Basics of TFNP	9
2.2 TFZPP	10
2.3 LOSSY-CODE	12
3 Randomized Proof Complexity and Explicit Separations	12
3.1 Separations	15
3.2 Explicit Separations	16
4 NEPHEW	16
5 Finding a Leaf in a Binary Tree	23
5.1 EMPTY-CHILD reduces to LOSSY-CODE	24
5.2 Finding a Leaf with Heights	25
5.3 LOSSY-Completeness of EMPTY-CHILD	29
5.4 EMPTY-CHILD Reduces to NEPHEW	32
5.5 EMPTY-CHILD and NEPHEW with Inverse	35
6 The Strength of LOSSY-CODE	39
6.1 Basics of APC ₁	39
6.2 Reductions to Lossy-Code	41
7 Dense Linear Ordering	48
8 Open Problems	49
References	50
A Herbrandization	57
B Proof Complexity Characterizations of Randomized Reductions	58
C Proof of Theorem 6.2	59

1 Introduction

Total search problems are abundant in theoretical computer science. The formal study of these problems has been highly impactful to a wide range of areas including game theory [DGP09, CD06], cryptography [HKKS20, FGH⁺24, BGSD25], proof complexity [BCE⁺98, DR23, LPR24, GHJ⁺22, Tha24, GKRS19, LLR24, FIM25], and recently in the study of explicit construction problems and derandomization [KKMP21, Kor21, Kor25]. Central to the latter has been the *Range Avoidance* (or simply AVOID) problem.

AVOID. Given a circuit $D : \{0, 1\}^{n-1} \rightarrow \{0, 1\}^n$, find $x \in \{0, 1\}^n$ such that for every $y \in \{0, 1\}^{n-1}$, $D(y) \neq x$.

AVOID captures the explicit construction problems for many combinatorial objects whose existence follows from the probabilistic method. Notable examples include functions with high circuit complexity, rigid matrices, Ramsey graphs, strong error correcting codes, and many more [Kor21, Jeř07a, GLW25, GGNS23]. By developing algorithms for AVOID, a line of work has shown circuit lower bounds against a variety of classes [RSW22, CHLR23, CHR24, Li24].

AVOID belongs to the class $\text{TF}\Sigma_2^P$, the second level of the total function polynomial hierarchy. If one *Herbrandizes*¹ the Avoid problem, then one obtains its TFNP sibling, the LOSSY-CODE problem (see [Kor25] for a survey). This problem asks to find an element that is not in the range of a pair of compressing and decompressing maps C and D .

LOSSY-CODE. Given a pair of circuits $C : \{0, 1\}^n \rightarrow \{0, 1\}^{n-1}$ and $D : \{0, 1\}^{n-1} \rightarrow \{0, 1\}^n$, find $x \in \{0, 1\}^n$ such that $D(C(x)) \neq x$.

LOSSY-CODE was originally defined by Jeřábek in [Jeř07a, Jeř07b], under the name *retraction weak pigeonhole principle*, showing that it is equivalent to the set of problems whose totality is provable in APC_1 . Since then, it has been considered predominantly through the lens of bounded arithmetic as a TFNP problem and as a combinatorial principle [Mül21, KT22]. Korten [Kor22] asked to understand the set of TFNP problems that are reducible to LOSSY-CODE. Besides being an interesting problem on its own, LOSSY-CODE also arises naturally in a few other places, further motivating its study:

- **Derandomization.** In the recent *certified derandomization* framework [PRZ23], the derandomization algorithm is required to either output the correct answer, or report that the underlying circuit lower bound assumption is false by providing a small circuit violating the assumption. It turns out that certified derandomization is characterized by LOSSY-CODE [LPT24].

Such derandomization ideas are particularly explored in the context of *catalytic computing* [BCK⁺14, Mer23] in a framework known as “compress-or-random” [Pyn24, CLMP25, KMPS25, AM25]: If the contents of the catalytic tape is “incompressible” (which usually means that it is not a solution of a certain LOSSY-CODE instance), then it can be used for derandomization; otherwise we can compress the catalytic tape and obtain more free space. As a result, although we are currently unable to prove that CL (catalytic logspace) is in P, we can show that CL reduces to LOSSY-CODE [CLMP25].

- **Metamathematics of complexity theory.** It turns out that (variants of) LOSSY-CODE captures the complexity of many *refuter* problems [CJSW24], which are natural total search problems reflecting the metamathematical complexity of proving lower bounds. Many lower bounds in circuit complexity and communication complexity have refuter problems equivalent to LOSSY-CODE [Kor22, CLO24], and the refuter complexity for some proof complexity lower bounds is captured by variants of LOSSY-CODE as well [LLR24].

¹Herbrandization is a basic construction in logic; see [Appendix A](#) for a short overview.

- **Bounded arithmetic.** A basic theory of bounded arithmetic for approximate counting and reasoning about randomized computation is APC_1 , developed in a series of papers by Jeřábek [Jeř04, Jeř05, Jeř07a]. Wilkie’s witnessing theorem [Tha02, Jeř04] implies that LOSSY-CODE is “complete” for APC_1 in the following sense: APC_1 proves the totality of LOSSY-CODE, and every TFNP problem provably total in APC_1 reduces to LOSSY-CODE.

LOSSY-CODE belongs to the class TFZPP, the subclass of TFNP containing the total search problems that admit polynomial-time randomized algorithms, introduced in [BO06]. Since we are dealing with total NP search problems, every randomized algorithm that may make mistakes can be turned into one that does not make any mistakes.² Hence, it seems that TFZPP is the only natural (semantic) subclass of TFNP capturing randomized polynomial time.

Besides LOSSY-CODE, there are a variety of important total search problems that sit inside TFZPP. We list two of them that we think reflect the importance of TFZPP the best:

Example 1.1. The Bertrand–Chebyshev theorem states that for every integer $N \geq 1$, there is a prime number between N and $2N$. This motivates the following total search problem called BERTRAND-CHEBYSHEV: Given an integer N (represented in binary), output a prime number between N and $2N$. In fact, the Prime Number Theorem implies that there are $\Theta(N/\log N)$ such prime numbers, and the AKS primality test [AKS04] provides a deterministic method for verifying solutions, hence BERTRAND-CHEBYSHEV is in TFZPP.

The complexity of BERTRAND-CHEBYSHEV remains elusive. Unless one makes strong assumptions such as Cramér’s conjecture [Cra36] or $\text{P} = \text{BPP}$ [IW97], the best known deterministic algorithm needs to spend $\approx N^{1/2}$ time [LO87, BHP01] (which is *exponential* in the input length). Improving this time bound was exactly the focus of the Polymath 4 project [TCH12]; however, despite much effort, no unconditional progress was made. This problem is also the flagship problem in the study of *pseudodeterministic algorithms* [GG11, OS17, LOS21, CLO⁺23].

On the complexity-theoretic side, the only upper bound known for BERTRAND-CHEBYSHEV is that it reduces to $\text{LOSSY-CODE}^{\text{FACTORING}}$, i.e., the LOSSY-CODE problem where both input circuits C, D have access to a FACTORING oracle [PWW88, Kor22]. It is unclear if it belongs to any standard TFNP subclasses such as PLS or PPA [GP18, GL25].

Example 1.2. A family of important total search problems is *refuter problems* [CLW20, CJSW24, CTW23] for complexity lower bounds. Let \mathcal{C} be a circuit class and L be a hard problem for \mathcal{C} , the refuter problem, $\text{REFUTER}(L \notin \mathcal{C})$, is the following total search problem: given a small \mathcal{C} circuit C attempting to compute L , the goal is to output an instance x such that $C(x) \neq L(x)$. The complexity of these refuter problems are closely related to the provability of complexity lower bounds [CLO24, LLR24].

Such refuter problems are often in TFZPP: In fact, if L is *average-case hard* against \mathcal{C} (and both \mathcal{C} and L are in polynomial-time), then $\text{REFUTER}(L \notin \mathcal{C}) \in \text{TFZPP}$ as the algorithm for the refuter problem can repeatedly sample inputs from the hard distribution until it finds a solution x where $C(x) \neq L(x)$. Even though average-case lower bounds against $\text{AC}^0[p]$ circuits have been proved for nearly 40 years [Raz87, Smo87], we are not aware of any non-trivial TFNP upper bound for the problem $\text{REFUTER}(\text{MAJ} \notin \text{AC}^0[2])$.

Another example is when $L = \text{search-SAT}$ and \mathcal{C} is the family of polynomial-size circuits: Given a circuit C attempting to solve search-SAT, the refuter problem asks to find a formula φ (along with a satisfying assignment a of φ) such that $C(\varphi)$ fails to satisfy φ . The complexity of this problem is of significant interest to the bounded arithmetic community [Kra95, Bus97, Pic15, PS23] as its hardness would imply the unprovability of $\text{NP} \not\subseteq \text{P}/\text{poly}$. However, this problem is in TFZPP under the existence of one-way functions against non-uniform adversaries, hence it is unclear how its complexity sheds light on the aforementioned unprovability question.

Finally, an additional motivation for studying TFZPP is its connection to AVOID and APEPP (the class of total search problems mapping reducible to AVOID [KKMP21]): it is the “projection” of AVOID to TFNP in the following sense:

Theorem 1.3. $\text{TFZPP} = \text{TFNP} \cap \text{APEPP}$.

²This was observed by Jeřábek [Jeř16]; in his terminology, we have $\text{TFRP} = \text{TFZPP}$.

Our Contributions.

In this work, we initiate a formal study of TFZPP as a class of total search problems. Analogous to the setting of decision problems, we expect that $\text{TFZPP} = \text{FP}$. Indeed, this follows from the same assumption used in [IW97]—namely that E requires circuits of exponential size. Moreover, $\text{TFZPP} = \text{FP}$ appears to be weaker than a full derandomization of BPP.

However, we show that this is not the case in the *black-box* setting in a very strong sense. In the black-box setting one only has access to the input via an oracle; black-box classes are denoted by a dt superscript (for “decision trees”). We say that a TFNP^{dt} class is *uniformly generated* if it has a complete problem $R = \{R_n\}_{n \in \mathbb{N}}$ such that there is a Turing Machine which on input 1^n outputs R_n in polynomial time. Note that all of the major TFNP^{dt} subclasses in the literature are uniformly generated. Under the assumption that NP is not in quasi-polynomial time (QP), we show that no uniformly generated TFNP^{dt} class contains TFZPP^{dt} .

Theorem 1.4. $\text{TFZPP}^{dt} \not\subseteq \mathcal{C}$ for every uniformly generated class $\mathcal{C} \subseteq \text{TFNP}^{dt}$, unless $\text{NP} \subseteq \text{QP}$.

To prove these separations, we employ a close connection between total search problems and proof complexity [BCE⁺98, GKRS19, BFI23, FIM25]. This connection shows that, in the black-box setting, a search problem belongs to a class if and only if an associated proof system can prove the totality of that search problem. In this case, we say that the class is *characterized* by that proof system. To prove our separations, we first show that TFZPP^{dt} is characterized by the random tree-like resolution proof systems of Buss et al. [BKT14].

Theorem 1.5. TFZPP^{dt} is characterized by random tree-like resolution.

More generally, we show that if a class \mathcal{C} of total search problems is characterized by a proof system Π , then the class of problems that are efficiently *randomized* reducible to a complete problem in \mathcal{C} is characterized by the proof system random Π . Theorem 1.4 then follows by combining the following two results: (1) Buss et al. [BFI23] showed that every uniformly generated TFNP class has a characterizing proof system, and (2) Pudlák and Thapen [PT19] showed that a propositional proof system simulating random tree-like resolution would imply faster algorithms for NP.

A Highly Unsatisfiable Cook-Reckhow Program. Theorem 1.4 is striking as it suggests that TFZPP problems can be arbitrarily hard in the black-box model. This motivates an interesting direction of research: find explicit TFZPP problems that are hard for stronger and stronger subclasses of TFNP^{dt} . By the close connection between TFNP^{dt} and proof complexity [BFI23], this can be seen as a Cook-Reckhow program for highly unsatisfiable formulas: for increasingly more expressive proof systems, exhibit a highly unsatisfiable CNF formula which is hard for that system.

Towards this program, we provide explicit separations of TFZPP^{dt} from every major TFNP class defined in the 1990s [JPY88, Pap94]. Note that all those TFNP classes are contained in PLS, PPP, and PPA. The separation from PPA was shown by Beame et al. [BCE⁺98]. Leveraging the recent work of Hopkins and Lin [HL22], we are able to show the following.

Theorem 1.6. There are *explicit* (polynomial-time constructable) problems in TFZPP^{dt} which are not in either PPP^{dt} or PLS^{dt} .

A TFZPP Zoo.

We now turn to studying the structure of problems inside of TFZPP. Figure 1 shows the zoo of problems within TFZPP that we consider, as well as their relationships.

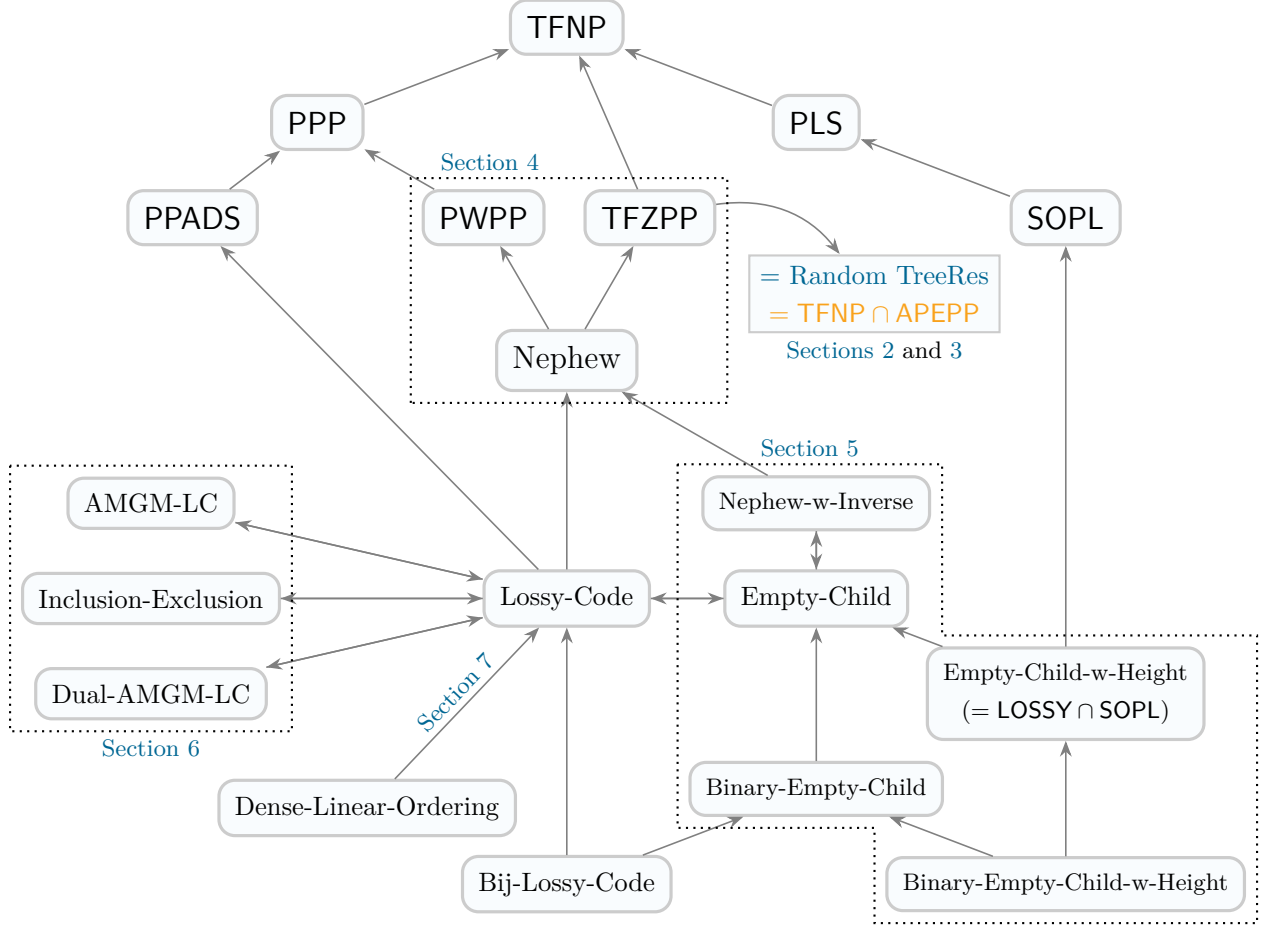


Figure 1: The TFZPP zoo.

Like ZPP, TFZPP is a semantic class and therefore it is unlikely to admit complete problems unless $FP = TFZPP$. However, we observe an interesting phenomenon: almost every TFZPP problem that has been studied in the literature is reducible to LOSSY-CODE!³ This raises the question: are there “natural” TFZPP problems which are not reducible LOSSY-CODE in the black-box setting, and what do they look like? [Theorem 1.4](#) and [Theorem 1.6](#) already provide examples of problems that are not reducible to LOSSY-CODE; however, we do not consider these problems natural—we are looking for problems that would be studied outside of the context of proving such separations.

While we are unable to resolve this question—indeed, many of our conjectured separating examples turned out to be reducible to LOSSY-CODE in surprising ways!—we provide natural TFZPP problems which we conjecture witness a separation, and which we believe are of independent interest. As well, we show several surprising reductions to LOSSY-CODE.

Nephew. Our primary candidate is the following.

³One exception is the Bertrand–Chebyshev problem for which it is not known whether it is reducible to LOSSY-CODE. However, it is unclear how to define the Bertrand–Chebyshev problem in the *black-box* model, and we are unable to separate any natural black-box TFZPP problem from LOSSY-CODE.

NEPHEW. Given a set V of vertices and two functions $f : V \rightarrow V$ and $g : V \rightarrow V$. Think of $f(v)$ as the *father* of v and $g(v)$ as the *nephew* of v . A solution is one of the following.

- s1. $v \in V$ such that $f(f(g(v))) \neq f(v)$ (your nephew's grandparent is not your parent)
- s2. $v \in V$ such that $f(g(v)) = v$ (you are your nephew's parent)

It may not be immediately obvious that NEPHEW is a total search problem. A proof may be found in the textbook of Börger, Grädel, and Gurevich [BGG01, Proposition 6.5.5]; we sketch the argument here. Create a directed graph G_f with vertex set V and with an edge from u to v if $f(u) = v$. Then one can assign to each vertex a “level” that represents its distance to the core⁴ of G_f . Let v^* be a vertex with maximum level ℓ_{\max} . However, if v^* is not a solution to NEPHEW, it must be that $g(v^*)$ has level $\ell_{\max} + 1$, a contradiction. A similar intuition will be used in Section 4 for other proofs involving NEPHEW.

NEPHEW is derived from the minimal *axioms of infinity* in model theory. One method for constructing a total search problem is to begin with a sentence in logic that has an infinite model but no finite model. Such a sentence is known as an *axiom of infinity*, since any model for it must be infinite. Axioms of infinity are classified under the number of quantifiers and predicate and function symbols of certain arity that they contain, and there are 10 minimal classes ([BGG01, Theorem 6.5.4]). Each of these classes corresponds to a total search problem, and in most cases, this problem is complete for a well-known TFNP class. The only exception is the one to which NEPHEW belongs; NEPHEW can be interpreted as the Herbrandization of

$$\forall x \exists y (F(F(y)) = F(x) \wedge F(y) \neq x). \quad (1)$$

The proof that NEPHEW belongs to TFZPP is highly non-trivial. Furthermore, our best upper bound on the complexity of NEPHEW is that it is contained within PWPP, the problems reducible to the weak pigeonhole principle, a relaxation of the LOSSY-CODE problem.

Theorem 1.7. $\text{NEPHEW} \in \text{TFZPP} \cap \text{PWPP}$.

To obtain evidence that NEPHEW is not reducible LOSSY-CODE, we attempt to isolate the potential hardness in NEPHEW. In doing so, we define a number of natural problems in TFZPP which may be of independent interest.

The proof that NEPHEW is in TFZPP proceeds by reducing it to the observation that a leaf in a binary tree can be found in logarithmic time in expectation. We define a total search problem whose membership in TFZPP formalizes this observation.

EMPTY-CHILD. Given a set V of vertices and three functions $F, L, R : V \rightarrow V$, where $F(u)$ is the father of u , and $L(u), R(u)$ are the left and the right child of u respectively, a solution is one of the following.

- s1. $u \in V$ such that $F(L(u)) \neq u$ or $F(R(u)) \neq u$ or $L(u) = R(u) \neq u$; (Empty child)
- s2. 1, if $L(1) = 1$ or $R(1) = 1$ or $F(1) \neq 1$. (Wrong root)

Surprisingly, EMPTY-CHILD is equivalent to LOSSY-CODE under decision tree reductions, denoted $=_{dt}$. Thus, if NEPHEW is indeed not reducible to LOSSY-CODE, the hardness of NEPHEW does not come from this portion of the reduction.

Theorem 1.8. $\text{EMPTY-CHILD} =_{dt} \text{LOSSY-CODE}$.

⁴We intentionally leave “core” undefined for this brief sketch.

As a warm-up to the techniques needed to prove this theorem, we consider a variant of EMPTY-CHILD which includes an additional “height” function that outputs the height of a given node in the tree. We show that this is a complete problem for the class $\text{LOSSY} \cap \text{PLS} = \text{LOSSY} \cap \text{SOPL}$, where LOSSY is the class of problems efficiently reducible to LOSSY-CODE . The proof resembles previous intersection theorems from TFNP [FGHS23, GHJ⁺24].

Then, we use EMPTY-CHILD as an intermediate problem to study the relationship between NEPHEW and LOSSY-CODE .

Theorem 1.9. $\text{EMPTY-CHILD} \leq_{dt} \text{NEPHEW}$.

Thus, combining with Theorem 1.8, we have LOSSY-CODE reduces to NEPHEW.

AM-GM Lossy-Code. One of our original (and failed) candidates for separation from LOSSY-CODE was a problem called *AM-GM Lossy-Code*, obtained by combining LOSSY-CODE itself with the *AM-GM Inequality*: $\frac{a+b}{2} \geq \sqrt{ab}$. This problem was inspired by the BAD 2-COLORING problem in [PPY23]: Given an undirected graph $G = (V, E)$ with $|V| = 2N$ vertices and $|E| = N^2 + 1$ edges along with a 2-coloring $C : V \rightarrow \{0, 1\}$ of V , find an edge $(x, y) \in E$ that is not colored properly. This problem is total exactly because of the AM-GM inequality: suppose there are a black vertices and $b = 2N - a$ white vertices and every edge is colored properly, then there are at most $ab \leq \left(\frac{a+b}{2}\right)^2 = N^2$ edges, contradicting $|E| > N^2$.

To compose this problem with LOSSY-CODE , we need to make two adaptations: First, to put it inside TFZPP, the number of edges needs to be *much* larger than N^2 , say $(1 + \varepsilon)N^2$; second, we are given a function F from $[(1 + \varepsilon)N^2]$ to the set of properly colored edges and its purported inverse G and we need to find $x \in [(1 + \varepsilon)N^2]$ such that $G(F(x)) \neq x$. We arrive at the following problem:

c-AMGM-LC. Let $c > 1$ be a constant, $V := [2N]$ and $P := [c \cdot N^2]$. The input is a coloring function $C : V \rightarrow \{0, 1\}$ and two mappings $F : P \rightarrow V \times V$, $G : V \times V \rightarrow P$. Let $H := C^{-1}(0) \times C^{-1}(1)$. The goal is to find solutions of either type:

- s1. a pigeon $x \in P$ such that $G(F(x)) \neq x$; (Wrong Encoding-Decoding)
- s2. a pigeon $x \in P$ such that $F(x) \notin H$; (Invalid Hole)

Indeed, it seems unclear how to massage a LOSSY-CODE instance of the shape $[cN^2] \rightleftharpoons H$ into a standard LOSSY-CODE instance of the form $[2M] \rightleftharpoons [M]$, even though the AM-GM inequality implies that $cN^2 \gg |H|$. However, it turns out that such a reduction is possible (although highly non-trivial)! We encourage the reader to take a moment to think about how to reduce AMGM-LC to LOSSY-CODE .

Theorem 1.10. For every constant $c > 1$, $c\text{-AMGM-LC} \leq_{dt} \text{LOSSY-CODE}$.

Our reduction goes through *bounded arithmetic*: We formalize the totality of $c\text{-AMGM-LC}$ in Jeřábek’s theory APC_1 [Jeř07a], and Wilkie’s witnessing theorem for APC_1 [Tha02, Jeř04] implies a reduction from $c\text{-AMGM-LC}$ to LOSSY-CODE . In particular, like every formalization in APC_1 , our reduction makes use of the *Nisan–Wigderson generator* [NW94].

Moreover, the techniques underlying APC_1 [Jeř07a] allow us to reduce problems to LOSSY-CODE in a *systematic* way; we provide two additional examples later ($c\text{-Dual-AMGM-LC}$ in Section 6.2.2 and a problem capturing the Inclusion-Exclusion principle in Section 6.2.3). A secondary goal of expounding these reductions is to introduce the ideas of APC_1 to audiences who are less familiar with bounded arithmetic.

Remark 1.11. Unfortunately, it seems unclear how to formalize the totality of NEPHEW in APC_1 , hence the bounded arithmetic approach does not seem to provide a reduction from NEPHEW to LOSSY-CODE. For example, our proof that $\text{NEPHEW} \in \text{TFZPP}$ requires reasoning about the *level* of each node, which seems to be global reasoning that is infeasible in APC_1 .

Linear Ordering. Finally, we consider one more natural problem in TFZPP. The Linear Ordering Principle has a storied history in proof complexity [Kri85, BG01, Pot20] and bounded arithmetic [CK98, Han04, BKT14, AT14]. It has been studied in the context of total search problems as well [KP24, HV25], where it was used in order to construct new algorithms for AVOID. A line of works in proof complexity [Rii01, AD08, Gry19, CdRN⁺23] has also considered a dense variant of this problem defined as follows, which lies in TFZPP.

DENSE-LINEAR-ORDERING. The input consists of the descriptions of a linear ordering \prec over N elements and a *median function* $\text{med} : [N] \times [N] \rightarrow [N]$. Without loss of generality, we may assume that for $x \neq y \in [N]$, exactly one of $(x \prec y)$ and $(y \prec x)$ is true, and that $\text{med}(x, y) = \text{med}(y, x)$. (That is, \prec is represented by a string of $\binom{N}{2}$ bits and med is represented by a list of $\binom{N}{2}$ elements in $[N]$.) A solution is one of the following.

- s1. $x, y, z \in [N]$ such that $x \prec y$, $y \prec z$, and $z \prec x$; or (Transitivity violation)
- s2. $x, y \in [N]$ such that $x \prec y$, but neither $x \prec \text{med}(x, y)$ nor $\text{med}(x, y) \prec y$. (Invalid median)

While AVOID reduces to the Linear Ordering Principle [KP24], we show a converse in the dense setting.

Theorem 1.12. $\text{DENSE-LINEAR-ORDERING} \leq_{dt} \text{LOSSY-CODE}$.

2 Preliminaries

2.1 Basics of TFNP

TFNP contains all search problems which are (i) total: a solution is guaranteed to exist, and (ii) in NP: there is an efficient procedure to check whether a candidate solution is valid. It is believed that TFNP does not admit complete problems [Pud15] and much of the research in this area has focused on studying syntactic subclasses (those with complete problems) which capture many of the total search problems of interest. These classes are typically defined by simple existence principles that capture the totality of the problems within that class. These naturally give rise to total search problems. For example, PWPP is the class of all search problems whose totality is witnessed by the existence principle: any map from $2N$ to N must have a collision [Jeř16]. To make these problems non-trivial, the input is presented succinctly as a circuit C that on input i outputs the i -th bit of the search problem. For example, the existence principle for PWPP gives rise to the following (white-box) total search problem.

WEAK-PIGEON. Given $P : \{0, 1\}^n \rightarrow \{0, 1\}^{n-1}$, a solution is $x \neq y$ such that $P(x) = P(y)$.

PWPP is then the class of all total search problems which are efficiently reducible to WEAK-PIGEON.

A major thrust of this line of work is to understand the relationships between these classes. However, a separation between classes would imply $\text{P} \neq \text{NP}$. As a proxy, and as natural objects in their own right, researchers have studied total search problems in the *black-box* model. In this setting, the input C is given as a black box which can be queried, but we no longer have access to the description of C .

In this setting a (*query*) *search problem* is a sequence of relations $R_n \subseteq \{0, 1\}^n \times \mathcal{O}_n$, for each $n \in \mathbb{N}$. It is total if for every $x \in \{0, 1\}^n$ there is $o \in \mathcal{O}_n$ such that $(x, o) \in R_n$. We think of the input $x \in \{0, 1\}^n$ as being accessed by querying the individual bits, and we will measure the complexity of solving R_n as the number of bits that must be queried to determine some suitable $o \in \mathcal{O}_n$. An efficient algorithm is one that makes at most $\text{poly}(\log n)$ -many queries⁵; these problems belong to the class FP^{dt} , where *dt* indicates that it is the black-box version of the class. Similarly, $R \in \text{TFNP}^{dt}$ if for every $n \in \mathbb{N}$ and each $o \in \mathcal{O}_n$ there exists a $\text{poly}(\log n)$ -depth decision tree $T_o : \{0, 1\}^n \rightarrow \{0, 1\}$ such that $T_o(x) = 1$ iff $(x, o) \in R_n$. While search problems are formally defined as a sequence $R = (R_n)_{n \in \mathbb{N}}$, we will often want to speak about individual members of this sequence. For readability, we will abuse notation and refer to elements R_n in the sequence as total search problems. Furthermore, we will often drop the subscript n and rely on context to differentiate.

We compare the complexity of total search problems by reductions between them; the following is the black-box (decision tree) analogue of a deterministic polynomial-time reduction between search problems.

Definition 2.1. For total search problems $R \subseteq \{0, 1\}^n \times \mathcal{O}_n$ and $S \subseteq \{0, 1\}^m \times \mathcal{O}'_m$, there is an S -formulation of R if for every $i \in [m]$ and $o \in \mathcal{O}'_m$ there are functions $f_i : \{0, 1\}^n \rightarrow \{0, 1\}$ and $g_o : \{0, 1\}^n \rightarrow \mathcal{O}_n$ such that

$$(f(x), o) \in S \implies (x, g_o(x)) \in R, \quad (2)$$

where $f(x) := (f_1(x), \dots, f_m(x))$. The *depth* of the S -formulation is

$$d := \max(\{\text{depth}(f_i) : i \in [m]\} \cup \{\text{depth}(g_o) : o \in \mathcal{O}'_m\}),$$

where $\text{depth}(f)$ denotes the minimum depth of any decision tree which computes f . The *size* of the S -formulation is m , the number of input bits to S . The *complexity* of an S -formulation is $\log m + d$ and the complexity of reducing R to S is the minimum complexity of any S -formulation of R .

This definition extends to sequences naturally. If $S = (S_n)$ is a sequence and R_n is a single search problem, then the complexity of reducing R_n to S is the minimum over m of the complexity of reducing R_n to S_m . For two sequences $S = (S_n)$ and $R = (R_n)$, the complexity of reducing R to S is the complexity of reducing R_n to S for each n . We say that a reduction from R to S is *efficient* if its complexity is $\text{poly}(\log(n))$ and denote this by $R \leq_{dt} S$.

2.2 TFZPP

In this work, we will be particularly interested in the total search problems which are solvable in *randomized* polynomial time. Formally, $R \subseteq \{0, 1\}^* \times \{0, 1\}^* \in \text{TFZPP}$ if there is a distribution \mathcal{D} over polynomial-time Turing Machines A with range $\{0, 1, \perp\}$, such that $\Pr_{A \sim \mathcal{D}}[A(x) = \perp] \leq 1/3$ and

TFZPP is defined semantically and it is unlikely to have complete problems. However, we show that it is exactly the TFNP problems in the $\text{TF}\Sigma_2^P$ class APEPP, where APEPP is the class of total search problems that are reducible to AVOID, as defined in [KKMP21].

Theorem 1.3. $\text{TFZPP} = \text{TFNP} \cap \text{APEPP}$.

Proof. Let $L \in \text{TFNP} \cap \text{APEPP}$. This means that given an instance x of L , there are deterministic polynomial-time algorithms V , C , and R , such that:

⁵As the input is succinctly encoded, this corresponds to looking at a polynomial part of the entire input.

- **V is a TFNP verifier for L.** For every string z of length polynomial in $|x|$, $V(x, z) = 1$ if and only if z is a valid solution for x .
- **(C, R) is a reduction from x to AVOID.** The output of $C(x)$ is a circuit C_x mapping ℓ input bits to $\ell + 1$ output bits, where $\ell \leq \text{poly}(|x|)$; given any $y \in \{0, 1\}^{\ell+1} \setminus \text{Range}(C_x)$, $R(x, y)$ outputs a valid solution for x .

Then we can solve L in TFZPP via the following procedure. Guess $y \in \{0, 1\}^{\ell+1}$ uniformly at random and compute $z := R(x, y)$. If $V(x, z)$ accepts, then we output z ; otherwise, we output \perp . By the correctness of V , if we did not output \perp , then our output is a valid solution of x . On the other hand, since at least a $1/2$ fraction of strings $y \in \{0, 1\}^{\ell+1}$ are valid outputs of AVOID on the instance C_x , by the correctness of (R, C) , we will output a valid solution w.p. at least $1/2$.

Now we prove the converse direction. If $L \in \text{TFZPP}$ then clearly $L \in \text{TFNP}$; hence we only need to show that there is a mapping reduction from L to AVOID. Let $U(x, r)$ be the zero-error randomized algorithm for L , i.e., $U(x, r)$ outputs a valid solution for x w.p. at least $1/2$ over its randomness r , and it outputs \perp whenever it fails to output a valid solution.

By standard results in derandomization [NW94, IW97, Uma03], there exist absolute constants $c, d \geq 1$ and a deterministic polynomial-time algorithm PRG such that the following holds. For every truth table f of length s^{10c} , if the circuit complexity of f is at least s^c , then $\text{PRG}(f)$ outputs a list of s^d strings that $(1/s^2)$ -fools every size- s^2 circuit. That is, for every circuit $C : \{0, 1\}^s \rightarrow \{0, 1\}$ of size at most s^2 ,

$$\left| \Pr_{x \sim \{0, 1\}^s} [C(x) = 1] - \Pr_{x \sim \text{PRG}(f)} [C(x) = 1] \right| \leq 1/s^2.$$

Now, let $s \leq \text{poly}(|x|)$ be the circuit complexity of U . Consider the *truth table generator* $\text{TT} : \{0, 1\}^{O(s^c \log s)} \rightarrow \{0, 1\}^{s^{10c}}$ that takes the description of a size- s^c circuit $C : \{0, 1\}^{10c \log s} \rightarrow \{0, 1\}$ as input and outputs the length- s^{10c} truth table of C . We treat TT as an instance for AVOID and reduce x to TT .⁶

It remains to show how to solve the instance x deterministically given a non-output of TT . Note that if $f \in \{0, 1\}^{s^{10c}}$ is a non-output of TT , then the circuit complexity of f is at least s^c , hence $\text{PRG}(f)$ outputs a list of $\text{poly}(s)$ strings that $(1/s^2)$ -fools every size- s^2 circuits. This implies that

$$\Pr_{r \sim \text{PRG}(f)} [U(x, r) \neq \perp] \geq \Pr_{r \sim \{0, 1\}^s} [U(x, r) \neq \perp] - 1/s^2 \geq 1/2 - 1/s^2 > 0,$$

and in particular, there exists at least one string $r \in \text{PRG}(f)$ such that $U(x, r) \neq \perp$. We can solve the instance x by cycling through every $r \in \text{PRG}(f)$ and outputting $U(x, r)$ whenever we encounter such a good r . \square

Note that this equivalence holds even in the black-box model, since its proof is relativizing.

Definition 2.2 (TFZPP). A total NP search problem $R \subseteq \{0, 1\}^* \times \{0, 1\}^*$ is in TFZPP if there is a distribution \mathcal{D} over polynomial-time algorithms A with output in $\{0, 1, \perp\}$ such that:

1. For every $x \in \{0, 1\}^*$ and every $A \sim \mathcal{D}$, if $A(x) \neq \perp$ then $(x, A(x)) \in R$,
2. For every $x \in \{0, 1\}^*$,

$$\Pr_{A \sim \mathcal{D}} [A(x) = \perp] \leq 1/3.$$

Similarly, $R \in \text{TFZPP}^{dt}$ if there is a family of distributions $\mathcal{D} = \{\mathcal{D}_n\}_{n \in \mathbb{N}}$ over $\text{poly} \log(n)$ -depth decision trees with leaves labeled in $\{0, 1, \perp\}$, where on input x we sample a decision tree $A \sim \mathcal{D}_{|x|}$, and \mathcal{D} satisfies (1) and (2).

⁶An unusual aspect of this reduction is that TT does *not* depend on x !

2.3 LOSSY-CODE

As mentioned in the introduction, LOSSY-CODE is the Herbrandization of AVOID. Let $N < M$ be two parameters (think of $N \ll M$), (the black-box version of) LOSSY-CODE is the following problem:

LOSSY-CODE $_{N \rightarrow M}$. Given query access to a pair of functions $f : [N] \rightarrow [M]$ and $g : [M] \rightarrow [N]$, find $x \in [M]$ such that $f(g(x)) \neq x$.

We need the following basic fact about LOSSY-CODE that roughly states that the “stretch function” of LOSSY-CODE does not influence its complexity as long as it is in the “weak” regime. This fact and similar statements for other variants of the weak pigeonhole principle have been very useful in bounded arithmetic [PWW88, Tha02, Kra04, Jeř04, Jeř07a, CLO24], total search problems [Kor21, Kor22, LLR24], and cryptography [GGM86, Mer87].

Lemma 2.3. Let $\varepsilon > 0$ and $M > (1 + \varepsilon)N$. There is a decision tree reduction of complexity $O(\varepsilon^{-1} \log(M/N))$ from LOSSY-CODE $_{N \rightarrow (1+\varepsilon)N}$ to LOSSY-CODE $_{N \rightarrow M}$.

By Lemma 2.3, LOSSY-CODE $_{N \rightarrow 1.01N}$, LOSSY-CODE $_{N \rightarrow 2N}$, and LOSSY-CODE $_{N \rightarrow N^{100}}$ are equivalent up to decision tree reductions of $\text{polylog}(N)$ depth. In this paper, unless otherwise stated, LOSSY-CODE always stands for LOSSY-CODE $_{N \rightarrow 2N}$.

We denote LOSSY as the class of total search problems reducible to LOSSY-CODE.⁷ It follows from Lemma 2.3 that LOSSY is robust in the sense that it does not matter whether it is defined using LOSSY-CODE $_{N \rightarrow 1.01N}$ or LOSSY-CODE $_{N \rightarrow N^{100}}$ as the complete problem. In fact, LOSSY is *extremely* robust: it is closed under Turing reductions ($\text{FP}^{\text{LOSSY}} = \text{LOSSY}$ [BJ12, LPT24]) and it is self-low ($\text{LOSSY}^{\text{LOSSY}} = \text{LOSSY}$ [GL25]).

We will also consider the *bijective* version of LOSSY-CODE, called BIJ-LOSSY-CODE. Let $N < M$, define:

BIJ-LOSSY-CODE $_{N \rightarrow M}$. Given query access to a pair of functions $f : [N] \rightarrow [M]$ and $g : [M] \rightarrow [N]$, find either $x \in [M]$ such that $f(g(x)) \neq x$, or $y \in [N]$ such that $g(f(y)) \neq y$.

Clearly, BIJ-LOSSY-CODE $_{N \rightarrow M}$ reduces to LOSSY-CODE $_{N \rightarrow M}$. We also use BIJ-LOSSY-CODE to denote BIJ-LOSSY-CODE $_{N \rightarrow 2N}$ by default. It is not difficult to see that the “strong” versions of these problems, LOSSY-CODE $_{N \rightarrow N+1}$ and BIJ-LOSSY-CODE $_{N \rightarrow N+1}$, are complete for PPADS and PPAD respectively.

3 Randomized Proof Complexity and Explicit Separations

We begin by describing the connection between black-box TFZPP and proof complexity, and how this can be leveraged to obtain explicit separations from other natural classes. Proof complexity is concerned with the efficient provability of propositional theorems (unsatisfiable CNF formulas) in various *proof systems*—simply a verifier for the language UNSAT of unsatisfiable CNF formulas.

Definition 3.1. A propositional proof system is a polynomial-time machine \mathcal{P} such that for every CNF formula F , $F \in \text{UNSAT}$ iff there exists a proof $\Pi \in \{0, 1\}^*$ such that $\mathcal{P}(F, \Pi) = 1$. We say that Π is a \mathcal{P} -proof of F and define the size of Π to be $s(\Pi) := |\Pi|$.

When studying connections between proof systems and TFNP classes, it is standard to also consider an associated notion of the *width* of a proof $w(\Pi)$. This is typically specific to the proof

⁷Previous literature [LPT24, CLMP25] defined LOSSY as the class of *decision problems* reducible to LOSSY-CODE. In our context, it is more natural to define LOSSY as a class of total search problems.

system—for example, in resolution (defined next), it is the maximum number of literals in a clause in Π , while for algebraic systems such as Sum-of-Squares, the width is the degree of the polynomials occurring in the proof.

With a definition of width, the *complexity* of proving F in \mathcal{P} is

$$\mathcal{P}(F) := \min_{\mathcal{P}\text{-proof } \Pi \text{ of } F} w(\Pi) + \log s(\Pi).$$

A standard example is the *resolution* proof system. A resolution proof of an unsatisfiable CNF formula F consists of a sequence of clauses $C_1, \dots, C_t = \emptyset$ ending with the empty clause which contains no literals, such that each clause C_i either belongs to F or is derived from earlier clauses in the sequence according to the *resolution rule*.

Resolution rule: From two clauses with complementary literals $A \vee x$ and $B \vee \bar{x}$, derive $A \vee B$.

The size of a resolution proof is the number of clauses that it contains, while the width is the maximum number of literals within any clause in the proof. A resolution proof C_1, \dots, C_t is *tree-like* if each C_i is used at most once as a premise for the resolution proof. They are named as such because the implication graph of such proofs is a tree.

There is a long line of work connecting proof complexity and black-box TFNP [BCE⁺98, GKRS19, GHJ⁺22, BFI23, FIM25, HKT24, Tha24, DR23, LPR24, FGPR24]. These connections show that a total search problem is contained within a class iff an associated proof system can prove the totality of that search problem. We can phrase the totality of any total search problem $R \subseteq \{0, 1\}^n \times \mathcal{O}$ as an unsatisfiable CNF formula in the following way: for each $o \in \mathcal{O}$ let V_o be a decision tree which checks whether o is a solution; that is, $V_o(x) = 1$ iff $(x, o) \in R$. A root-to-leaf path in V_o is a *1-path* if its leaf is labeled 1. We will associate with any path p the conjunction of literals that it follows. Then the totality of R is expressed as

$$F_R := \neg \left(\bigvee_{o \in \mathcal{O}} \bigvee_{1\text{-path } p \in V_o} p \right).$$

If $R \in \text{TFNP}^{dt}$ then V_o can be assumed to have depth $\text{poly}(\log(n))$, and hence the width of F_R is also $\text{poly}(\log(n))$.

Similarly, we can associate with any unsatisfiable CNF formula $F = C_1 \wedge \dots \wedge C_m$ a total search problem $\text{SEARCH}_F \subseteq \{0, 1\}^n \times [m]$ such that $(x, o) \in \text{SEARCH}_F$ iff $C_o(x) = 0$. Observe that whenever F has $\text{poly}(\log(n))$ width then $\text{SEARCH}_F \in \text{TFNP}^{dt}$ and furthermore that SEARCH_{F_R} is reducible to R by decision trees of depth at most the width of F_R .

For a syntactic class $\mathcal{C} \subseteq \text{TFNP}^{dt}$ we will denote by $\mathcal{C}(R)$ the complexity of reducing R to S , where S is any complete problem for \mathcal{C} . We say that a proof system \mathcal{P} is *characterized* by a class $\mathcal{C} \subseteq \text{TFNP}^{dt}$ if $R \in \mathcal{C}$ iff $\mathcal{P}(F) = \text{poly}(\mathcal{C}(R))$. A standard example is that FP^{dt} characterizes tree-like resolution. Said differently, decision trees are equivalent to tree-like resolution proofs.

We extend these characterizations to capture randomized reductions. We show that randomized reductions between total search problems give rise to proofs in randomized proof systems, a notion introduced by Buss, Kolodziejczyk, and Thapen [BKT14].

Definition 3.2. Let \mathcal{P} be any propositional proof system. A *randomized \mathcal{P} -proof*, denoted $r\mathcal{P}$, of an unsatisfiable formula F is a distribution \mathcal{D} supported on pairs (Π, B) , such that

1. Each B is a CNF formula over the variables of F ,
2. Π is a \mathcal{P} proof of $F \wedge B$,

3. For any assignment $x \in \{0, 1\}^n$, $\Pr_{(\Pi, B) \sim \mathcal{D}}[B(x) = 1] \geq 2/3$.

The *size* $s(\mathcal{D})$, and *width* $w(\mathcal{D})$ of an $r\mathcal{P}$ -proof \mathcal{D} is the maximum width and size of a proof Π in the support of \mathcal{D} . The *complexity* of proving F in $r\mathcal{P}$ is

$$r\mathcal{P}(F) := \min_{r\mathcal{P}\text{-proof } \mathcal{D} \text{ of } F} w(\mathcal{D}) + \log s(\mathcal{D})$$

Note that a randomized proof system is not a Cook-Reckhow proof system in the sense of [Definition 3.1](#) since its proofs typically cannot be polynomial-time verified [\[PT19\]](#).

The main theorem of this section, [Theorem 3.4](#), shows that a proof system \mathcal{P} is characterized by class \mathcal{C} iff the totality of the total search problems R which are randomly reducible to any complete problem for \mathcal{C} is provable in $r\mathcal{P}$. The following definition is equivalent to the probabilistic reduction in [\[Jeř16\]](#).

Definition 3.3. A randomized (ZPP) reduction from a search problem $S \subseteq [t] \times \mathcal{O}$ to $R \subseteq [n] \times \mathcal{Q}$ is a distribution \mathcal{D} over deterministic reductions $\mathcal{T} = (T, \{T_o\})$ such that each output decision tree is labeled either by some $j \in \mathcal{O}$ or by \perp , and \mathcal{D} satisfies

1. For every $x \in [t]$ and every $\mathcal{T} \sim \mathcal{D}$, if $(T(x), o) \in R$ then either $T_o(x) = \perp$ or $(x, T_o(x)) \in S$.
2. For every $x \in [t]$,

$$\Pr_{\mathcal{T} \sim \mathcal{D}}[\exists o \in \mathcal{O} : (T(x), o) \in R \wedge T_o(x) = \perp] \leq \varepsilon$$

Theorem 3.4. If a proof system \mathcal{P} is characterized by the total search problems reducible to $R \in \text{TFNP}^{dt}$, then $r\mathcal{P}$ is characterized by the total search problems that are randomized-reducible to R .

The intuition for this theorem is most clear in the case of randomized reductions to FP^{dt} (which is TFZPP^{dt}) and random tree-like resolution. This is also the case that we will use to derive consequences about TFZPP^{dt} . We leave the proof of [Theorem 3.4](#) to the [Appendix B](#).

We remark that [Theorem 3.4](#) reduces the task of showing that a TFNP class is closed under randomized reduction to showing that the corresponding proof system is *closed* in the sense that $\mathcal{P} = r\mathcal{P}$. We are not aware of any such proof system, and it would be interesting to exhibit one.

Lemma 3.5. There is a quasi-polynomial size random tree-like resolution proof of $F = C_1 \wedge \dots \wedge C_m$ iff $\text{Search}_F \in \text{TFZPP}^{dt}$.

Proof. Suppose that $\text{SEARCH}_F \in \text{TFZPP}^{dt}$ and let \mathcal{D} be a distribution over depth- d decision trees solving SEARCH_F as in the definition of TFZPP^{dt} . We construct a ε -error randomized tree resolution proof \mathcal{P} , which is defined by the following sampling procedure:

1. Sample $T \sim \mathcal{D}$.
2. Let B be the set of clauses obtained by taking the negation of each root-to-leaf path in T ending in \perp ,

$$B := \{\neg p : p \in T \text{ is a root-to-}\perp \text{ path}\},$$

where we think of a path as the conjunctions of the literals that appear along it (a queried variable is a positive literal if p took the 1-branch, and a negative literal if p took the 0-branch).

3. To construct Π , we will use the equivalence between a decision tree solving the false-clause search problem and tree-resolution proofs. Let T^* be obtained from T by relabeling each path p ending in \perp by the clause $\neg p \in B$. Let $F \cup B$ be the CNF formula whose clauses are the clauses of F and those in B , and observe that T^* is a depth- d decision tree solving $\text{SEARCH}_{F \cup B}$. Thus, there is a depth- d tree resolution proof Π of $F \cup B$.

As B states that we do not follow any root-to- \perp path in T , for any $x \in \{0, 1\}^n$, $T(x) \neq \perp$ iff x satisfies all of B . Therefore, $\Pr_{T \sim \mathcal{D}}[T(x) = \perp] \leq \varepsilon$ implies that $\Pr_{(B, \Pi) \sim \mathcal{P}}[B(x) = 1] \geq 1 - \varepsilon$ for every $x \in \{0, 1\}^n$.

In the other direction, suppose that \mathcal{P} is a ε -random tree resolution proof of F with complexity c . We construct a distribution \mathcal{D} over decision trees solving SEARCH_F by the following sampling procedure:

1. Sample $(\Pi, B) \sim \mathcal{P}$.
2. Let T^* be the depth- d decision tree solving $\text{SEARCH}_{F \cup B}$ obtained from Π obtained by the equivalence between tree resolution proofs and decision trees in the same manner as in point (3) above. It is well-known that the depth of a resolution proof is bounded by its width, and hence T^* has depth at most c .
3. Let T be the decision tree obtained from T^* by relabeling each leaf of Π that is labeled by a clause in B by \perp .

As for any $x \in \{0, 1\}^n$, $\Pr_{(B, \Pi) \sim \mathcal{D}}[B(x) = 1] \geq 1 - \varepsilon$, we have that $\Pr_{T \sim \mathcal{D}}[T(x) = \perp] \leq \varepsilon$. \square

3.1 Separations

We now use [Lemma 3.5](#) to show that TFZPP^{dt} is not contained within any uniformly generated TFNP^{dt} class unless NP is contained within quasi-polynomial time (QP).

Theorem 3.6. $\text{TFZPP}^{dt} \not\subseteq \mathcal{C}$ for any uniformly generated class $\mathcal{C} \subseteq \text{TFNP}^{dt}$ unless $\text{NP} \subseteq \text{QP}$.

This theorem follows by combining the characterization of uniform TFNP^{dt} classes by proof systems of Buss et al. [[BFI23](#)], [Lemma 3.5](#), and the following result of Pudlák and Thapen [[PT19](#)].

This separation relies on a theorem of Pudlák and Thapen [[PT19](#)] who showed that random resolution cannot be simulated by any propositional proof system unless $\text{P} \neq \text{NP}$. A straightforward examination of their theorem reveals that it also holds for tree-like resolution and can be stated in the following form.

Theorem 3.7 (Proposition 10 in [[PT19](#)]). There is a family of unsatisfiable 3-CNFs \mathcal{F} such that:

1. There are $O(\log(n))$ -complexity random tree-like resolution proofs of \mathcal{F} .
2. If there is a propositional proof system which has $\text{poly} \log(n)$ -complexity proofs of \mathcal{F} then $\text{NP} \subseteq \text{QP}$.

Indeed, to prove their theorem Pudlák and Thapen observe that random (treelike) resolution has small proofs of any highly unsatisfiable formula (one for which any assignment falsifies many clauses), and that the PCP theorem can be used to put any 3-CNF formula into this form (the family \mathcal{F}). Using this, they show that if there existed a propositional proof system which could (quasi-polynomially) simulate random tree-like resolution, then one could use its variability to decide SAT.

Combining this theorem with [Lemma 3.5](#) and the characterization of TFNP^{dt} classes by propositional proof systems due to Buss et al. [[BFI23](#)] proves [Theorem 3.6](#). Say that $R = \{R_n\} \in \text{TFNP}^{dt}$ is *uniformly generated* if there is a Turing Machine which on input 1^n outputs R_n , and say that a class is uniformly generated if it has a uniformly generated complete problem. Note that all major TFNP^{dt} subclasses are uniformly generated.

Proof of Theorem 3.6. Let \mathcal{C} be a uniformly generated class such that $\text{TFZPP}^{dt} \subseteq \mathcal{C}$. Buss et al. [BF123] showed that every uniformly generated TFNP^{dt} subclass is characterized by a proof system; let \mathcal{P} be the system for \mathcal{C} . Consider the family of formulas \mathcal{F} from Theorem 3.7. As $\text{TFZPP}^{dt} \subseteq \mathcal{C}$, there are $\text{poly log}(n)$ -complexity \mathcal{P} -proofs of \mathcal{F} . Hence, Theorem 3.7 implies that $\text{NP} \subseteq \text{QP}$. \square

3.2 Explicit Separations

The separating examples in Theorem 3.6 rely on an unproven hypothesis. We end this section by proving explicit separations between TFZPP^{dt} and every major TFNP^{dt} subclass which do not rely on any unproven assumptions. A separation of PPA^{dt} from TFZPP^{dt} was implicitly shown by Beame et. al. [BCE+98], who proved Nullstellensatz lower bounds for LOSSY-CODE.⁸ The remaining major TFNP^{dt} classes are contained within PLS^{dt} and PPP^{dt} . We show the following.

Theorem 3.8. There exist explicit total search problems in TFZPP^{dt} which are not contained in PLS^{dt} nor PPP^{dt} .

It is known that if a total search problem SEARCH_F is in PLS^{dt} or PPP^{dt} , then F has a small low-degree *Sum-of-Squares* (SoS) proof; see [FKP19] for an exposition on this proof system. Our hard instance is based on the recent work of Hopkins and Lin [HL22], who exhibited the first explicit hard 3-XOR instance for SoS.

Theorem 3.9 ([HL22]). There exist constants $\mu_1, \mu_2 \in (0, 1)$ and a polynomial time algorithm which, given 1^n as input, outputs a 3-XOR formula $F = C_1 \wedge \dots \wedge C_m$ on n variables such that:

- For every $x \in \{0, 1\}^n$, $\Pr_{i \sim [m]}[C_i(x) = 1] \leq 1 - \mu_1$.
- Any Sum-of-Squares refutation of F requires degree at least $\mu_2 n$.

Proof of Theorem 3.8. Let F be as in Theorem 3.9. We show that $R := \text{SEARCH}_F$ satisfies the desired properties. We first prove $R \in \text{TFZPP}^{dt}$. Consider the following simple algorithm: sample $i \sim [m]$ uniformly at random, and make three queries to check if $C_i(x) = 0$. If so, output i ; otherwise, output \perp . By the first item of Theorem 3.9, the algorithm succeeds with probability at least μ_1 . Repeating the procedure $O(1/\mu_1)$ times boosts the success probability to at least $2/3$.

To separate R from PPP^{dt} and PLS^{dt} , we only need to show there is no efficient black-box reduction from R to a complete problem for one of these classes. If there was, then there would be an efficient SoS proof of F , contradicting the second item of Theorem 3.9. \square

4 NEPHEW

Recall the NEPHEW problem, which is our main candidate for a total search problem in TFZPP but not reducible to LOSSY-CODE:

NEPHEW. Given a set V of vertices and two functions $f : V \rightarrow V$ and $g : V \rightarrow V$. Think of $f(v)$ as the *father* of v and $g(v)$ as the *nephew* of v . A solution is one of the following.

- s1.** $v \in V$ such that $f(f(g(v))) \neq f(v)$ (your nephew's grandparent is not your parent)

⁸More specifically, Beame et. al. [BCE+98, Theorem 12] proved the Nullstellensatz degree lower bounds for WEAK-PIGEON. They also showed that any Nullstellensatz degree lower bounds for WEAK-PIGEON implies the same Nullstellensatz degree lower bounds for LOSSY-CODE in [BCE+98, Lemma 10] (see also Definition 3.1, 3.2 in [BCE+98] for the definition of LOSSY-CODE and WEAK-PIGEON).

s2. $v \in V$ such that $f(g(v)) = v$ (you are your nephew's parent)

The main result of this section is the inclusion theorem:

Theorem 1.7. $\text{NEPHEW} \in \text{TFZPP} \cap \text{PWPP}$.

The intuition behind [Theorem 1.7](#) is as follows:

- We can treat certain vertices in a NEPHEW instance like the root of a directed binary tree, where the leaves of the tree correspond to solutions of the NEPHEW instance.
- Finding a leaf of a rooted binary tree is easy for both PWPP and TFZPP computations.

Once we have proven the above, we are nearly done. First, choose an arbitrary vertex. Perhaps that vertex is one of the many that we can treat as the root of a binary tree, and so from it we can find a solution. Otherwise, we use a procedure to find such a root vertex. In fact, we will be able to find two vertices, one of which must be a good root vertex. To show inclusion in TFZPP, we just need to pick one of these two randomly. For PWPP, we will have to consider both. We make an adjustment to the argument for a single root vertex so that it works for two potential root vertices.

To be more concrete, we will reduce NEPHEW to the following *promise* search problem.

LEAF-OF-ROOTED-TREE. An instance consists of a set V of vertices, a special vertex v^* , and two functions $L : V \rightarrow V \cup \{\perp\}$ and $R : V \rightarrow V \cup \{\perp\}$. We define a subset $V^* \subseteq V$ recursively: $v^* \in V^*$ and, for every $v \in V^*$, we add $L(v)$ to V^* if $L(v) \neq \perp$ and $R(v)$ to V^* if $R(v) \neq \perp$. We promise that the induced subgraph on V^* is a (directed) tree rooted at v^* and that for all $v \in V^*$ either:

- $L(v) = R(v) = \perp$, or
- $L(v) \neq \perp$ and $R(v) \neq \perp$ and $L(v) \neq R(v)$.

A solution is a $(\lceil \log |V| \rceil + 1)$ -length path, represented by a string $p \in \{L, R\}^{\lceil \log |V| \rceil + 1}$, where starting at v^* and descending by the functions specified by the characters of p in order will at some point reach a vertex v where $L(v) = R(v) = \perp$.

Note that instead of simply asking for a leaf, we require a root-to-leaf path for a solution. This is to confirm that the leaf is in the binary tree rooted at v^* .

Finding a root-to-leaf path in a rooted binary tree. It is easy for a PWPP or TFZPP computation to find a solution to LEAF-OF-ROOTED-TREE. This follows from the simple observation that there are (many) more paths of length $(\lceil \log |V| \rceil + 1)$ than vertices in the tree, so most paths must be solutions.

Lemma 4.1. A solution to LEAF-OF-ROOTED-TREE can be found with high probability using a randomized algorithm.

Proof. The algorithm guesses a random path of length $\lceil \log |V| \rceil + 1$, which will be a solution with probability at least $5/6$. This is because if a path p is not a solution, then following p reaches a vertex v_p with two children. The path p is the only non-solution path of length $\lceil \log |V| \rceil + 1$ to reach v_p ; furthermore, because V^* is an induced tree, no other path ends at a vertex with $L(v_p)$ or $R(v_p)$ as children. This means that the set $\{v_p, L(v_p), R(v_p)\}$ are uniquely reached by p out of all of the non-solution paths. Therefore, there are at least 3 times as many vertices in V as there are non-solution paths. Let the fraction of non-solution paths be α . Then

$$|V| \geq 3\alpha \left| \{L, R\}^{\lceil \log |V| \rceil + 1} \right| \geq 3\alpha \cdot 2|V|,$$

and so $\alpha \leq 1/6$. □

To show inclusion in PWPP, we reduce to the PWPP-complete problem WEAK-PIGEON. (Note that this is not a reduction between TFNP problems, as LEAF-OF-ROOTED-TREE is a promise problem.) We recall its definition here:

WEAK-PIGEON. Given $h : \{0, 1\}^n \rightarrow \{0, 1\}^{n-1}$ a solution is $x \neq y$ such that $h(x) = h(y)$.

Lemma 4.2. LEAF-OF-ROOTED-TREE reduces to WEAK-PIGEON.

Proof. Let $n = \lceil \log |V| \rceil + 1$. Let v_p be the vertex reached by p if p is a non-solution path. Then we define h as a map from $(\lceil \log |V| \rceil + 1)$ -length paths to vertices:

$$h(p) = \begin{cases} v_p & \text{if } p \text{ is not a solution;} \\ v^* & \text{otherwise.} \end{cases}$$

(Recall that since v^* is the root, we have that $v_p \neq v^*$ for any path p .) Thus, paths (x, y) are a collision if and only if x and y are both solutions to LEAF-OF-ROOTED-TREE, and so from any collision we can find a solution to LEAF-OF-ROOTED-TREE by arbitrarily choosing one from the pair. □

The structure of NEPHEW instances and finding a rooted binary tree. For any NEPHEW instance (V, f, g) , let G_f be the directed graph with vertex set V and where (u, v) is an edge if and only if $v = f(u)$. Then G_f is a directed graph with out-degree one, and therefore the connected components of G_f have a simple structure: they are composed of a cycle (perhaps a self-loop) and trees (with edges oriented leaf-to-root) that are rooted at vertices in the cycle. For any vertex $v \in V$, define its *level* (denoted $\ell(v)$) as the distance from v to any vertex on the cycle of its connected component. So, any vertex on the cycle has level 0, any non-cycle vertex pointing to a cycle vertex has level 1, and so on. See Figure 2 for an illustration.

We give a reduction from NEPHEW to LEAF-OF-ROOTED-TREE under the assumption that we can find a vertex v^* with $\ell(v^*) \geq 2$. After proving this, the hard part will be finding such a vertex. The main component of this reduction is the procedure **Find-Children** (Algorithm 1), which is based on the functions f and g from an NEPHEW instance. Define **CheckSol**(u) to be the procedure that returns **True** iff u is a solution to the NEPHEW instance, that is, iff $f(f(g(u))) \neq f(u)$ or $f(g(u)) = u$.

Algorithm 1 Procedure **Find-Children** _{f, g} (v)

```

1:  $h(v) \leftarrow g(f(g(v)))$   $\triangleright$  Rename for notational brevity
2: if CheckSol( $v$ )  $\vee$  CheckSol( $g(v)$ )  $\vee$  CheckSol( $f(g(v))$ )  $\vee$  CheckSol( $h(v)$ ) then
3:   return  $(\perp, \perp)$   $\triangleright$  We have found a solution, so we can stop here
4: else
5:   return  $(g(v), h(v))$ 
```

See Figure 3 for an illustration. The idea is to construct the tree by defining the left and right children of v to be two nodes reachable from v , unless the procedure finds a nearby solution, in which case we can make v into a leaf, as it is easy to compute a solution given v .

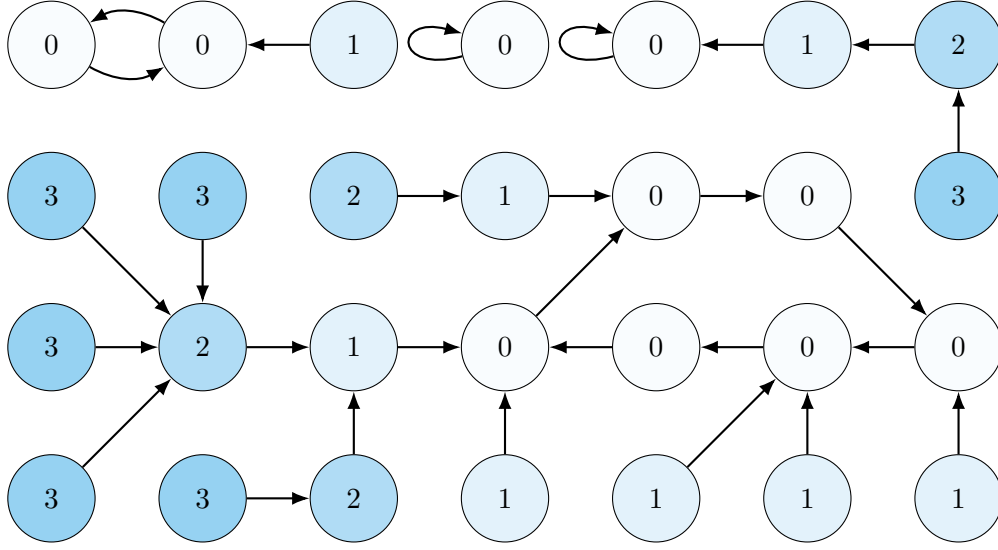


Figure 2: An example G_f with levels marked.

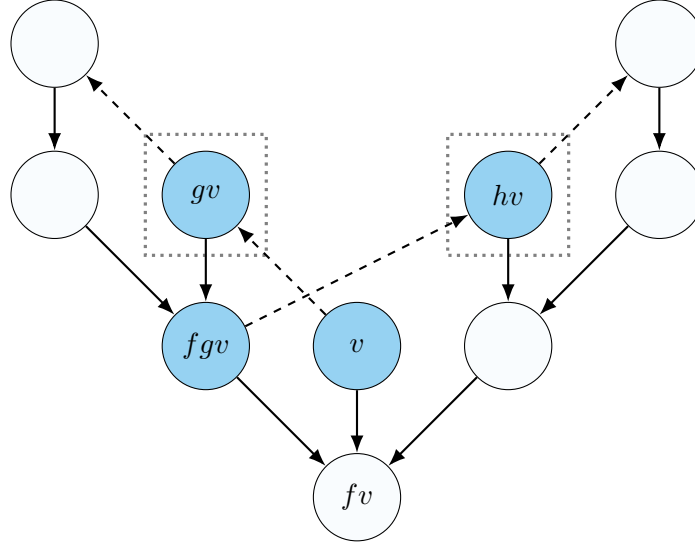


Figure 3: The procedure performed by $\text{Find-Children}_{f,g}(v)$. Solid arrows represent f and dashed arrows represent g . Parentheses are omitted in labels. The dotted boxes indicate that vertices $g(v)$ and $h(v)$ will be returned as the children of v . The procedure will check if the shaded vertices are NEPHEW solutions, and in doing so will visit the unshaded vertices (but will not detect if these are solutions). Note that $f(h(v)) = v$ is possible, but $f(h(v)) = f(g(v))$ is not.

Although the intuition behind the reduction is straightforward, some work needs to be done to show that **Find-Children** gives a valid binary tree. The following properties will help.

Lemma 4.3. Let $\text{Find-Children}_{f,g}(v) = (a, b)$. If $(a, b) \neq (\perp, \perp)$, then

- (i) $a \neq b$,
- (ii) $f(a) \neq f(b)$,
- (iii) $f(f(a)) = f(f(b)) = f(v)$, and
- (iv) if $\ell(v) \geq 2$, then $\ell(a) = \ell(b) = \ell(v) + 1$.

Proof.

- (i) Since $(a, b) \neq (\perp, \perp)$, we have that $f(g(v)) = f(a)$ is not an NEPHEW solution, and in particular $f(g(f(a))) \neq f(a)$. This means that $a \neq b$, as $b = g(f(a))$.
- (ii) As $f(g(v))$ is not a solution, applying $f(g(u)) \neq u$ to $u = f(g(v))$ gives $f(b) = f(g(f(g(v)))) \neq f(g(v)) = f(a)$.
- (iii) As v is not a solution, $f(f(a)) = f(f(g(v))) = f(v)$. As $f(g(v))$ is not a solution, applying $f(f(g(u))) = f(u)$ for $u = f(g(v))$ gives $f(f(b)) = f(f(g(f(g(v)))) = f(f(g(v))) = f(v)$.
- (iv) Under the assumption that $\ell(v) \geq 2$, we know that $\ell(f(v)) = \ell(v) - 1 > 0$, which implies that any vertex with $f(f(u)) = f(v)$ has $\ell(u) = \ell(f(v)) + 2 = \ell(v) + 1$. By [Item \(iii\)](#), this applies to a and b . \square

Remark 4.4. [Lemma 4.3 \(iv\)](#) does not hold without the hypothesis $\ell(v) \geq 2$, as if $\ell(f(v)) = 0$ there is no guarantee that vertices with edges pointing to $f(v)$ have level greater than 0. This is why it is necessary to assume $\ell(v^*) \geq 2$ for the simple reduction given here.

Lemma 4.5. Let (V, f, g) be an instance of NEPHEW. Then define L and R by

$$(L(v), R(v)) \leftarrow \text{Find-Children}_{f,g}(v).$$

Given $v^* \in V$ with $\ell(v^*) \geq 2$, (V, v^*, L, R) is a valid instance to LEAF-OF-ROOTED-TREE.

Proof. All we need to show is that v^* is the root of an induced tree. Recall that V^* denotes the set of vertices reachable from v^* via L and R (i.e., using edges of the form $(u \rightarrow L(u))$ and $(u \rightarrow R(u))$). By [Lemma 4.3 \(iv\)](#), the induced subgraph on V^* can be assigned levels such that edges are directed only from lower levels to higher levels, i.e. it is a DAG. By [Lemma 4.3 \(i\)](#), the DAG has outdegree 2. To prove this induced DAG is indeed a tree, it suffices to show that no two vertices share the same child. Indeed, if there are vertices $u, u' \in V^*$ such that $\{L(u), R(u)\} \cap \{L(u'), R(u')\} \neq \emptyset$, then by [Lemma 4.3 \(iii\)](#), we have $f(u) = f(u')$. Hence, it suffices to prove the following claim:

Claim 4.6. If $u, u' \in V^*$ are two different vertices, then $f(u) \neq f(u')$.

Proof of Claim 4.6. We may assume that $\ell(u) = \ell(u')$, as otherwise $f(u) \neq f(u')$ by the observation about levels in V^* . The proof is by induction on levels. At level $\ell(v^*)$, there is only one vertex, hence the base case is trivially true. Assume that at level k (where $k \geq \ell(v^*)$), no two vertices in V^* map via f to the same vertex. We will prove that it also holds for level $k + 1$.

For two different vertices u, u' with $\ell(u) = \ell(u') = k + 1$, we may assume u and u' are the children of two different vertices in V^* : if they are children of the same vertex, by [Lemma 4.3 \(ii\)](#) $\{u, u'\} = \{L(w), R(w)\}$ implies $f(u) \neq f(u')$ and we are done. So, let $w \neq w'$ be such that

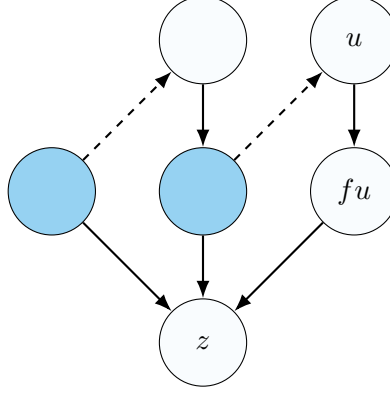


Figure 4: The argument behind [Claim 4.6](#). Solid arrows represent f and dashed arrows represent g . Parentheses are omitted in labels. The shaded vertices are possible locations of the $w \in V^*$ such that either $L(w) = u$ or $R(w) = u$.

$u \in \{L(w), R(w)\}$ and $u' \in \{L(w'), R(w')\}$. Then $\ell(w) = \ell(w') = k$ and by the inductive hypothesis $f(w) \neq f(w')$.

Assume that $f(u) = f(u')$. We will prove shortly that all vertices in V^* that map to u or u' by L or R map to some common vertex z by f , a contradiction with $f(w) \neq f(w')$. Indeed, set $z = f(f(u)) = f(f(u'))$. If a vertex $x \in V^*$ has $L(x) = g(x) = u$ (the same arguments will apply to x that maps to u'), then

$$z = f(f(u)) = f(f(g(x))) = f(x).$$

If a vertex $x \in V^*$ has $R(x) = g(f(g(x))) = u$, then

$$z = f(f(u)) = f(f(g(f(g(x)))))) = f(f(g(x))) = f(x),$$

where we have applied $f(f(g(y))) = f(y)$ to $y = f(g(x))$ which we know is not a NEPHEW solution because it was checked by **Find-Children**. \diamond

See [Figure 4](#) for an illustration of the argument behind [Claim 4.6](#). \square

Completing the argument. It remains to find a vertex v^* with $\ell(v^*) \geq 2$. We do not know how to achieve this deterministically; instead, we will find a pair of vertices (v, v') such that *either* $\ell(v) \geq 2$ or $\ell(v') \geq 2$.

Lemma 4.7. Let $u \in V$ be an arbitrary starting vertex. Let $v = g(u)$ and $v' = g(f(u))$. Then either

- at least one of $\{u, v, v'\}$ is a solution, or
- at least one of $\{v, v'\}$ is at level at least 2.

Proof. Assume that there are no solutions among $\{u, v, v'\}$. There are three cases:

- **Case I:** Suppose that $\ell(u) = 0$. We claim that $\ell(v) = 2$ in this case. Indeed, u is the only vertex at level 0 that maps via f to $f(u)$. Since $f(f(v)) = f(f(g(u))) = f(u)$ but $f(v) = f(g(u)) \neq u$, $f(v)$ has to be at level 1, hence v is at level 2.
- **Case II:** Suppose that $\ell(u) = 1$. In this case, we have $\ell(f(u)) = 0$, hence the same argument as above applies to show that $v' = g(f(u))$ is at level 2.

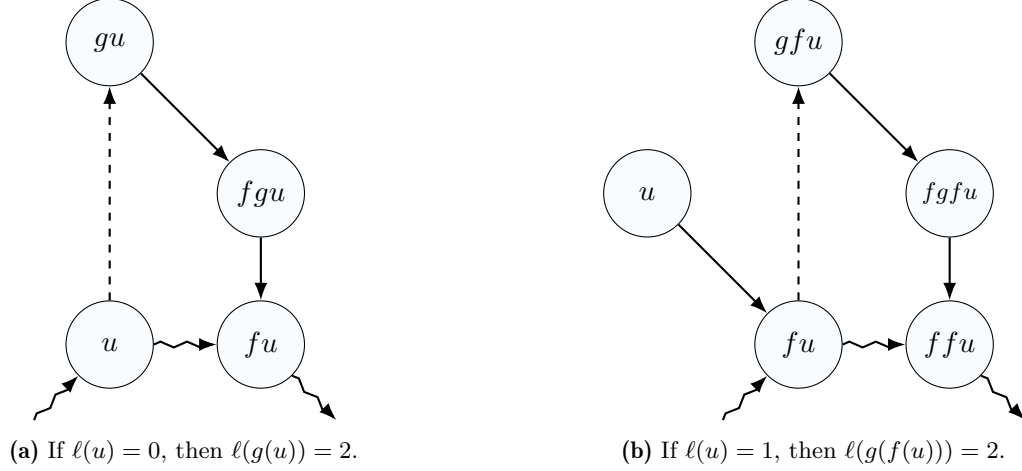


Figure 5: Illustration of cases I and II of Lemma 4.7. Solid arrows represent f and dashed arrows represent g . Zigzag f arrows are between vertices of level 0. Parentheses are omitted in labels.

- **Case III:** Suppose that $\ell(u) \geq 2$. In this case, $f(u)$ is at level at least 1. Since $f(f(v)) = f(f(g(u))) = f(u)$, we have that v is at level at least 2. \square

See Figure 5 for an illustration of cases I and II of Lemma 4.7.

Now we are ready to prove our main results. Recall the statement of Theorem 1.7:

Theorem 1.7. $\text{NEPHEW} \in \text{TFZPP} \cap \text{PWPP}$.

Proof. The proof of $\text{NEPHEW} \in \text{TFZPP}$ is easy. Pick an arbitrary vertex u and define $v = g(u)$ and $v' = g(f(u))$. By Lemma 4.7, if none of u, v, v' are solutions of NEPHEW, then at least one of v and v' is at level at least 2. Therefore, we can randomly select one vertex in $\{v, v'\}$ as the root v^* and run the randomized algorithm for LEAF-OF-ROOTED-TREE on the instance (V, v^*, L, R) . The correctness is guaranteed by Lemma 4.5.

Now we prove that $\text{NEPHEW} \in \text{PWPP}$. Intuitively, we will run the reduction of Lemma 4.5 in parallel on (the direct product of) two graphs where the root is v in one graph and is v' in the other. Because we can only guarantee that our starting point is at level ≥ 2 in only one of these graphs, we only know that one of the graphs has an induced tree rooted at its respective starting point by the reduction process. Fortunately, the overall graph on $V \times V$ will have a rooted induced tree.

By Lemma 4.7, starting with an arbitrary vertex, we can obtain either a NEPHEW solution (in which case we are done) or a pair (v, v') where at least one is at level at least 2. We will reduce to LEAF-OF-ROOTED-TREE on the vertex set $V \times V$. Set

$$(A(u, u'), B(u, u')) \leftarrow \text{Find-Children}_{f,g}(u),$$

$$(C(u, u'), D(u, u')) \leftarrow \text{Find-Children}_{f,g}(u').$$

Set $L(u, u') = (A(u, u'), C(u, u'))$, unless one of $A(u, u'), C(u, u')$ is \perp , in which case $L(u, u') = \perp$. Set $R(u, u') = (B(u, u'), D(u, u'))$, again unless one side of the pair is \perp in which case $R(u, u') = \perp$. By the construction of Find-Children, if one of L, R is set to \perp the other one will as well. The special vertex in $V \times V$ will be (v, v') .

Then $V^* \subset V \times V$ induces a binary tree (recall that V^* is the set of vertices reachable from (v, v') by L, R). Say that $\ell(v) \geq 2$; the other case follows by symmetry. Starting at (v, v') , L and R will yield a tree structure on the first member of the pair by the same argument as Lemma 4.5.

Let p, p' be two distinct paths of any length which start at (v, v') and traverse with L and R . The vertex reached by p will have a different first pair element than the vertex reached by p' (by the tree structure on the first element of the pair), and thus p and p' arrive at distinct vertices. This means that there are no (undirected) cycles in the induced graph on V^* , and so it is a tree. \square

5 Finding a Leaf in a Binary Tree

In the previous section, we reduced NEPHEW to the promise problem LEAF-OF-ROOTED-TREE, which captures the following principle:

Randomly walking down a binary tree will reach a leaf in logarithmic time in expectation.

In this section, we study a family of *total* (instead of promise) search problems that capture the same principle. We then study the power of these search problems.

The problem EMPTY-CHILD captures the task of finding a leaf in a tree where each non-leaf vertex v has at least two different children $L(v)$ and $R(v)$. This is made total (i.e. we do not need the promises as in LEAF-OF-ROOTED-TREE) by adding a parent function, denoted $F(v)$.

EMPTY-CHILD. The input is a set of vertices V and three functions $F, L, R : V \rightarrow V$, where $F(u)$ is the father of u , and $L(u), R(u)$ are the left and the right child of u , respectively. There are two possible solutions:

- s1.** $u \in V$ such that $F(L(u)) \neq u$ or $F(R(u)) \neq u$ or $L(u) = R(u) \neq u$. **(Empty Child)**
- s2.** 1, if $L(1) = 1$ or $R(1) = 1$ or $F(1) \neq 1$. **(Wrong Root)**

Intuition behind EMPTY-CHILD. We think of EMPTY-CHILD as defining a directed graph on V . Draw an edge from v to w if $F(w) = v$ and either $L(v) = w$ or $R(v) = w$. If v does not point to two distinct vertices in this graph, then v has an “empty child” and v is a leaf⁹ of the graph.

The directed graph defined above may have multiple connected components. They will be of the following sorts:

- Isolated vertices. We may have $F(v) = L(v) = R(v) = v$.
- Trees.
- Cycles with trees rooted at each node in the cycle.

Unless there is a solution of type **(Wrong Root)**, we are forced to have at least one tree, rooted at vertex 1. This forces the graph to have structures other than just isolated vertices. Then we can find a leaf of that tree using the principle captured by LEAF-OF-ROOTED-TREE. An important observation is that in the third type of structure—cycles with trees—walking randomly will also reach a leaf in logarithmic time. We will show shortly that solution **(Wrong Root)** can be relaxed to allow for graphs without trees, as long as at least one cycle-with-trees is present.

In EMPTY-CHILD, it is possible that there are vertices v, w such that $F(w) = v$ but $L(v) \neq w$ and $R(v) \neq w$. This is fine: structurally, we can interpret w as being the root of a new tree. We will consider a variant of EMPTY-CHILD where such a situation is forbidden.

In LEAF-OF-ROOTED-TREE, we needed a root-to-leaf path to prove that the leaf vertex was in the promised tree. In contrast, here we enforce the tree structure syntactically, and hence do not require a path. Indeed, a solution to EMPTY-CHILD need not be in the connected component that contains the canonical root vertex.

⁹We slightly stretch the definition of leaf to include vertices with only one child. Equivalently, we could redefine the graph such that a vertex may only have two or zero children.

Results about EMPTY-CHILD. The main result of this section is the LOSSY-completeness of EMPTY-CHILD:

Theorem 5.1. EMPTY-CHILD is equivalent to LOSSY-CODE.

The problem from the previous section, NEPHEW, is at least as strong as EMPTY-CHILD: we give a reduction in [Section 5.4](#). We conjecture that NEPHEW is strictly more powerful than EMPTY-CHILD. A variant of NEPHEW is considered in [Section 5.5](#) and is shown to be equivalent to EMPTY-CHILD.

Variants of EMPTY-CHILD. For a proof later in this section it will be useful to weaken ([Wrong Root](#)). Instead of enforcing that a *root* vertex exists, we only enforce that an *internal* vertex that is not a self-loop exists.

EMPTY-CHILD’. Given the same inputs as in EMPTY-CHILD, we accept the solutions ([Empty Child](#)) and:

s2a. 1, if $L(1) = 1$ or $R(1) = 1$. ([Wrong Internal Vertex](#))

It turns out that EMPTY-CHILD’ is equivalent to EMPTY-CHILD in terms of decision tree reductions, but we do not know of a direct way to prove this. Instead, we will use LOSSY-CODE as an intermediate step in the reduction.

Theorem 5.2. EMPTY-CHILD and EMPTY-CHILD’ are equivalent under black-box reductions.

It is easy to see that EMPTY-CHILD reduces to EMPTY-CHILD’, as the difference between the two is just a weakening of one of the solutions. In [Section 5.1](#), we will prove that EMPTY-CHILD’ reduces to LOSSY-CODE ([Lemma 5.3](#)). In [Section 5.3](#), we will prove that LOSSY-CODE reduces to EMPTY-CHILD ([Theorem 5.9](#)), completing the proof of [Theorem 5.2](#).

We also consider a stricter variant where the F function and the L and R functions are required to “agree”:

BINARY-EMPTY-CHILD. Given the same inputs as in EMPTY-CHILD but in addition to ([Empty Child](#)) and ([Wrong Root](#)), we also accept the following solution:

s3. a vertex $u \in V \setminus \{1\}$ such that $u \notin \{L(F(u)), R(F(u))\}$. ([Wrong Father](#))

Finally, we will consider variants of EMPTY-CHILD with added *height* functions in [Section 5.2](#). The analysis of these will be a warm-up for the proof that LOSSY-CODE reduces to EMPTY-CHILD in [Section 5.3](#).

5.1 EMPTY-CHILD reduces to LOSSY-CODE

It is easy to see that EMPTY-CHILD is in TFZPP: Start from the root 1 and walk down the tree using L or R randomly until we reach a leaf. In fact, the same proof reduces EMPTY-CHILD (in fact, also EMPTY-CHILD’) to LOSSY-CODE:

Lemma 5.3. EMPTY-CHILD’ reduces to LOSSY-CODE via a black-box reduction.

Proof. Consider an instance of EMPTY-CHILD’ (V, F, L, R) and let $\ell := \lceil \log |V| \rceil$. We construct a LOSSY-CODE instance (f, g) with $N := 2^\ell$. Intuitively, $f : [2N] \rightarrow [N]$ maps a tree-path of length $\ell + 1$ to a vertex in u by travelling downwards in the tree, and $g : [N] \rightarrow [2N]$ maps a vertex in V to a tree-path by travelling upwards.

More formally, identify $[2N]$ with $\{L, R\}^{\ell+1}$, the space of length- $(\ell+1)$ strings over the alphabet $\{L, R\}$. Every such string $x \in \{L, R\}^{\ell+1}$ corresponds to a path of length $\ell+1$ starting from the root: Let $v_0 := 1$ be the root and for each $i \geq 1$,

$$v_i := \begin{cases} L(v_{i-1}) & \text{if } x_i = L; \\ R(v_{i-1}) & \text{if } x_i = R. \end{cases}$$

Then, x corresponds to the path $p_x := (v_0, v_1, \dots, v_{\ell+1})$. We define $f(x)$ to be $v_{\ell+1}$, the endpoint of this path. For instance, $f(\text{LLR}) = R(L(L(1)))$ when $\ell = 2$.

Conversely, for any $u \in [N]$, we can construct a string $x \in \{L, R\}^{\ell+1}$ by traversing $\ell+1$ steps upwards from u : in the i -th step, we set $x_{\ell+2-i} \leftarrow L$ if $u = L(F(u))$, or $x_{\ell+2-i} \leftarrow R$ otherwise¹⁰; then update $u \leftarrow F(u)$. We set $g(u) = x$.

Now we prove the correctness of the reduction. Let $x \in [2N]$ be such that $g(f(x)) \neq x$, we use a case analysis to find a solution of $\text{EMPTY-CHILD}'$ from x . First we find the path $p_x = (v_0, v_1, \dots, v_{\ell+1})$ as defined above. Normally, two adjacent vertices in the path should be distinct (since a node should be different from its child); we deal with the abnormal case before proceeding. Suppose there exists some $i \geq 0$ such that $v_i = v_{i+1}$, and let i be the smallest such index.

- If $i = 0$, then either $L(1) = 1$ or $R(1) = 1$, hence 1 is a **(Wrong Internal Vertex)**.
- Otherwise, we have $v_{i-1} \neq v_i$. Without loss of generality, let us assume that $v_i = L(v_{i-1})$ and $v_{i+1} = L(v_i)$ ($= v_i$); the cases where we walk down the right child are completely symmetric.
 - If $F(v_i) \neq v_{i-1}$, then $F(L(v_{i-1})) \neq v_{i-1}$ and hence v_{i-1} has an **(Empty Child)**.
 - Otherwise, $F(L(v_i)) = F(v_i) = v_{i-1} \neq v_i$ and hence v_i has an **(Empty Child)**.

Now we are in the “normal” case where for every $i \geq 0$, $v_i \neq v_{i+1}$. Let $x' := g(f(x))$. Recall that the precise definition of x' is as follows: let $u_{\ell+1} := v_{\ell+1}$ ($= f(x)$), and let $u_i := F(u_{i+1})$ for each i from ℓ down to 0, then $x'_i = 0$ if $u_i = L(F(u_i))$ and $x'_i = 1$ otherwise.

Let i be the largest index such that either $x'_i \neq x_i$ or $v_{i-1} \neq u_{i-1}$. (Note that such i exists because $x' = g(f(x)) \neq x$.) Then $v_i = u_i$. We claim that v_{i-1} has an **(Empty Child)**, which would finish the proof.

- If $v_{i-1} \neq u_{i-1} = F(v_i)$, then $v_{i-1} \neq F(L(v_{i-1}))$ or $v_{i-1} \neq F(R(v_{i-1}))$ depending on whether x_i is 0 or 1. In either case, v_{i-1} has an **(Empty Child)**.
- If $x_i = 0$ and $x'_i = 1$, then $v_i = L(v_{i-1})$ but $u_i \neq L(F(u_i))$ (hence $v_i \neq L(F(v_i))$). Note that this implies that $v_{i-1} \neq F(v_i)$, hence $F(L(v_{i-1})) = F(v_i) \neq v_{i-1}$ and v_{i-1} has an **(Empty Child)**.
- If $x_i = 1$ and $x'_i = 0$, then $v_i = R(v_{i-1})$ but $u_i = L(F(u_i))$ (hence $v_i = L(F(v_i))$). This implies that $F(R(v_{i-1})) = F(v_i)$. If $F(v_i) \neq v_{i-1}$, then v_{i-1} has an **(Empty Child)**. Otherwise we have $L(v_{i-1}) = R(v_{i-1}) = v_i$ and $v_{i-1} \neq v_i$ (by the reasoning above) and v_{i-1} has an **(Empty Child)**. \square

5.2 Finding a Leaf with Heights

Before showing that EMPTY-CHILD is complete for LOSSY , we take a detour to study variants of EMPTY-CHILD with *heights*. Indeed, the proof that EMPTY-CHILD is LOSSY -complete is inspired by the investigations of these variants. We first define the variants of EMPTY-CHILD and $\text{BINARY-EMPTY-CHILD}$ with heights:

¹⁰It is possible that $u \notin \{L(F(u)), R(F(u))\}$.

EMPTY-CHILD-W-HEIGHT. The input is a set V of vertices and four functions $F, L, R : V \rightarrow V$, $H : V \rightarrow [|V|]$. Besides **(Empty Child)** and **(Wrong Root)** listed in the definition of EMPTY-CHILD, the following solution is also valid:

- s4. a vertex $u \in V \setminus \{1\}$ such that $u \neq F(u)$ and $H(u) \neq H(F(u)) + 1$; or 1, if $H(1) \neq 1$.
(Wrong Height)

BINARY-EMPTY-CHILD-W-HEIGHT. All of **(Empty Child)**, **(Wrong Root)**, **(Wrong Father)**, and **(Wrong Height)** are valid solutions.

Again, we remark that an isolated vertex u with $F(u) = L(u) = R(u) = u$ is *not* a solution of EMPTY-CHILD-W-HEIGHT or BINARY-EMPTY-CHILD-W-HEIGHT.

Due to the presence of heights, these problems are in PLS now:

Lemma 5.4. EMPTY-CHILD-W-HEIGHT is in PLS, and therefore, SOPL.

Proof. We reduce the EMPTY-CHILD-W-HEIGHT instance to the instance of SINK-OF-DAG where for each node $v \in V$, the successor of v is $L(v)$ and the potential of v is $H(v)$. Let v be a solution of SINK-OF-DAG, then one of the following cases happens:

- $v = 1$ and $L(v) = 1$. Then 1 is a **(Wrong Root)**.
- $L(v) \neq v$ and $L(L(v)) = L(v)$. This implies that either $F(L(v)) \neq v$ or $F(L(L(v))) \neq L(v)$, hence either v or $L(v)$ is a node with **(Empty Child)**.
- $L(v) \neq v$ and $H(L(v)) \leq H(v)$. If $F(L(v)) \neq v$ then v is a node with **(Empty Child)**. Otherwise, let $u := L(v)$, we have that $u \neq F(u) = v$ and that $H(u) \leq H(F(u))$, hence u has **(Wrong Height)**.

Therefore, EMPTY-CHILD-W-HEIGHT reduces to SINK-OF-DAG and is in PLS. Finally, since EMPTY-CHILD-W-HEIGHT is also in LOSSY \subseteq PPADS, it is in SOPL = PLS \cap PPADS [GHJ⁺24]. \square

In contrast, it is easy to show that EMPTY-CHILD (as well as LOSSY-CODE and BIJ-LOSSY-CODE) is not in PLS using Prover-Delayer games and resolution width lower bounds (see, e.g., [PT19, Proposition 3.4]).

Remark 5.5. Consider the following seemingly harder variant of EMPTY-CHILD-W-HEIGHT where the definition of **(Wrong Height)** is changed to

- s4'. a vertex $u \in V \setminus \{1\}$ such that $H(u) \leq H(F(u))$.
(Wrong Height')

Call this variant EMPTY-CHILD-W-HEIGHT'. We can define BINARY-EMPTY-CHILD-W-HEIGHT' likewise. Clearly, the proof of Lemma 5.4 also shows that EMPTY-CHILD-W-HEIGHT' is in PLS (hence SOPL). Since EMPTY-CHILD-W-HEIGHT is LOSSY \cap SOPL-complete (as we will show in Theorem 5.6), it follows that EMPTY-CHILD-W-HEIGHT' is equivalent to EMPTY-CHILD-W-HEIGHT.

The difference between **(Wrong Height)** and **(Wrong Height')** resembles the difference between SINK-OF-METERED-LINE and SINK-OF-POTENTIAL-LINE. Indeed, our proof below (that EMPTY-CHILD-W-HEIGHT' reduces to EMPTY-CHILD-W-HEIGHT) starts from the fact that SINK-OF-METERED-LINE is SOPL-complete, hence it makes black-box use of the reduction from SINK-OF-POTENTIAL-LINE to SINK-OF-METERED-LINE [FGMS20].¹¹

¹¹Although [FGMS20] only claims a reduction from END-OF-POTENTIAL-LINE to END-OF-METERED-LINE, the same reduction also proves that SINK-OF-POTENTIAL-LINE reduces to SINK-OF-METERED-LINE.

Next, we show that **EMPTY-CHILD-W-HEIGHT** is complete for the class $\text{LOSSY} \cap \text{SOPL}$. The proof is inspired by the recent intersection results in **TFNP**: $\text{CLS} = \text{PPAD} \cap \text{PLS}$ [FGHS23], $\text{EOPL} = \text{PPAD} \cap \text{PLS}$, and $\text{SOPL} = \text{PPADS} \cap \text{PLS}$ [GHJ⁺24]. Given a **PPAD** instance A and a **PLS** instance B , [FGHS23] showed how to “combine” A and B into a **CLS** instance C such that a solution of C implies either a solution of A or a solution of B . Our idea is similar here: Given a **LOSSY-CODE** instance (f, g) and a **SINK-OF-METERED-LINE** instance (S, P, V) , we show how to create an instance (F, L, R, H) of **EMPTY-CHILD-W-HEIGHT** such that a solution of (F, L, R, H) implies either a solution of (f, g) or a solution of (S, P, V) . Roughly speaking, we use (f, g) to implicitly define an exponentially large tree and use (S, P, V) to define the heights on the tree.

SINK-OF-METERED-LINE [HY20, FGMS20]. Given functions $S, P : [N] \rightarrow [N]$ and $V : [N] \rightarrow [N] \cup \{0\}$ there are four types of possible solutions:

- s1. 1, if $P(1) \neq 1$ or $S(1) = 1$ or $V(1) \neq 1$, (Bad Source)
- s2. A vertex $x \in [N]$ such that $P(S(x)) \neq x$, (Sink of Line)
- s3. A vertex $x \in [N] \setminus \{1\}$ such that $V(x) = 1$, (Bad Meter I)
- s4. A vertex $x \in [N]$ such that $(V(x) > 0 \text{ and } V(S(x)) - V(x) \neq 1)$ or $(V(x) > 1 \text{ and } V(x) - V(P(x)) \neq 1)$. (Bad Meter II)

Theorem 5.6. $\text{LOSSY-CODE} \cap \text{SINK-OF-METERED-LINE}$ reduces to **EMPTY-CHILD-W-HEIGHT** by a black-box reduction.

Proof. An input of $\text{LOSSY-CODE} \cap \text{SINK-OF-METERED-LINE}$ consists of an instance (f, g) of **LOSSY-CODE** and an instance (S, P, V) of **SINK-OF-METERED-LINE**; either a solution for (f, g) or a solution for (S, P, V) is a valid solution. Now we show how to reduce this instance (f, g, S, P, V) to an instance of **EMPTY-CHILD-W-HEIGHT**. Let N, M denote the dimensions of the inputs, where $f : [N] \rightarrow [2N]$, $g : [2N] \rightarrow [N]$, $S, P : [M] \rightarrow [M]$, and $V : [M] \rightarrow [M] \cup \{0\}$. Without loss of generality, we may assume that $N = 2^n$ is a power of 2 (this follows from the robustness of **LOSSY-CODE** as shown in Lemma 2.3).

We will create a “binary tree” with $n + M$ levels. Roughly speaking, the first n levels form a complete binary tree, and each of the last M levels contains N vertices. Let (i, j) denote the j -th vertex in the i -th level; we abuse notation and write (i, j) also as the *index* of the node (i, j) , hence

$$(i, j) = \begin{cases} j + 2^{i-1} - 1 & \text{if } i \leq n, \\ j + N \cdot (i - n) - 1 & \text{otherwise.} \end{cases}$$

We now specify the structure of the tree.

- The father of the root is $F(1, 1) := (1, 1)$.
- For every $2 \leq i \leq n + 1$ and $j \in [2^{i-1}]$, $F(i, j) := (i - 1, \lceil j/2 \rceil)$.
- For every $2 \leq i \leq M$ and $j \in [N]$ s.t. $V(i) \neq 0$, $F(i + n, j) := (P(i) + n, \lceil f(j)/2 \rceil)$.
- For every $2 \leq i \leq M$ and $j \in [N]$ s.t. $V(i) = 0$, $F(i + n, j) := (i + n, j)$.
- For every $i \in [n]$ and $j \in [2^{i-1}]$, the left and right children of (i, j) are $L(i, j) := (i + 1, 2j - 1)$ and $R(i, j) := (i + 1, 2j)$ respectively.
- For every $i \in [M]$ and $j \in [N]$ s.t. $V(i) \neq 0$, $L(i + n, j) := (S(i) + n, g(2j - 1))$ and $R(i + n, j) := (S(i) + n, g(2j))$.
- For every $i \in [M]$ and $j \in [N]$ s.t. $V(i) = 0$, $L(i + n, j) := R(i + n, j) := (i + n, j)$.

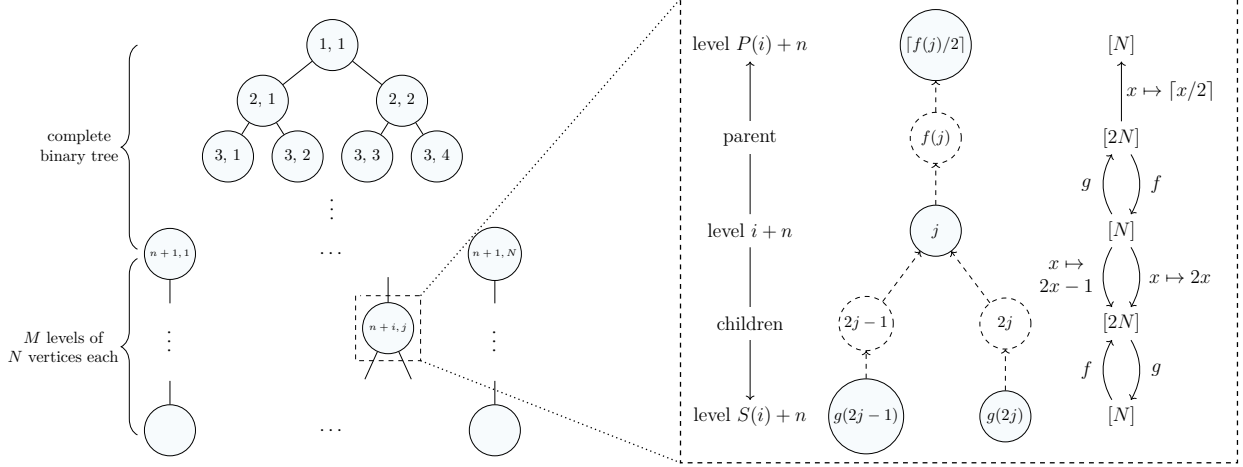


Figure 6: The “tree” constructed in [Theorem 5.6](#). **Left:** The first $n + 1$ levels form a complete binary tree and the last M levels are computed from the LOSSY-CODE instance (f, g) and the SINK-OF-METERED-LINE instance (S, P, V) . **Right:** For each node $(i + n, j)$ in the last M levels, the levels of its parent and children are computed from (S, P) , and the positions of these nodes within their levels are computed from (f, g) .

- For every $i \in [n]$, the height of every node in the i 'th level is i .
- For every $i \in [M]$, the height of every node in the $(i + n)$ 'th level is $V(i) + n$.

Now, given any solution (i', j) of EMPTY-CHILD-W-HEIGHT, we show how to obtain a solution of either (f, g) (for LOSSY-CODE) or (S, P, V) (for SINK-OF-METERED-LINE). In fact, from the above definitions of F, L, R one can see that it must be the case that $i' > n$; moreover, if (i', j) has [\(Wrong Height\)](#), then it must be the case that $i' > n + 1$ (unless $V(1) = 1$ and 1 is a [\(Bad Source\)](#) for SINK-OF-METERED-LINE). Now we define $i := i' - n$, hence $i \in [M]$. We can see that

$$\begin{aligned} F(L(i', j)) &= (P(S(i)) + n, \lceil f(g(2j-1))/2 \rceil); & (\text{if } S(i) \neq 1) \\ F(R(i', j)) &= (P(S(i)) + n, \lceil f(g(2j))/2 \rceil); & (\text{if } S(i) \neq 1) \\ H(i', j) - H(F(i', j)) &= V(i) - V(P(i)). & (\text{if } i > 1) \end{aligned}$$

Consider the following case analysis.

- Suppose that (i', j) has an [\(Empty Child\)](#). Clearly, $V(i) \neq 0$. If $F(L(i', j)) \neq (i', j)$, then either $S(i) = 1$, or $P(S(i)) \neq i$, or $\lceil f(g(2j-1))/2 \rceil \neq j$. If $F(R(i', j)) \neq (i', j)$, then either $S(i) = 1$, or $P(S(i)) \neq i$, or $\lceil f(g(2j))/2 \rceil \neq j$. If $L(i', j) = R(i', j) \neq (i', j)$, then $g(2j-1) = g(2j)$. Hence, one of the following holds:
 - $S(i) = 1$. If $P(1) \neq 1$ or $S(1) = 1$, then 1 is a [\(Bad Source\)](#) for SINK-OF-METERED-LINE; otherwise, since $P(S(i)) = 1 \neq i$, i is a [\(Sink of Line\)](#) for SINK-OF-METERED-LINE.
 - $P(S(i)) \neq i$. In this case, i is a [\(Sink of Line\)](#) for SINK-OF-METERED-LINE.
 - $\lceil f(g(2j-1))/2 \rceil \neq j$. In this case, $f(g(2j-1)) \neq 2j-1$, hence $2j-1$ is an answer for LOSSY-CODE.
 - $\lceil f(g(2j))/2 \rceil \neq j$. In this case, $f(g(2j)) \neq 2j$, hence $2j$ is an answer for LOSSY-CODE.
 - $g(2j-1) = g(2j)$. In this case, either $f(g(2j-1)) \neq 2j-1$ or $f(g(2j)) \neq 2j$, hence we can find an answer for LOSSY-CODE.
- Clearly $(1, 1)$ is not a [\(Wrong Root\)](#), so we do not need to care about this case.

- Suppose that (i', j) has **(Wrong Height)**. Then $V(i) - V(P(i)) \neq 1$ and $F(i', j) \neq (i', j)$, which implies that $V(i) \geq 1$. If $V(i) > 1$, then i is a **(Bad Meter II)** for SINK-OF-METERED-LINE; otherwise i is a **(Bad Meter I)** for SINK-OF-METERED-LINE.

It follows that given a solution for relaxed EMPTY-CHILD-W-HEIGHT, we could find a solution for either LOSSY-CODE or SINK-OF-METERED-LINE. \square

Since EMPTY-CHILD-W-HEIGHT is hard for LOSSY-CODE \cap SINK-OF-METERED-LINE ([Theorem 5.6](#)), is in SOPL ([Lemma 5.4](#)), and is in LOSSY ([Lemma 5.3](#)), we have:

Corollary 5.7. EMPTY-CHILD-W-HEIGHT is complete for SOPL \cap LOSSY.

Similarly, we can also reduce BIJ-LOSSY-CODE \cap END-OF-METERED-LINE to BINARY-EMPTY-CHILD-W-HEIGHT. Recall that END-OF-METERED-LINE is defined as follows:

END-OF-METERED-LINE. The input is functions $S, P : [N] \rightarrow [N]$ and $V : [N] \rightarrow [N] \cup \{0\}$ (i.e., same as SINK-OF-METERED-LINE). Besides the solutions of SINK-OF-METERED-LINE, we also accept the following solutions:

- s5.** A vertex $x \in [N] \setminus \{1\}$ such that $S(P(x)) \neq x$. **(End of Line)**
- s6.** A vertex $x \in [N]$ such that $(V(x) > 0 \text{ and } V(S(x)) - V(x) \neq 1)$. **(Bad Meter III)**

Corollary 5.8. BIJ-LOSSY-CODE \cap END-OF-METERED-LINE can be reduced to BINARY-EMPTY-CHILD-W-HEIGHT.

Proof Sketch. The reduction is exactly the same, except that we also need to handle the case that (i', j) has a **(Wrong Father)**. We can also see that $i' > n + 1$, hence $i = i' - n \geq 2$. We also have that $V(i) \neq 0$. Hence,

$$\begin{aligned} L(F(i', j)) &= (S(P(i)) + n, g(2\lceil f(j)/2 \rceil - 1)); & (\text{since } i > 1) \\ R(F(i', j)) &= (S(P(i)) + n, g(2\lceil f(j)/2 \rceil)). & (\text{since } i > 1) \end{aligned}$$

Hence, the case when (i', j) has a **(Wrong Father)** is argued as follows.

- Suppose that (i', j) has a **(Wrong Father)**. Letting $v := 2\lceil f(j)/2 \rceil$, then

$$(i', j) \notin \{(S(P(i)) + n, g(v - 1)), (S(P(i)) + n, g(v))\}.$$

Hence, one of the following holds:

- $i \neq S(P(i))$. In this case, i is an **(End of Line)** for END-OF-METERED-LINE.
- $j \notin \{g(v - 1), g(v)\}$. However $f(j) \in \{v - 1, v\}$, hence $g(f(j)) \neq j$ and j is an answer for BIJ-LOSSY-CODE. \square

5.3 LOSSY-Completeness of EMPTY-CHILD

Next, we adapt the above proof to show that LOSSY-CODE reduces to EMPTY-CHILD, establishing the equivalence between the two problems. Roughly speaking, our reduction makes use of two ideas:

1. LOSSY-CODE reduces to PPADS. In fact, a LOSSY-CODE instance $f : [N] \rightarrow [2N]$ and $g : [2N] \rightarrow [N]$ can be seen as a SINK-OF-LINE instance where g is the successor function, f is the predecessor function, and there are N distinguished sources numbered from $N + 1$ to $2N$.

2. One can reduce $\text{LOSSY-CODE} \cap \text{PPADS}$ to EMPTY-CHILD by slightly adapting the reduction in [Section 5.2](#), using the PPADS instance to handle the “heights”.

Our reduction starts by constructing a forest with $2N$ levels, where each level contains N vertices. The children and parents of each vertex can then be defined in the same way as those in the bottom M levels of the tree from [Theorem 5.6](#), except that a vertex at the i -th level has its father at the $f(i)$ -th level and children at the $g(i)$ -th level. This is already a “forest” where every vertex in levels $N + 1 \sim 2N$ are distinguished roots (hence there are N^2 distinguished roots). Finally, we create a complete binary tree of depth $\log(N^2)$ at the top to connect them.

Theorem 5.9. There is a reduction from LOSSY-CODE to EMPTY-CHILD .

Proof. Given $f : [N] \rightarrow [2N]$ and $g : [2N] \rightarrow [N]$ as the inputs to LOSSY-CODE , we construct an instance (F, L, R) of EMPTY-CHILD as follows: As in the proof of [Theorem 5.6](#), we may assume without loss of generality that $N = 2^n$ is a power of 2. The “binary tree” has $2(n + N)$ levels: the first $2n$ levels form a perfect binary tree, while each of the remaining $2N$ levels contains exactly N vertices. Following the notation in the proof of [Theorem 5.6](#), we write (i, j) for the j -th vertex in the i -th level. Slightly abusing notation, we also use (i, j) to denote the index of this vertex, namely

$$(i, j) = \begin{cases} j + 2^{i-1} - 1 & \text{if } i \leq 2n, \\ j + N \cdot (i - 2n - 1) + N^2 - 1 & \text{otherwise.} \end{cases}$$

The structure of the tree is described below:

- The father of the root is $F(1, 1) := (1, 1)$.
- For every $2 \leq i \leq 2n$ and $j \in [2^{i-1}]$, $F(i, j) := (i - 1, \lceil j/2 \rceil)$.
- For every $1 \leq i \leq N$ and $j \in [N]$, $F(i + 2n + N, j) := (2n, \lceil ((i - 1) \cdot N + j)/2 \rceil)$.
- For every $1 \leq i \leq N$ and $j \in [N]$, $F(i + 2n, j) := (f(i) + 2n, \lceil f(j)/2 \rceil)$.
- For every $i \in [2n - 1]$ and $j \in [2^{i-1}]$, the left and right children of (i, j) are $L(i, j) := (i + 1, 2j - 1)$ and $R(i, j) := (i + 1, 2j)$ respectively.
- For every $1 \leq i \leq N$ and $j \in [N/2]$, $L(2n, (i - 1)N/2 + j) := (i + 2n + N, 2j - 1)$ and $R(2n, (i - 1)N/2 + j) := (i + 2n + N, 2j)$.
- For every $i \in [2N]$ and $j \in [N]$, $L(i + 2n, j) := (g(i) + 2n, g(2j - 1))$ and $R(i + 2n, j) := (g(i) + 2n, g(2j))$.

Given any solution (i', j) of EMPTY-CHILD , we show how to obtain a solution of LOSSY-CODE . Note that our EMPTY-CHILD instance only has solutions with **(Empty Child)**: **(Wrong Father)** and **(Wrong Height)** are not allowed, and **(Wrong Root)** does not apply since our tree has a “correct” root. Hence, we have that $F(L(i', j)) \neq (i', j)$, or $F(R(i', j)) \neq (i', j)$, or $L(i', j) = R(i', j) \neq (i', j)$. It follows from the structure of our tree that $i' > 2n$; define $i := i' - 2n$. We have

$$\begin{aligned} F(L(i', j)) &= F(g(i) + 2n, g(2j - 1)) = (f(g(i)) + 2n, \lceil f(g(2j - 1))/2 \rceil); \\ F(R(i', j)) &= F(g(i) + 2n, g(2j)) = (f(g(i)) + 2n, \lceil f(g(2j))/2 \rceil). \end{aligned}$$

There are three cases:

- **Case I:** $F(L(i', j)) \neq (i', j)$. Then either $f(g(i)) \neq i$, or $\lceil f(g(2j - 1))/2 \rceil \neq j$. It follows that either i or $2j - 1$ is a valid solution for LOSSY-CODE .

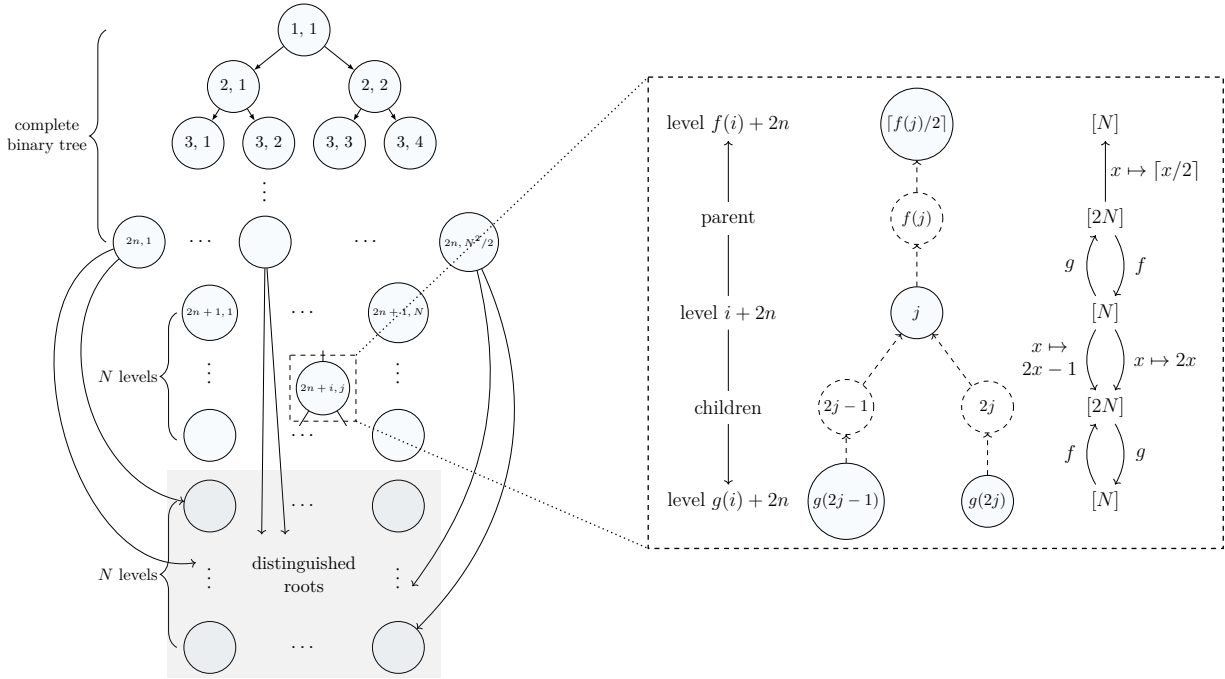


Figure 7: The “tree” constructed in [Theorem 5.9](#) where solid arrows represent tree edges. The first $2n$ levels along with the last N levels form the perfect binary tree. Every vertex in the last N levels are distinguished roots of the bottom forest, and hence are the leaves of the perfect binary tree. The last $2N$ levels form the bottom forest, whose structure is computed from the LOSSY-CODE instance (f, g) .

- **Case II:** $F(R(i', j)) \neq (i', j)$. Then either $f(g(i)) \neq i$, or $\lceil f(g(2j))/2 \rceil \neq j$. It follows that either i or $2j$ is a valid solution for LOSSY-CODE.
- **Case III:** $L(i', j) = R(i', j)$. Then $g(2j - 1) = g(2j)$, which means that either $2j$ or $2j - 1$ is a valid solution for LOSSY-CODE. \square

Similarly, BIJ-LOSSY-CODE reduces to BINARY-EMPTY-CHILD:

Corollary 5.10. There is a reduction from BIJ-LOSSY-CODE to BINARY-EMPTY-CHILD.

Proof Sketch. The reduction is exactly the same as in [Theorem 5.9](#), except that we also need to handle the case that (i', j) has a [\(Wrong Father\)](#). Clearly, $i = i' - 2n \in [N]$. Letting $v := 2\lceil f(j)/2 \rceil$, we have

$$\begin{aligned} L(F(i', j)) &= L(f(i) + 2n, \lceil f(j)/2 \rceil) = (g(f(i)) + 2n, g(v - 1)); \\ R(F(i', j)) &= R(f(i) + 2n, \lceil f(j)/2 \rceil) = (g(f(i)) + 2n, g(v)). \end{aligned}$$

Since $(i', j) \notin \{L(F(i', j)), R(F(i', j))\}$, there are two cases:

- **Case I:** $i' \neq g(f(i)) + 2n$. This implies that $g(f(i)) \neq i$, hence i is a valid solution for BIJ-LOSSY-CODE.
- **Case II:** $j \notin \{g(v - 1), g(v)\}$. However $f(j) \in \{v - 1, v\}$, hence $g(f(j)) \neq j$ and j is a valid solution for BIJ-LOSSY-CODE. \square

It is unclear whether BINARY-EMPTY-CHILD is equivalent to BIJ-LOSSY-CODE. We are unable to reduce the former to the latter, and the proof strategy of [Lemma 5.3](#) does not work here. We conjecture that (in the black-box setting) BINARY-EMPTY-CHILD is strictly harder than BIJ-LOSSY-CODE.

5.4 EMPTY-CHILD Reduces to NEPHEW

Recall once more the definition of NEPHEW:

NEPHEW. Given a set V of vertices and two functions $f : V \rightarrow V$ and $g : V \rightarrow V$. Think of $f(v)$ as the *father* of v and $g(v)$ as the *nephew* of v . A solution is one of the following.

- s1. $v \in V$ such that $f(f(g(v))) \neq f(v)$ (your nephew's grandparent is not your parent)
- s2. $v \in V$ such that $f(g(v)) = v$ (you are your nephew's parent)

NEPHEW is at least as powerful as EMPTY-CHILD. We show a reduction from an EMPTY-CHILD instance (V, F, L, R) to a NEPHEW instance (V', f, g) , where $V' = V \times \{0, 1\}$. If there is an EMPTY-CHILD solution at $v \in V$, then both of $(v, 0), (v, 1)$ will be self-loops for both f and g , giving a solution in the NEPHEW instance. Otherwise, we will construct a valid NEPHEW substructure. Special care will need to be taken in the case of isolated vertices in the EMPTY-CHILD instance, as well as for the distinguished root vertex $1 \in V$.

The procedure **Reduce-EC-to-Nephew** in [Algorithm 2](#) is the reduction. It takes in a vertex $(v, i) \in V'$ and returns a tuple containing $f(v, i)$ and $g(v, i)$. See [Figure 8](#) for an illustration of the NEPHEW instance returned by this procedure.

Before we prove its validity, we provide some intuition for the reduction. In the NEPHEW instance, we construct a structure where, for each vertex $v \in V$, the vertices corresponding to $F(v)$ are pointed to via the f function and the vertices corresponding to $L(v), R(v)$ are pointed to via the g function. In order to do this, we need to double the number of vertices: $(v, 0)$ will point via g to $R(v)$ and be pointed at via f by $L(v)$; and $(v, 1)$ will point via g to $L(v)$ and be pointed at via f by $R(v)$. Isolated vertices and the root vertex will be handled separately.

Algorithm 2 Procedure Reduce-EC-to-Nephew_{F,L,R}(v, i)

```
1: if  $F(L(v)) \neq v \vee F(R(v)) \neq v \vee L(v) = R(v) \neq v$  then
2:   return  $((v, i), (v, i))$   $\triangleright$  (Empty Child), so self-loop
3: else if  $v = 1$  and  $L(1) = 1 \vee R(1) = 1 \vee F(1) \neq 1$  then
4:   return  $((1, i), (1, i))$   $\triangleright$  (Wrong Root), so self-loop
5: else if  $F(v) = L(v) = R(v) = v$  then
6:   return  $((1, 0), (v, 1 - i))$   $\triangleright$  Isolated vertex, treat specially
7: else
8:   if  $v = 1$  then
9:      $f \leftarrow (1, 0)$ 
10:  else if  $v = R(F(v))$  then
11:     $f \leftarrow (F(v), 1)$ 
12:  else  $\triangleright v \neq 1$  and  $v$  is not a right child
13:     $f \leftarrow (F(v), 0)$ 
14:  if  $(v, i) = (1, 0)$  then
15:     $g \leftarrow (L(L(v)), 0)$ 
16:  else if  $(v, i) = (1, 1)$  then
17:     $g \leftarrow (1, 1)$ 
18:  else if  $i = 0$  then
19:     $g \leftarrow (R(v), 0)$ 
20:  else  $\triangleright v \neq 1$  and  $i = 1$ 
21:     $g \leftarrow (L(v), 0)$ 
22:  return  $(f, g)$ 
```

Theorem 5.11. $(f(v, i), g(v, i)) \leftarrow \text{Reduce-EC-to-Nephew}_{F,L,R}$ is a reduction from EMPTY-CHILD to NEPHEW.

Proof. Consider any $(v, i) \in V'$ such that v is a solution to EMPTY-CHILD. Then there is a self-loop on (v, i) with both f and g , meaning that $f(g(v, i)) = (v, i)$, and so (v, i) is a solution to the NEPHEW instance. Given such a NEPHEW solution, then, it is easy to find a solution to EMPTY-CHILD by checking v . Additionally, it is possible for there to be a NEPHEW solution if an EMPTY-CHILD solution exists at $L(v)$ or $R(v)$, or at $v = 1$ if a solution exists at $L(L(1))$. Again, it is easy to check if any of these is the case.

We will show that those scenarios are the only solutions to the NEPHEW instance. Consider any $(v, i) \in V'$ and assume there is no solution at v , $L(v)$, $R(v)$, or $L(L(1))$. Then we have the following cases:

- $F(v) = L(v) = R(v) = v$. Note then $v \neq 1$.
There is no solution of type 1:

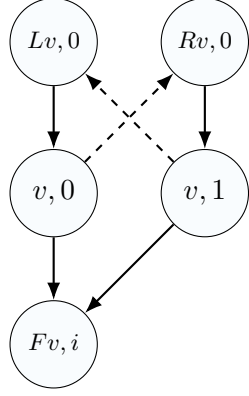
$$f(f(g(v, i))) = f(f(v, 1 - i)) = f(1, 0) = (1, 0) = f(v, i),$$

where the second-to-last equality comes from the fact that either 1 is not a solution, so $f(1, 0) = (1, 0)$, or 1 is a solution and $(1, 0)$ self-loops.

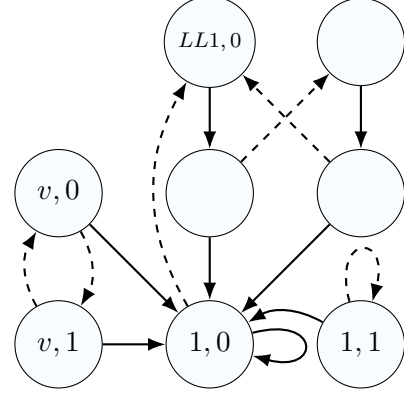
There is no solution of type 2:

$$f(g(v, i)) = f(v, 1 - i) = (1, 0) \neq (v, i).$$

For the remaining cases, assume $F(v) = L(v) = R(v) = v$ does not hold.



(a) Local Nephew structure for a typical node v . The value of i depends on whether or not v is a right child.



(b) Local Nephew structure around node 1, with a node v that is a self-loop in EMPTY-CHILD.

Figure 8: Illustration of $\text{Reduce-EC-to-Nephew}_{F,L,R}$. Solid arrows represent f and dashed arrows represent g . Parentheses are omitted in labels.

- $(v, i) = (1, 0)$.

There is no solution of type 1:

$$\begin{aligned} f(f(g(1, 0))) &= f(f(L(L(1)), 0)) = f(F(L(L(1))), 0) = f(L(1), 0) = (F(L(1)), 0) = (1, 0) \\ &= f(1, 0), \end{aligned}$$

where we used the fact that $L(1)$ and $L(L(1))$ are not solutions.

There is no solution of type 2:

$$f(g(1, 0)) = f(L(L(1)), 0) = (F(L(L(1))), 0) = (L(1), 0) \neq (1, 0).$$

- $(v, i) = (1, 1)$.

There is no solution of type 1:

$$f(f(g(1, 1))) = f(f(1, 1)) = f(1, 0) = (1, 0) = f(1, 1).$$

There is no solution of type 2:

$$f(g(1, 1)) = f(1, 1) = (1, 0) \neq (1, 1).$$

- $v \neq 1$ and $i = 0$. Observe that $R(F(R(v))) = R(v)$, as $F(R(v)) = v$.

There is no solution of type 1:

$$f(f(g(v, i))) = f(f(R(v), 0)) = f(F(R(v)), 1) = f(v, 1) = f(v, 0),$$

where the last inequality comes from the fact that $(v, 0)$ and $(v, 1)$ will always map to the same value under f .

There is no solution of type 2:

$$f(g(v)) = f(R(v), 0) = (F(R(v)), 1) = (v, 1) \neq (v, 0).$$

- $v \neq 1$ and $i = 1$.

There is no solution of type 1:

$$f(f(g(v, i))) = f(f(L(v), 0)) = f(F(L(v)), 0) = f(v, 0) = f(v, 1),$$

where the last inequality comes from the fact that $(v, 0)$ and $(v, 1)$ will always map to the same value under f .

There is no solution of type 2:

$$f(g(v, i)) = f(L(v), 0) = (F(L(v)), 0) = (v, 0) \neq (v, 1).$$

We have ruled out Nephew solutions other than those listed before the case analysis. Thus, every solution in the Nephew instance can be used to efficiently map back to an Empty-Child solution as required. \square

Combining [Theorem 5.11](#) with [Theorem 5.9](#), we obtain the following.

Corollary 5.12. There is a reduction from LOSSY-CODE to Nephew.

5.5 EMPTY-CHILD and Nephew with Inverse

It seems to be difficult to reduce Nephew to Empty-Child. It is possible that Nephew is strictly more powerful than Empty-Child. However, a proof of this seems elusive. To demonstrate this, we consider a natural modification to Nephew by adding an *inverse* to f . Interestingly, we show that this results in a problem that is *equivalent* to Empty-Child. If we believe that this modification is superficial, we could take this as evidence that Nephew and Empty-Child are in fact equivalent. Alternatively, it could indicate that any proof that shows that Nephew does not reduce to Empty-Child needs to argue how an inverse to f makes Nephew significantly easier.

We need to be a little bit careful in our definition of the inverse function. In the previous subsection we reduced Empty-Child to Nephew and created vertices that were not solutions yet were not pointed at via the f function: for an illustration see [Figure 8b](#). This situation only arises in the case where $f(f(v)) = f(v)$. Therefore we allow $f^{-1}(v) = \perp$ in instances where $f(f(v)) = f(v)$.

Nephew-W-Inverse. Given the same inputs as in Nephew and a function $f^{-1} : V \rightarrow V \cup \{\perp\}$, in addition to the solutions of Nephew, we accept the following solutions:

- s3.** $v \in V$ such that $f^{-1}(v) \neq \perp$ and $f(f^{-1}(v)) \neq v$. **(Wrong Inverse)**
- s4.** $v \in V$ such that $f^{-1}(v) = \perp$ and $f(f(v)) \neq f(v)$. **(Bad \perp)**

First we show that **Reduce-EC-to-Nephew** can be augmented to include the inverse function, and therefore Empty-Child reduces to Nephew-W-Inverse. This augmented procedure is shown in [Algorithm 3](#). It returns a tuple (f, f^{-1}, g) . New additions are in blue and underlined.

Theorem 5.13. Empty-Child reduces to Nephew-W-Inverse.

Proof. [Theorem 5.11](#) shows that **Reduce-EC-to-Nephew** is a reduction from Empty-Child to Nephew. The analysis in that proof still holds, as the f and g pointers returned are unmodified by the augmentation to **Reduce-EC-to-NwI**. Thus, all that is left to do is to prove that no additional solutions are introduced by the addition of the f^{-1} pointer.

As in the proof of [Theorem 5.11](#), consider $(v, i) \in V'$ and assume there is no solution at $v, L(v), R(v)$, or $L(L(1))$. We know already that there are no solutions of types 1 or 2. All that is left to prove is that no solutions of type 3 (**Wrong Inverse**) or type 4 (**Bad \perp**) exist.

- $F(v) = L(v) = R(v) = v$. Note then $v \neq 1$. $f^{-1}(v) = \perp$, so we need only to consider solutions of type 4. None exist:

$$f(f(v, i)) = f((1, 0)) = (1, 0) = f(v, i).$$

Algorithm 3 Procedure **Reduce-EC-to-NwI** _{F,L,R} (v, i)

```
1: if  $F(L(v)) \neq v \vee F(R(v)) \neq v \vee L(v) = R(v) \neq v$  then
2:   return  $((v, i), \underline{(v, i)}, (v, i))$   $\triangleright$  (Empty Child), so self-loop
3: else if  $v = 1$  and  $L(1) = 1 \vee R(1) = 1 \vee F(1) \neq 1$  then
4:   return  $((1, i), \underline{(1, i)}, (1, i))$   $\triangleright$  (Wrong Root), so self-loop
5: else if  $F(v) = L(v) = R(v) = v$  then
6:   return  $((1, 0), \underline{\perp}, (v, 1 - i))$   $\triangleright$  Isolated vertex, treat specially
7: else
8:   if  $v = 1$  then
9:      $f \leftarrow (1, 0)$ 
10:  else if  $v = R(F(v))$  then
11:     $f \leftarrow (F(v), 1)$ 
12:  else  $\triangleright v \neq 1$  and  $v$  is not a right child
13:     $f \leftarrow (F(v), 0)$ 
14:  if  $(v, i) = (1, 0)$  then
15:     $g \leftarrow (L(L(v)), 0)$ 
16:     $\underline{f^{-1}} \leftarrow \underline{(L(v), 0)}$ 
17:  else if  $(v, i) = (1, 1)$  then
18:     $g \leftarrow (1, 1)$ 
19:     $\underline{f^{-1}} \leftarrow \underline{\perp}$ 
20:  else if  $i = 0$  then
21:     $g \leftarrow (R(v), 0)$ 
22:     $\underline{f^{-1}} \leftarrow \underline{(L(v), 0)}$ 
23:  else  $\triangleright v \neq 1$  and  $i = 1$ 
24:     $g \leftarrow (L(v), 0)$ 
25:     $\underline{f^{-1}} \leftarrow \underline{(R(v), 0)}$ 
26:  return  $(f, \underline{f^{-1}}, g)$ 
```

- $(v, i) = (1, 0)$. $f^{-1}(v) \neq \perp$, so we need only to consider solutions of type 3. None exist:

$$f(f^{-1}(1, 0)) = f(L(1), 0) = (F(L(1)), 0) = (1, 0).$$

- $(v, i) = (1, 1)$. $f^{-1}(v) = \perp$, so we need only to consider solutions of type 4. None exist:

$$f(f(1, 1)) = f(1, 0) = (1, 0) = f(1, 1).$$

- $v \neq 1$ and $i = 0$. $f^{-1}(v) \neq \perp$, so we need only to consider solutions of type 3. None exist:

$$f(f^{-1}(v, 0)) = f(L(v), 0) = (F(L(v)), 0) = (v, 0).$$

- $v \neq 1$ and $i = 1$. $f^{-1}(v) \neq \perp$, so we need only to consider solutions of type 3. None exist:

$$f(f^{-1}(v, 1)) = f(R(v), 0) = (F(R(v)), 1) = (v, 1). \quad \square$$

We will now show that Nephew-w-Inverse reduces to Empty-Child. To do so, we will reduce it to Empty-Child'.

Theorem 5.14. Nephew-w-Inverse reduces to Empty-Child'.

Algorithm 4 Procedure Find-Children-and-Parent _{f, f^{-1}, g} (v)

```
1: if  $f^{-1}(v) = \perp$  then
2:   return ( $v, v; v$ )  $\triangleright$  Self-loop on vertices with no  $f$ -inverse.
3: else
4:    $v' \leftarrow f^{-1}(v)$   $\triangleright$  We will use the  $f$ -inverse of  $v$  as our starting point.
5:    $h(v') \leftarrow g(f(g(v')))$   $\triangleright$  Rename for notational brevity
6:   if CheckSol( $v$ )  $\vee$  CheckSol( $v'$ )  $\vee$  CheckSol( $f(g(v'))$ ) then
7:     return ( $\perp, \perp; f(v)$ )  $\triangleright$  We have found a solution, but always return  $f(v)$ .
8:   else
9:     return ( $f(g(v')), f(h(v')); f(v)$ )  $\triangleright$  The two children of  $v$  are  $f(g(v'))$  and  $f(h(v'))$ ; the parent of  $v$  is  $f(v)$ .
```

Proof. We use a variant of the Find-Children procedure named Find-Children-and-Parent that in addition also returns the parent vertex. Here, we define CheckSol(u) to be the procedure that returns True iff u is a solution to the NEPHEW-W-INVERSE instance.

Find-Children-and-Parent is given in Algorithm 4. See Figure 9 for an illustration. We show some useful properties of this procedure.

Claim 5.15. Let Find-Children-and-Parent _{f, f^{-1}, g} (v) = ($a, b; c$). Suppose $f^{-1}(v) \neq \perp$. If (a, b) \neq (\perp, \perp), then

- (i) $a \neq b$,
- (ii) $f(a) = f(b) = v$.

Proof of Claim 5.15.

- (i) As a is not a solution, we know that $b = f(g(a)) \neq a$.
- (ii) As v is not a solution, $f(v') = f(f^{-1}(v)) = v$. As v' is not a solution, $f(a) = f(f(g(v')))$ = $f(v') = v$. As a is not a solution, $f(b) = f(f(g(a))) = f(a) = v$. \diamond

Now we show the reduction. First, we need to choose a vertex from the NEPHEW-W-INVERSE instance to be our distinguished vertex 1 in EMPTY-CHILD'. We will select it such that $f(f(1)) \neq f(1)$.

Let v^* be the lexicographically first NEPHEW-W-INVERSE vertex. Before performing the rest of the reduction, check the vertices $\{v^*, f(v^*), g(f(v^*))\}$ to see if they are solutions. If any are, halt the reduction by returning some easily-falsified EMPTY-CHILD' instance (e.g. a single vertex that self-loops on F, L, R). Given such an EMPTY-CHILD' instance we can easily find a solution to the NEPHEW-W-INVERSE instance by ensuring that our reduction always checks these specified vertices.

Claim 5.16. If none of $\{v^*, f(v^*), g(f(v^*))\}$ are NEPHEW-W-INVERSE solutions, then either $f(f(v^*)) \neq f(v^*)$ or $f(f(g(f(v^*)))) \neq f(g(f(v^*)))$.

Proof. Assume otherwise. $f(v^*)$ is not a solution, so $f(f(g(f(v^*)))) = f(f(v^*))$ and $f(g(f(v^*))) \neq f(v^*)$. Then

$$f(f(v^*)) = f(f(g(f(v^*)))) = f(g(f(v^*))) \neq f(v^*),$$

a contradiction. \diamond

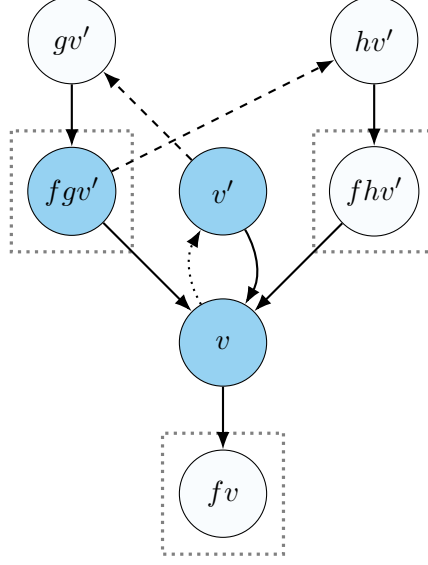


Figure 9: The procedure performed by $\text{Find-Children-and-Parent}_{f, f^{-1}, g}(v)$. Solid arrows represent f , the dotted arrow represents f^{-1} , and dashed arrows represent g . Parentheses are omitted in labels. The dotted boxes indicate the vertices that will be returned. The procedure will check if the shaded vertices are NEPHEW-W-INVERSE solutions, and in doing so will visit the unshaded vertices (but will not detect if these are solutions). Note that $f(h(v')) = v'$ is possible.

Set vertex 1 to be v^* if $f(f(v^*)) \neq f(v^*)$ and $g(f(v^*))$ otherwise. In the following, assume we have relabeled V such that the vertex chosen above is vertex 1.

For every vertex v , if $\text{Find-Children-and-Parent}_{f, f^{-1}, g}(v)$ returns (\perp, \perp, c) , let v^* be any vertex other than v , and assign $L(v) = v^*$, $R(v) = v^*$, and $F(v) = c$. This will be an EMPTY-CHILD' solution of type **(Empty Child)** and we can easily compute an NEPHEW-W-INVERSE solution if such an EMPTY-CHILD' solution is detected: check v , v' , and $f(g(v'))$ for solutions. Otherwise, assign

$$(L(v), R(v); F(v)) \leftarrow \text{Find-Children-and-Parent}_{f, f^{-1}, g}(v).$$

We will show that the only solutions to the EMPTY-CHILD' instance are those that are associated with (\perp, \perp, c) outputs as described above. Consider any v in the EMPTY-CHILD' instance and split into two cases: one where $F(v) = L(v) = R(v) = v$ and one where this does not hold.

In the first case:

- $F(L(v)) = F(v) = v$ and $F(R(v)) = F(v) = v$.
- $L(v) = R(v) = v$.
- It is not the case that $v = 1$. We must have $f^{-1}(v) = \perp$ but we used [Claim 5.16](#) to choose vertex 1 such that $f(f(1)) \neq f(1)$. Together, these would imply that 1 is a solution of type **(Bad \perp)**, but we also chose 1 to not be a solution.

In the second case, we may assume there is no NEPHEW-W-INVERSE solution at v , v' , and $f(g(v'))$. Then:

- $F(L(v)) = v$ and $F(R(v)) = v$ by [Claim 5.15 Item \(ii\)](#). Note that this is why we need to always return a value for F in **Find-Children-and-Parent**: even if $L(v)$ or $R(v)$ are solutions, they will still correctly point back to v .
- $L(v) \neq R(v)$ by [Claim 5.15 Item \(i\)](#).

- $L(1) \neq 1$ and $R(1) \neq 1$. We assigned vertex 1 such that $f(f(1)) \neq f(1)$, which implies $f(1) \neq 1$. Thus, $F(L(1)) = 1$ but $F(1) \neq 1$, implying $L(1) \neq 1$ and the same argument works for R . \square

6 The Strength of LOSSY-CODE

We start with the following candidate problem that, at the beginning of this research, was conjectured to be strictly harder than LOSSY-CODE.

c -AMGM-LC. Let $c > 1$ be a constant, $V := [2N]$ and $P := [c \cdot N^2]$.

The input is a coloring function $C : V \rightarrow \{0, 1\}$ and two mappings $F : P \rightarrow V \times V$, $G : V \times V \rightarrow P$. Let $H := C^{-1}(0) \times C^{-1}(1)$. The goal is to find solutions of either type:

- s1. a pigeon $x \in P$ such that $G(F(x)) \neq x$; (Wrong Encoding-Decoding)
- s2. a pigeon $x \in P$ such that $F(x) \notin H$; (Invalid Hole)

The main result in this section is that c -AMGM-LC is, in fact, reducible to LOSSY-CODE. Moreover, our techniques allow us to reduce problems similar to (and sometimes more complicated than) c -AMGM-LC to LOSSY-CODE in a *systematic* way; we provide two additional examples later (c -Dual-AMGM-LC in [Section 6.2.2](#) and a problem capturing the Inclusion-Exclusion principle in [Section 6.2.3](#)).

Some of the results in this section are already known in the world of bounded arithmetic: they follow from the machineries underlying Jeřábek’s theory of (additive) approximate counting $\text{APC}_1 := \text{PV}_1 + \text{dwPHP}(\text{PV})$ [[Jeř07a](#)]. For example, it is not hard to formalize the AM-GM inequality in APC_1 (more precisely, prove in APC_1 that c -AMGM-LC is total); Wilkie’s witnessing theorem ([[Tha02](#)], [[Jeř04](#), Proposition 1.14]) implies that every NP search problem provably total in APC_1 (including c -AMGM-LC) reduces to LOSSY-CODE.

One of the goals of this section is to introduce the ideas of APC_1 to audiences who are less familiar with APC_1 (or bounded arithmetic in general). In [Section 6.1](#), we introduce reconstructive pseudorandom generators with *feasible witnesses*, which is the central technique underlying [[Jeř07a](#)]. This technique allows us to put a wide range of problems similar to c -AMGM-LC inside LOSSY-CODE.

6.1 Basics of APC_1

This subsection presents some background of APC_1 [[Jeř07a](#)] that is needed in our reductions. In [[Jeř07a](#)], Jeřábek uses the *Nisan–Wigderson generator* [[NW94](#)] to approximate the size of feasible sets. Looking ahead, we will abstract the properties needed for the Nisan–Wigderson generator as *reconstructive PRGs* with “feasible witnesses” in some sense.

We first define the notion of *injection-surjection pairs*. Let A, B be two sets. We say that there is an *injection-surjection pair* certifying $|A| \leq |B|$ if there are polynomial-time computable functions $f : A \rightarrow B$ and $g : B \rightarrow A$ such that for every $x \in A$, $g(f(x)) = x$. This will be denoted by the following notation

$$A \overset{f}{\underset{g}{\rightleftharpoons}} B.$$

(Note that in the above notation, the injection-surjection pair is implicitly assuming that the left-hand side (A) is smaller than the right-hand side (B); hence $A \rightleftharpoons B$ and $B \rightleftharpoons A$ have very different meanings.)

Injection-surjection pairs are the basic primitive used in APC_1 to compare the sizes of two sets; roughly speaking, this is because the underlying principle is the *retraction (weak) pigeonhole principle* (over polynomial-time functions; i.e., the totality of LOSSY-CODE).

For any sets A, B and two functions $A \xrightleftharpoons[g]{f} B$, we say that an input $x \in A$ *witnesses* that $A \rightleftharpoons B$ is not an injection-surjection pair, if $g(f(x)) \neq x$. Otherwise (i.e., $g(f(x)) = x$), we say that x *maps to itself* via $A \rightleftharpoons B$. We also use $X \dot{\cup} Y$ to denote the *disjoint union* of two sets X, Y .

APC_1 -provably reconstructive pseudorandom generators. Let $N = 2^n, D = 2^d, M = 2^m$, a function $\text{PRG} : [N] \times [D] \rightarrow [M]$ is called a (k, ε) -reconstructive PRG if for every subset $S \subseteq [M]$, for all but at most $K := 2^k$ values of $f \in [N]$, we have

$$\left| \frac{|S \cap \text{PRG}_f|}{D} - \frac{|S|}{M} \right| \leq \varepsilon, \quad (3)$$

where we write $\text{PRG}_f = \{\text{PRG}(f, \text{seed}) : \text{seed} \in [D]\}$ for convenience. Furthermore, we want this generator to have “feasible witnesses” in the following sense:

Definition 6.1. Let $G_{<}^{(-)}, H_{<}^{(-)}, G_{>}^{(-)}, H_{>}^{(-)}$ be polynomial-time oracle algorithms. We say that $f \in [N]$ *provides an ε -(additive) approximation of $|S|$ feasibly*, if letting $\text{val} := |S \cap \text{PRG}_f| \cdot (M/D)$, then we have the following two injection-surjection pairs:

$$[\text{val}] \times [D] \xrightleftharpoons[G_{<}^S(f, -)]{H_{<}^S(f, -)} (S \dot{\cup} [\varepsilon M]) \times [D], \text{ and} \quad (4)$$

$$S \times [D] \xrightleftharpoons[G_{>}^S(f, -)]{H_{>}^S(f, -)} ([\text{val}] \dot{\cup} [\varepsilon M]) \times [D]. \quad (5)$$

Roughly speaking, these injection-surjection pairs certify (3): $(G_{<}, H_{<})$ certifies $\frac{|S \cap \text{PRG}_f|}{D} \leq \frac{|S|}{M} + \varepsilon$, while $(G_{>}, H_{>})$ certifies $\frac{|S|}{M} \leq \frac{|S \cap \text{PRG}_f|}{D} + \varepsilon$.

Fix $S \subseteq [M]$, we want that all but K values $f \in [N]$ provide an ε -additive approximation of $|S|$ feasibly. In fact, we can construct a pair of *reconstruction algorithm* **Comp** and **Decomp**, which are deterministic oracle algorithms satisfying the following. Given any witness w that (4) or (5) is *not* an injection-surjection pair, $\text{Comp}^S(f, w)$ compresses f into an element $\tilde{f} \in [K]$ (i.e., compresses f into k bits), and $\text{Decomp}^S(\tilde{f})$ decompresses \tilde{f} back to f .

The following theorem asserts that some explicit generator (in fact, the Nisan–Wigderson generator [NW94]) has “feasible witnesses” in the above sense. It is implicit in [Jeř07a]; for completeness, we provide a proof in [Appendix C](#).

Theorem 6.2. Let $n, m \in \mathbb{N}$, $\varepsilon > 0$, and $\rho > 1$ be parameters. Let $d := O\left(\frac{\log^2(nm/\varepsilon)}{\log \rho}\right)$ and $k := d + (\rho + 1)(m - 1) + O(\log(m/\varepsilon))$. Let $N := 2^n$, $M := 2^m$, $K := 2^k$, and $D := 2^d$.

Then there is a (k, ε) -reconstructive generator $\text{PRG} : [N] \times [D] \rightarrow [M]$ with $\text{poly}(\rho mn/\varepsilon)$ -time deterministic oracle algorithms $G_{<}, H_{<}, G_{>}, H_{>}, \text{Comp}, \text{Decomp}$ such that the following holds. For every set $S \subseteq [M]$, every $f \in [N]$, and every input w witnessing that either (4) or (5) is not an injection-surjection pair (where $\text{val} := |S \cap \text{PRG}_f| \cdot \frac{M}{D}$), we have $\text{Comp}^S(f, w) \in [K]$ and

$$\text{Decomp}^S(\text{Comp}^S(f, w)) = f.$$

In particular, the above theorem implies that if we take f to be “hard enough” (i.e., f is not in the range of Decomp^S) then f provides an ε -additive approximation of $|S|$ feasibly, analogous to the classical theorem that we can use a hard truth table to derandomize BPP. This is reminiscent of the “compress-or-random” technique in catalytic computing [Pyn24, CLMP25, KMPS25, AM25]: Any string f is either “random”, which means it can be used for derandomization, or “compressible” and admits a short description.

6.2 Reductions to Lossy-Code

6.2.1 AMGM-LC

We first show that c -AMGM-LC is in LOSSY-CODE.

Theorem 6.3. For every constant $c > 1$, there is a decision tree reduction of $\text{polylog}(N)$ query complexity from c -AMGM-LC to LOSSY-CODE.

Proof. Let $\varepsilon := (c-1)/3$, $\rho := \lceil \log N \rceil$, $d := O(\log \log N)$, $k := d + (\rho+1) \log |V| + O(\log(|V|/\varepsilon)) \leq O(\log^2 N)$, and $n' := 10 \lceil \log N \rceil^2$. Let $D := 2^d$, $K := 2^k$, and $N' := 2^{n'}$. Let $\text{PRG} : [N'] \times [D] \rightarrow V$ be defined in Theorem 6.2.

Let $f \in [N']$; for now, it would be convenient to assume that f successfully provides ε -approximations feasibly. Then we can estimate the size of $C^{-1}(0)$ as $v_0 := |C^{-1}(0) \cap \text{PRG}_f| \cdot (2N)/D$; the estimation of $|C^{-1}(1)|$ is thus $2N - v_0 = |C^{-1}(1) \cap \text{PRG}_f| \cdot (2N)/D$. Furthermore, we can obtain injection-surjection pairs (that depend on f)

$$C^{-1}(0) \times [D] \rightleftharpoons [v_0 + 2\varepsilon N] \times [D] \text{ and} \quad (6)$$

$$C^{-1}(1) \times [D] \rightleftharpoons [2N - v_0 + 2\varepsilon N] \times [D]. \quad (7)$$

Since

$$|C^{-1}(0)| \cdot |C^{-1}(1)| \leq (v_0 + \varepsilon \cdot 2N) \cdot (2N - v_0 + \varepsilon \cdot 2N) \leq (1 + 2\varepsilon)N^2,$$

we obtain an injection-surjection pair

$$C^{-1}(0) \times C^{-1}(1) \times [D]^2 \rightleftharpoons [(1 + 2\varepsilon)N^2] \times [D]^2. \quad (8)$$

Combining this with (F, G) , the purported injection-surjection pair from P to H , we obtain an injection-surjection pair

$$P \times [D]^2 \rightleftharpoons [(1 + 2\varepsilon)N^2] \times [D]^2. \quad (9)$$

Since $|P| = cN^2 > (1 + \Omega(1)) \cdot (1 + 2\varepsilon)N^2$, this would be a contradiction to the retraction weak pigeonhole principle. Hence, by solving LOSSY-CODE we can find a witness that (9) is not an injection-surjection pair, which is also an answer of the original c -AMGM-LC instance.

We have shown that given some f that successfully provides ε -approximations, we can reduce c -AMGM-LC to LOSSY-CODE. But what if f does not provide such approximations? In this case, Comp allows us to compress such f into a k -bit string. In particular:

- For any input (x, y) witnessing that (6) is not an injection-surjection pair, $\text{Comp}^{C^{-1}(0)}(f, x, y)$ returns some $\tilde{f} \in [K]$ such that $\text{Decomp}^{C^{-1}(0)}(\tilde{f}) = f$.
- For any input (x, y) witnessing that (7) is not an injection-surjection pair, $\text{Comp}^{C^{-1}(1)}(f, x, y)$ returns some $\tilde{f} \in [K]$ such that $\text{Decomp}^{C^{-1}(1)}(\tilde{f}) = f$.

Now we are ready to formally present our reduction from AMGM-LC to LOSSY-CODE. Let $X := [N'] \times P \times [D]^2$, $Y_1 := [N'] \times [(1 + 2\varepsilon)N^2] \times [D]^2$, $Y_2 := [2] \times [K] \times P \times [D]^2$, and $Y := Y_1 \cup Y_2$. Then

$$\begin{aligned} |Y| &= N'(1 + 2\varepsilon)N^2 \cdot D^2 + 2K \cdot cN^2 \cdot D^2 \\ &\leq N'(1 + 2.1\varepsilon)N^2 D^2 \\ &\leq (1 - \Omega(1))N' \cdot cN^2 \cdot D^2 = (1 - \Omega(1))|X|. \end{aligned}$$

We reduce the AMGM-LC instance to a LOSSY-CODE instance that consists of a pair of functions $F^* : X \rightarrow Y$ and $G^* : Y \rightarrow X$.

- The function $F^* : X \rightarrow Y$ takes as inputs $f \in [N']$ and $(p, u_1, u_2) \in P \times [D]^2$. It considers the injection defined in (9) that corresponds to f and computes its output (q, v_1, v_2) on input (p, u_1, u_2) . Then it checks whether f is “good”: if f is “good” then it outputs $(f, q, v_1, v_2) \in Y_1$, otherwise it outputs some value in Y_2 containing a compression of f .

More precisely, let $(a, b) := F(p)$, assume that $a \in C^{-1}(0)$ and $b \in C^{-1}(1)$. To evaluate (9) on input (p, u_1, u_2) , we need to evaluate (a, u_1) on (6) and (b, u_2) on (7). If (a, u_1) witnesses that (6) is not an injection-surjection pair, then f is not “good”, and f can be compressed into $\tilde{f} := \text{Comp}^{C^{-1}(0)}(f, a, u_1) \in [K]$. We return $(0, \tilde{f}, p, u_1, u_2) \in Y_2$ in this case. Similarly, if (b, u_2) witnesses (7) is not an injection-surjection pair, then we let $\tilde{f} := \text{Comp}^{C^{-1}(1)}(f, b, u_2) \in [K]$ and return $(1, \tilde{f}, p, u_1, u_2) \in Y_2$. If none of the above happens, then f is “good” and we return $(f, q, v_1, v_2) \in Y_1$.

- The function $G^* : Y \rightarrow X$ can be decomposed into functions $G_1^* : Y_1 \rightarrow X$ and $G_2^* : Y_2 \rightarrow X$. In either case, we are given a “compressed” form of some $(f, p, u_1, u_2) \in X$ and need to recover (f, p, u_1, u_2) .

For $G_1^* : Y_1 \rightarrow X$, the “compressed form” is $(f, q, v_1, v_2) \in Y_1$. This corresponds to the case that f is “good”. We consider the surjection defined in (9) corresponding to f and compute its output (p, u_1, u_2) given input (q, v_1, v_2) . Then we output $(f, p, u_1, u_2) \in X$.

For $G_2^* : Y_2 \rightarrow X$, the “compressed form” is $(b, \tilde{f}, p, u_1, u_2) \in Y_2$. This corresponds to the case that f is “not good”, and we can directly compute $f := \text{Decomp}^{C^{-1}(b)}(\tilde{f})$. Then we output $(f, p, u_1, u_2) \in X$.

To see the correctness of this reduction, consider any input (f, p, u_1, u_2) witnessing that $X \xrightleftharpoons[G^*]{F^*} Y$ is not an injection-surjection pair. There are two cases:

- Suppose that $F^*(f, p, u_1, u_2) = (f, q, v_1, v_2) \in Y_1$. Letting $(a, b) := H(p)$, this means that (a, u_1) maps to itself via (6), and (b, u_2) maps to itself via (7). In other words, (a, b, u_1, u_2) maps to itself via (8). Hence, it has to be the case that $P \xrightleftharpoons[G]{F} C^{-1}(0) \times C^{-1}(1)$ does not map p to itself, which means that p is a solution to AMGM-LC.
- The other case is that $F^*(f, p, u_1, u_2) = (b, \tilde{f}, p, u_1, u_2) \in Y_2$. We claim that this case would not have happened. Indeed, in this case, G_2^* always maps $(b, \tilde{f}, p, u_1, u_2)$ back to (f, p, u_1, u_2) .

Finally, it is easy to see that both F^* and G^* run in deterministic $\text{polylog}(N)$ time. \square

6.2.2 Dual-AMGM-LC

Now we define the *dual* of the problem c -AMGM-LC, which expresses the AM-GM inequality in a different way:

c -Dual-AMGM-LC. Let $V = [2N]$, $H = [c \cdot N^2]$. The input consists of a coloring function $C : V \rightarrow \{0, 1\}$ and two mappings $F : H \rightarrow V \times V$, $G : V \times V \rightarrow H$. Let

$$P := \{(u, v) : C(u) = C(v)\} = (C^{-1}(0) \times C^{-1}(0)) \cup (C^{-1}(1) \times C^{-1}(1)).$$

The goal is to find a pigeon $x \in P$ such that $F(G(x)) \neq x$.

Theorem 6.4. For every constant $0 < c < 2$, there is a decision tree reduction of $\text{polylog}(N)$ query complexity from c -Dual-AMGM-LC to LOSSY-CODE.

Proof. Let $\varepsilon := (2 - c)/32$ and $c' := c + 8\varepsilon(1 + \varepsilon)$, then $c' < 2$. Let $\rho := \lceil \log N \rceil$, $d := O(\log \log N)$, $n' := 10 \lceil \log N \rceil^2$, and $k := d + (\rho + 1)(\log |V| - 1) + O(\log(|V|/\varepsilon)) \leq 2 \lceil \log N \rceil^2$. Let $D := 2^d = \text{polylog}(N)$, $K := 2^k$, $N' := 2^{n'}$. Consider the generator $\text{PRG} : [N'] \times D \rightarrow V$ in [Theorem 6.2](#).

Let $f \in [N']$. Let $v_0 := |C^{-1}(0) \cap \text{PRG}_f| \cdot (2N/D)$ and $v_1 := 2N - v_0$ be the estimations of $|C^{-1}(0)|$ and $|C^{-1}(1)|$ provided by PRG_f , respectively. Then we obtain (purported) injection-surjection pairs

$$\begin{aligned} [v_0] \times [D] &\rightleftharpoons (C^{-1}(0) \dot{\cup} [2\varepsilon N]) \times [D] \text{ and} \\ [v_1] \times [D] &\rightleftharpoons (C^{-1}(1) \dot{\cup} [2\varepsilon N]) \times [D]. \end{aligned}$$

This implies an injection-surjection pair

$$\begin{aligned} [v_0^2 + v_1^2] \times [D]^2 &\xrightarrow[F_f]{F_f} \left((C^{-1}(0) \dot{\cup} [2\varepsilon N])^2 \dot{\cup} (C^{-1}(1) \dot{\cup} [2\varepsilon N])^2 \right) \times [D]^2 \\ &= \left(P \dot{\cup} (C^{-1}(0) \dot{\cup} C^{-1}(1)) \times [4\varepsilon N] \dot{\cup} [2(2\varepsilon N)^2] \right) \times [D]^2 \\ &= (P \dot{\cup} [8\varepsilon(1 + \varepsilon)N^2]) \times [D]^2. \end{aligned}$$

Composing this with our input $P \xrightarrow[G]{F} H$ and noting that $H = [cN^2]$ and $v_0^2 + v_1^2 \geq 2N^2$, we obtain

$$[2N^2] \times [D]^2 \xrightarrow[G_f \circ G]{F \circ F_f} [(c + 8\varepsilon(1 + \varepsilon))N^2] \times [D]^2 = [c'N^2] \times [D]^2.$$

(Note that we are abusing notation here in “ $G_f \circ G$ ” and “ $F \circ F_f$ ”; for example, we are extending the domain of G to $H \dot{\cup} [8\varepsilon(1 + \varepsilon)N^2]$ so that G is the identity map on $[8\varepsilon(1 + \varepsilon)N^2]$.)

Since $(c + 8\varepsilon(1 + \varepsilon)) < 2 - \Omega(1)$, $(F \circ F_f, G \circ G_f)$ is a valid LOSSY-CODE instance. Solving this LOSSY-CODE instance gives us a witness that $(F \circ F_f, G \circ G_f)$ is not an injection-surjection pair. This implies either a witness that (F, G) is not an injection-surjection pair (which is what we need), or a witness that (F_f, G_f) is not an injection-surjection pair (which will allow us to compress f).

Now we formally define the complete reduction from c -Dual-AMGM-LC to LOSSY-CODE. Let $X := [N'] \times [2N^2] \times [D]^2$, $Y_1 := [N'] \times [c'N^2] \times [D]^2$, $Y_2 := [K + O(1)] \times [2N^2] \times [D]^2$, and $Y := Y_1 \dot{\cup} Y_2$. Note that

$$|Y| = |Y_1| + |Y_2| \leq (c' + o(1))N'N^2D^2 \leq (2 - \Omega(1))N'N^2D^2 \leq (1 - \Omega(1))|X|.$$

We reduce the c -Dual-AMGM-LC instance (C, F, G) to the LOSSY-CODE instance $X \xrightleftharpoons[G^*]{F^*} Y$, defined as follows.

- The function $F^* : X \rightarrow Y$ takes as inputs $f \in [N']$ and $u \in [2N^2] \times [D]^2$. It tests if u is a witness that (F_f, G_f) is not an injection-surjection pair. If this is the case, then it uses **Comp** to compress f into $K + O(1)$ bits as \tilde{f} , and returns $(\tilde{f}, u) \in Y_2$. Otherwise it computes $v := F(F_f(u)) \in [c'N^2] \times [D]^2$ and returns $(f, v) \in Y_1$.

- The function $G^* : Y \rightarrow X$ takes either $(f, v) \in Y_1$ or $(\tilde{f}, u) \in Y_2$ as inputs. If the input is $(f, v) \in Y_1$ where $f \in [N']$ and $v \in [c'N^2] \times [D]^2$, then it computes $u := G(G_f(v)) \in [2N^2] \times [D]^2$ and returns $(f, u) \in X$. If the input is $(\tilde{f}, u) \in Y_2$ where $\tilde{f} \in [K + O(1)]$ and $u \in [2N^2] \times [D]^2$, then it decompresses $f \in [N']$ from \tilde{f} and returns $(f, u) \in X$.

Given any $(f, u) \in X$ witnessing that (F^*, G^*) is not an injection-surjection pair, we have that $F^*(f, u) \in Y_1$ and we can find a witness that (F, G) is not an injection-surjection pair in deterministic $\text{polylog}(N)$ time. Finally, F^*, G^* can be computed in deterministic $\text{polylog}(N)$ time as well. \square

6.2.3 The Inclusion-Exclusion Principle

Finally, as a proof-of-concept, we consider the following more complicated total search problem capturing the *inclusion-exclusion principle* and show that it is in LOSSY. Given that Jeřábek already proved this principle in APC₁ [Jeř07a, Proposition 2.19], it should come as no surprise that this problem is in LOSSY. Needless to say, our proof is just a translation of Jeřábek's proof into the language of black-box total search problems. We include this example as an additional demonstration of how to reduce more complicated problems to LOSSY-CODE via Theorem 6.2.

Consider the following inequality expressing the inclusion-exclusion principle: Let $S_1, S_2, \dots, S_\ell \subseteq [N]$ be sets, $a_i := |S_i|$, $a_{i,j} := |S_i \cap S_j|$, and

$$\tilde{a} := \sum_{i=1}^{\ell} a_i - \sum_{1 \leq i < j \leq \ell} a_{i,j}, \quad (10)$$

then $\left| \bigcup_{i \in [\ell]} S_i \right| \geq \tilde{a}$.

Now we formalize the above inequality as a TFZPP problem INCLUSION-EXCLUSION.

INCLUSION-EXCLUSION. The input consists of:

- Parameters $N, \ell \leq \text{polylog}(N)$, and $\varepsilon > 1/\text{polylog}(N)$.
- A table $T \subseteq [\ell] \times [N]$ where $T_{i,j} = 1$ if and only if $j \in S_i$.
- Numbers $0 \leq a_i \leq N$ for each $i \in [\ell]$, and $0 \leq a_{i,j} \leq N$ for each $1 \leq i < j \leq \ell$, which are purported estimates for $|S_i|$ and $|S_i \cap S_j|$ respectively. Let \tilde{a} be defined as in (10) and assume $\tilde{a} \geq \varepsilon N$.
- Injection-surjection pairs $f_i : [a_i] \rightarrow [N]$ and $g_i : [N] \rightarrow [a_i]$.
- Injection-surjection pairs $f_{i,j} : [a_{i,j}] \rightarrow [N]$ and $g_{i,j} : [N] \rightarrow [a_{i,j}]$.
- Finally, an injection-surjection pair $\tilde{f} : [\tilde{a} - \varepsilon N] \rightarrow [N]$ and $\tilde{g} : [N] \rightarrow [\tilde{a} - \varepsilon N]$.

The goal is to find any solution of the following types:

- s1. some $x \in [a_i]$ such that $T_{i,f_i(x)} = 0$ or $g_i(f_i(x)) \neq x$; (Violation of $|S_i| \geq a_i$)
- s2. some $x \in [N]$ such that $(T_{i,x} = T_{j,x} = 1)$ but $f_{i,j}(g_{i,j}(x)) \neq x$; or (Violation of $|S_i \cap S_j| \leq a_{i,j}$)
- s3. some $x \in [N]$ such that $\tilde{f}(\tilde{g}(x)) \neq x$, and there is some $j \in [\ell]$ such that $T_{j,x} = 1$. (Violation of $\left| \bigcup_{i \in [\ell]} S_i \right| \leq \tilde{a} - \varepsilon N$)

Theorem 6.5. INCLUSION-EXCLUSION reduces to LOSSY-CODE.

Proof. Let $\varepsilon' := \varepsilon/(10\ell^2)$, $\rho := \lceil \log N \rceil$, $d := O(\log \log N)$, $k := d + (\rho + 1) \log N + O(\log(N/\varepsilon)) \leq 3\lceil \log N \rceil^2$, and $n' := \log(100\ell^4\varepsilon^{-1}) + k \leq 4\lceil \log N \rceil^2$. Let $D := 2^d$, $K := 2^k$, and $N' := 2^{n'}$. Note

that $d' \geq O(\frac{\log^2(n' \cdot \log N / \varepsilon')}{\log \rho})$, hence it is valid to apply [Theorem 6.2](#) to obtain $\text{PRG} : [N'] \times [D] \rightarrow [N]$. Recall that we define $S_i := \{j \in [N] : T_{i,j} = 1\}$. We also define $S := \bigcup_{i \in [\ell]} S_i$.

Let $f \in [N']$. We use f to estimate each $|S_i|$, $|S_i \cap S_j|$, and $|S|$ within additive error $\varepsilon' \cdot N$. Let $v_i := |S_i \cap \text{PRG}_f| \cdot (N/D)$, $v_{i,j} := |S_i \cap S_j \cap \text{PRG}_f| \cdot (N/D)$, and $v := |S \cap \text{PRG}_f| \cdot (N/D)$, then we obtain (purported) injection-surjection pairs

$$S_i \times [D] \xleftrightarrow[G_i]{F_i} ([v_i] \dot{\cup} [\varepsilon' N]) \times [D], \quad (11)$$

$$[v_{i,j}] \times [D] \xleftrightarrow[F_{i,j}]{G_{i,j}} ((S_i \cap S_j) \dot{\cup} [\varepsilon' N]) \times [D], \quad (12)$$

$$[v] \times [D] \xleftrightarrow[F]{G} (S \dot{\cup} [\varepsilon' N]) \times [D]. \quad (13)$$

If $v_i + 2\varepsilon' N < a_i$, then we can compose (11) with the injection-surjection pair (f_i, g_i) to obtain the LOSSY-CODE instance

$$[a_i] \times [D] \xleftrightarrow[g_i \times [D]]{f_i \times [D]} S_i \times [D] \xleftrightarrow[G_i]{F_i} [v_i + \varepsilon' N] \times [D]. \quad (14)$$

Here, we use the notation $f_i \times [D]$ to denote the function mapping (x, y) to $(f_i(x), y)$, where $x \in [a_i]$ and $y \in [D]$; $g_i \times [D]$ is defined similarly. Note that since $\frac{(v_i + \varepsilon' N) \cdot D}{a_i \cdot D} \leq 1 - \frac{\varepsilon' N}{a_i} \leq \varepsilon'$, (14) is a valid LOSSY-CODE instance with good stretch; in particular, it reduces to standard LOSSY-CODE (of stretch $[2t] \Rightarrow [t]$) in decision tree depth $O(1/\varepsilon')$. Solving the instance (14) gives us an element $(x, y) \in [a_i] \times [D]$ such that either (1) x is a **(Violation of $|S_i| \geq a_i$)** or (2) $(f_i(x), y) \in S_i \times [D]$ witnesses that (11) is not a valid injection-surjection pair.

Similarly, if $v_{i,j} > a_{i,j} + 2\varepsilon' N$, then we compose (12) with the injection-surjection pair $(f_{i,j}, g_{i,j})$ to obtain the LOSSY-CODE instance

$$[v_{i,j}] \times [D] \xleftrightarrow[F_{i,j}]{G_{i,j}} ((S_i \cap S_j) \dot{\cup} [\varepsilon' N]) \times [D] \xleftrightarrow[f_{i,j} \times [D]]{g_{i,j} \times [D]} [a_{i,j} + \varepsilon' N] \times [D]. \quad (15)$$

Again, we abuse notation to extend $f_{i,j}$ and $g_{i,j}$ into functions $f_{i,j} : [a_{i,j}] \dot{\cup} [\varepsilon' N] \rightarrow [N] \dot{\cup} [\varepsilon' N]$ and $g_{i,j} : [N] \dot{\cup} [\varepsilon' N] \rightarrow [a_{i,j}] \dot{\cup} [\varepsilon' N]$ such that they are the identity function over $[\varepsilon' N]$. (15) is a valid LOSSY-CODE instance that reduces to the standard LOSSY-CODE in decision tree depth $O(1/\varepsilon')$. Solving the instance (15) gives us an element $(x, y) \in [v_{i,j}] \times [D]$ such that either (1) $x' := G_{i,j}(x)$ is a **(Violation of $|S_i \cap S_j| \leq a_{i,j}$)** or (2) (x, y) witnesses that (12) is not a valid injection-surjection pair.

Finally, if $v > \tilde{a} - \varepsilon N + 2\varepsilon' N$, then we compose (13) with the injection-surjection pair (\tilde{f}, \tilde{g}) to obtain the LOSSY-CODE instance

$$[v] \times [D] \xleftrightarrow[F]{G} (S \dot{\cup} [\varepsilon' N]) \times [D] \xleftrightarrow[\tilde{f} \times [D]]{\tilde{g} \times [D]} [\tilde{a} - \varepsilon N + \varepsilon' N] \times [D]. \quad (16)$$

Solving the instance (16) gives us an element $(x, y) \in [v] \times [D]$ such that either (1) $x' := G(x)$ is a **(Violation of $|\bigcup_{i \in [\ell]} S_i| \leq \tilde{a} - \varepsilon N$)** or (2) (x, y) witnesses that (13) is not a valid injection-surjection pair.

We now observe that one of the above three cases must happen. This is exactly due to the inclusion-exclusion principle itself: Let $\tilde{S}_i := S_i \cap \text{PRG}_f$, then

$$\left| \bigcup_{i \in [\ell]} \tilde{S}_i \right| \geq \sum_{i=1}^{\ell} |\tilde{S}_i| - \sum_{1 \leq i < j \leq \ell} |\tilde{S}_i \cap \tilde{S}_j|.$$

Recall that $v_i = |\tilde{S}_i|(N/D)$, $v_{i,j} = |\tilde{S}_i \cap \tilde{S}_j|(N/D)$, and $v = \left| \bigcup_{i \in [\ell]} \tilde{S}_i \right|(N/D)$, hence $v \geq \sum_{i=1}^{\ell} v_i - \sum_{1 \leq i < j \leq \ell} v_{i,j}$. If all three cases above do not happen, then

$$\begin{aligned}
\tilde{a} &\geq v + \varepsilon N - 2\varepsilon' N && (\because v > \tilde{a} - \varepsilon N + 2\varepsilon' N \text{ does not happen}) \\
&\geq \sum_{i=1}^{\ell} v_i - \sum_{1 \leq i < j \leq \ell} v_{i,j} + \varepsilon N - 2\varepsilon' N \\
&\geq \sum_{i=1}^{\ell} (a_i - 2\varepsilon' N) - \sum_{1 \leq i < j \leq \ell} v_{i,j} + \varepsilon N - 2\varepsilon' N && (\because v_i + 2\varepsilon' N < a_i \text{ does not happen}) \\
&\geq \sum_{i=1}^{\ell} (a_i - 2\varepsilon' N) - \sum_{1 \leq i < j \leq \ell} (a_{i,j} + 2\varepsilon' N) + \varepsilon N - 2\varepsilon' N && (\because v_{i,j} > a_{i,j} + 2\varepsilon' N \text{ does not happen}) \\
&\geq \sum_{i=1}^{\ell} a_i - \sum_{1 \leq i < j \leq \ell} a_{i,j} + (\varepsilon - 5\ell^2 \varepsilon') N > \tilde{a},
\end{aligned}$$

a contradiction.

Now we are ready to describe our reduction from INCLUSION-EXCLUSION to LOSSY-CODE. Let $X := [N'] \times [2N] \times [D]$ and Y be some set that we will define later, we will produce a LOSSY-CODE instance $X \xrightleftharpoons[G^*]{F^*} Y$. Given an element in X , we parse it as (f, x, y) where $f \in [N']$, $x \in [2N]$, and $y \in [D]$. To compute $F^*(f, x, y)$, we find out which of the above three cases happens for f .

- **Case I:** Suppose that there exists $i \in [\ell]$ such that $v_i + 2\varepsilon' N < a_i$. If $x \notin [a_i]$, then we let $x' := x - \varepsilon' N$ and $y' := y$; otherwise we feed (x, y) into

$$[a_i] \times [D] \xrightleftharpoons[g_i \times [D]]{f_i \times [D]} S_i \times [D] \xrightleftharpoons[G_i]{F_i} [v_i + \varepsilon' N] \times [D] \quad (14)$$

to obtain $(x', y') \in [v_i + \varepsilon' N] \times [D]$. We define $Y_0 := [N'] \times [2N - \varepsilon' N] \times [D]$ and map $F^*(f, x, y) = (f, x', y') \in Y_0$.

Note that the evaluation of (14) might be ill-behaved in the following cases. Suppose $x \in [a_i]$ and let $x' := f_i(x)$. If $x' \notin S_i$ or $g_i(x') \neq x$, then we know that x' is a **(Violation of $|S_i| \geq a_i$)** and we simply let $F^*(f, x, y) := \perp$. Otherwise, if (x', y) witnesses that (F_i, G_i) is not an injection-surjection pair, then S_i is a distinguisher for PRG_f . Let $Y_1 := [K] \times [\ell] \times [2N] \times [D]$, we define $F^*(f, x, y) := (\text{Comp}^{S_i}(f, x', y), i, x, y) \in Y_1$. If both cases above do not happen, we define $F^*(f, x, y) := (f, x', y') \in Y_0$ as usual.

- **Case II:** Suppose that there exists $1 \leq i < j \leq \ell$ such that $v_{i,j} > a_{i,j} + 2\varepsilon' N$. If $x \notin [v_{i,j}]$, then we let $x' := x - \varepsilon' N$ and $y' := y$; otherwise we feed (x, y) into

$$[v_{i,j}] \times [D] \xrightleftharpoons[F_{i,j}]{G_{i,j}} ((S_i \cap S_j) \cup [\varepsilon' N]) \xrightleftharpoons[f_{i,j} \times [D]]{g_{i,j} \times [D]} [a_{i,j} + \varepsilon' N] \times [D] \quad (15)$$

to obtain $(x', y') \in [a_{i,j} + \varepsilon' N] \times [D]$. We let $F^*(f, x, y) := (f, x', y') \in Y_0$.

Similarly, the evaluation of (15) might be ill-behaved in the following cases. Suppose $x \in [v_{i,j}]$. If (x, y) witnesses that $(G_{i,j}, F_{i,j})$ is not a valid injection-surjection pair, then $S_i \cap S_j$ is a distinguisher for PRG_f ; in this case, letting $Y_2 := [K] \times [\ell]^2 \times [2n] \times [D]$, we map $F^*(f, x, y) := (\text{Comp}^{S_i \cap S_j}(f, x, y), i, j, x, y) \in Y_2$. Otherwise, let $(x', y') := G_{i,j}(x, y)$, if $x' \in S_i \cap S_j$ but x' witnesses that $(g_{i,j}, f_{i,j})$ is not a valid injection-surjection pair, then x' is a **(Violation of $|S_i \cap S_j| \leq a_{i,j}$)** and we simply let $F^*(f, x, y) := \perp$. If both cases above do not happen, we define $F^*(f, x, y) = (f, x', y') \in Y_0$ as usual.

- **Case III:** Suppose that $v > \tilde{a} - \varepsilon N + 2\varepsilon' N$. If $x \notin [v]$, then we let $x' := x - \varepsilon' n$ and $y' := y$; otherwise we feed (x, y) into

$$[v] \times [D] \xrightarrow[F]{G} (S \dot{\cup} [\varepsilon' N]) \times [D] \xrightleftharpoons[\tilde{f} \times [D]]{\tilde{g} \times [D]} [\tilde{a} - \varepsilon N + \varepsilon' N] \times [D] \quad (16)$$

to obtain $(x', y') \in [\tilde{a} - \varepsilon N + \varepsilon' N] \times [D]$. We let $F^*(f, x, y) = (f, x', y') \in Y_0$.

Similarly, the evaluation of (16) on (x, y) might be ill-behaved in the following cases. Suppose $x \in [v]$. If (x, y) witnesses that (G, F) is not a valid injection-surjection pair, then S is a valid distinguisher for PRG_f ; in this case, letting $Y_3 := [K] \times [2N] \times [D]$, we map $F^*(f, x, y) := (\text{Comp}^S(f, x, y), x, y) \in Y_3$. Otherwise, let $(x', y') := G(x, y)$, if $x' \in S$ but x' witnesses that (\tilde{g}, \tilde{f}) is not a valid injection-surjection pair, then x' is a **(Violation of $|\bigcup_{i \in [q]} S_i| \leq \tilde{a} - \varepsilon N$)** and we let $F^*(f, x, y) := \perp$. If both cases above do not happen, we define $F^*(f, x, y) := (f, x', y') \in Y_0$ as usual.

Let Y be the disjoint union $Y := Y_0 \dot{\cup} Y_1 \dot{\cup} Y_2 \dot{\cup} Y_3 \dot{\cup} \{\perp\}$, then

$$\frac{|Y|}{|X|} \leq \frac{N'(2N - \varepsilon' N)D + K \cdot 2\ell^2 \cdot 2N \cdot D + 1}{N' \cdot 2N \cdot D} \leq 1 - \varepsilon'/2 + (K \cdot 2\ell^2 + 1)/N' \leq 1 - \varepsilon'/4.$$

We have described a function $F^* : X \rightarrow Y$ and it remains to describe $G^* : Y \rightarrow X$. The LOSSY-CODE instance (F^*, G^*) reduces to a standard LOSSY-CODE instance (of stretch $[2t] \Rightarrow [t]$) in decision tree depth $O(\log(1/\varepsilon')) \leq \text{polylog}(N)$. Given an input in Y :

- If the input is $(f, x', y') \in Y_0$, then one of the above three cases happen for f ; depending on this, we map (x', y') back to (x, y) via the appropriate surjection. For example, in case I: If $x' \notin [v_i + \varepsilon' n]$ then we let $x := x' + \varepsilon' n$ and $y' := y$; otherwise we let $(x'', y) := G_i(x', y')$ and $x := g_i(x')$.
- If the input is $(\hat{f}, i, x, y) \in Y_1$, then we recover $f := \text{Decomp}^{S_i}(\hat{f})$ and return (f, x, y) .
- If the input is $(\hat{f}, i, j, x, y) \in Y_2$, then we recover $f := \text{Decomp}^{S_i \cap S_j}(\hat{f})$ and return (f, x, y) .
- If the input is $(\hat{f}, x, y) \in Y_3$, then we recover $f := \text{Decomp}^S(\hat{f})$ and return (f, x, y) .
- We do not care about the value of $G^*(\perp)$; we map it to an arbitrary value.

Finally, we need to argue that this is a correct reduction. Let $(f, x, y) \in X$. If $F^*(f, x, y) \in Y_0$, then the injection-surjection pairs ((14), (15), (16)) are well-behaved, hence $G^*(F^*(f, x, y)) = (f, x, y)$; if $F^*(f, x, y) \in Y_1 \cup Y_2 \cup Y_3$, then the compression $(\text{Comp}^O(f, x, y))$ for the suitable oracle O works correctly, hence $G^*(F^*(f, x, y)) = (f, x, y)$ as well. Therefore, if (f, x, y) is a solution of the LOSSY-CODE instance, then $F^*(f, x, y) = \perp$. As discussed above, in this case, we will find a solution of INCLUSION-EXCLUSION. \square

6.2.4 Conclusion: LOSSY-Completeness

Finally, it is easy to see that the problems we considered (AMGM-LC, Dual-AMGM-LC, and INCLUSION-EXCLUSION) are at least as hard as LOSSY-CODE, hence they are LOSSY-complete.

Theorem 6.6. The following problems are LOSSY-complete:

- c -AMGM-LC for every constant $c > 1$;
- c -dual-AMGM-LC for every $0 < c < 2$;
- INCLUSION-EXCLUSION.

7 Dense Linear Ordering

Finally, we show that the dense version of the Linear Ordering Principle reduces to LOSSY-CODE. The Linear Ordering Principle was recently studied by Korten and Pitassi [KP24], where they showed that AVOID reduces to Linear Ordering. In the TFNP world, we show the opposite reduction and prove that Dense Linear Ordering reduces to Lossy Code. This problem has been studied in proof complexity [Rii01, AD08, Gry19, CdRN⁺23] as well.

DENSE-LINEAR-ORDERING. The input consists of the descriptions of a linear ordering \prec over N elements and a *median function* $\text{med} : [N] \times [N] \rightarrow [N]$. Without loss of generality, we may assume that for $x \neq y \in [N]$, exactly one of $(x \prec y)$ and $(y \prec x)$ is true, and that $\text{med}(x, y) = \text{med}(y, x)$. (That is, \prec is represented by a string of $\binom{N}{2}$ bits and med is represented by a list of $\binom{n}{2}$ elements in $[N]$.) A solution is one of the following.

- s1. $x, y, z \in [N]$ such that $x \prec y$, $y \prec z$, and $z \prec x$; or (Transitivity Violation)
- s2. $x, y \in [N]$ such that $x \prec y$, but neither $x \prec \text{med}(x, y)$ nor $\text{med}(x, y) \prec y$. (Invalid Median)

Theorem 7.1. DENSE-LINEAR-ORDERING reduces to LOSSY-CODE.

Proof. Let (\prec, med) be an input of DENSE-LINEAR-ORDERING over a universe \mathcal{U} of size N .

Fix some arbitrary l_0, r_0 in advance such that $l_0 \prec r_0$, and let $\ell := 4 \log N$. Consider the following LOSSY-CODE instance (f, g) between \mathcal{U} and $\{\text{L}, \text{R}\}^{\leq \ell}$.

Algorithm 5 $f : \mathcal{U} \rightarrow \{\text{L}, \text{R}\}^{\leq \ell}$

```

1: function  $f(m)$ 
2:    $\sigma \leftarrow$  the empty string
3:   for  $i \leftarrow 1$  to  $\ell$  do
4:      $m_{i-1} \leftarrow \text{med}(l_{i-1}, r_{i-1})$ 
5:     if  $m_{i-1} = m$  then
6:       return  $\sigma$ 
7:     else if  $m \prec m_{i-1}$  then
8:        $(l_i, r_i) \leftarrow (l_{i-1}, m_{i-1})$ ,  $\sigma \leftarrow \sigma \circ \text{L}$ 
9:     else  $\triangleright m_{i-1} \prec m$ 
10:       $(l_i, r_i) \leftarrow (m_{i-1}, r_{i-1})$ ,  $\sigma \leftarrow \sigma \circ \text{R}$ 
11:   return  $\sigma \triangleright$  regardless of whether we have
      arrived at  $m$ 

```

Algorithm 6 $g : \{\text{L}, \text{R}\}^{\leq \ell} \rightarrow \mathcal{U}$

```

1: function  $g(\sigma)$ 
2:   for  $i \leftarrow 1$  to  $|\sigma|$  do
3:      $m_{i-1} \leftarrow \text{med}(l_{i-1}, r_{i-1})$ 
4:     if  $\sigma_i = \text{L}$  then
5:        $(l_i, r_i) \leftarrow (l_{i-1}, m_{i-1})$ 
6:     else  $\triangleright \sigma_i = \text{R}$ 
7:        $(l_i, r_i) \leftarrow (m_{i-1}, r_{i-1})$ 
8:   return  $\text{med}(l_{|\sigma|}, r_{|\sigma|})$ 

```

In the following discussion, we assume that throughout the executions of f and g , we always have $l_i \prec r_i$ for each valid i . This is because otherwise we can find an (Invalid Median): Let i be the smallest index such that $l_i \not\prec r_i$, then $i \geq 1$ and $l_{i-1} \prec r_{i-1}$. Either we have $(l_i, r_i) = (l_{i-1}, m_{i-1})$, or we have $(l_i, r_i) = (m_{i-1}, r_{i-1})$. In both cases, (l_{i-1}, r_{i-1}) has an (Invalid Median).

Consider the following scenario. Suppose that $g(\sigma) = v$, but during the execution of $g(\sigma)$, we encountered some interval (l_i, r_i) such that either $(l_i \not\prec v)$ or $(v \not\prec r_i)$. We argue that in this scenario, it is easy to find a solution of (\prec, med) . Suppose $l_i \not\prec v$; the case that $v \not\prec r_i$ can be handled symmetrically.

- If $l_{|\sigma|} \not\prec v$ then, $(l_{|\sigma|}, r_{|\sigma|})$ has an (Invalid Median).

- Otherwise there is an index $j \in [i, |\sigma|)$ such that $l_j \not\prec v$ but $l_{j+1} \prec v$. It follows that $l_j \neq l_{j+1}$, which implies that $l_{j+1} = \text{med}(l_j, r_j)$.
 - If $l_j \not\prec l_{j+1}$ then we have an **(Invalid Median)** (l_j, r_j) .
 - Otherwise, since $l_{j+1} \prec v$ and $l_j \prec l_{j+1}$, we have $v \neq l_j$. Since $l_j \not\prec v$, we have $v \prec l_j$. Now we have a **(Transitivity Violation)** (l_j, l_{j+1}, v) where $v \prec l_j$, $l_j \prec l_{j+1}$, and $l_{j+1} \prec v$.

Now we prove the correctness of our reduction. That is, given any string $\sigma \in \{\text{L}, \text{R}\}^{\leq \ell}$ such that $f(g(\sigma)) \neq \sigma$, we can find a solution for DENSE-LINEAR-ORDERING. Let $v := g(\sigma)$ and $\sigma' := f(v)$, then $\sigma' \neq \sigma$. Let i be the smallest index such that $\sigma'_i \neq \sigma_i$. In particular, if σ' is a prefix of σ then we define $i := |\sigma'| + 1$. (Note that it cannot be the case that σ is a prefix of σ' , as otherwise the execution of $f(v)$ would have reached the element v in the $|\sigma|$ -th step and returned σ instead of σ' .) It follows that the first $i - 1$ rounds of $f(v)$ and $g(\sigma)$ produce the same intervals $\{(l_j, r_j)\}_{0 \leq j < i}$. Assume that $\sigma_i = \text{L}$; the case that $\sigma_i = \text{R}$ is symmetric. Let (l_i^g, r_i^g) denote the interval (l_i, r_i) in the execution of $g(\sigma)$, then $l_i^g = l_{i-1}$ and $r_i^g = m_{i-1} = \text{med}(l_{i-1}, r_{i-1})$.

- If $i = |\sigma'| + 1$, then $m_{i-1} = v = g(\sigma)$.
- Otherwise, $\sigma'_i = \text{R}$ and hence $m_{i-1} \prec v = g(\sigma)$.

In either case, we have found some (l_i^g, r_i^g) during the execution of $g(\sigma)$ such that $v \not\prec r_i^g$. By the previous discussion, we can find a solution of (\prec, med) . \square

8 Open Problems

We end with some future directions. The main problem left open by this work is to exhibit a natural problem in TFZPP^{dt} which is not reducible to LOSSY-CODE ; we conjecture that NEPHEW is such a problem. Some additional open questions are the following:

- Find a TFZPP upper bound for $\text{BERTRAND-CHEBYSHEV}$, i.e., a natural problem in TFZPP to which $\text{BERTRAND-CHEBYSHEV}$ reduces. The best upper bound we are aware of is only $\text{LOSSY}^{\text{FACTORING}}$ [PWW88, Kor22], which is in $\text{LOSSY}^{\text{PPA}}$ and $\text{LOSSY}^{\text{PWPP}}$ under the generalized Riemann Hypothesis [Jeř16]. Unfortunately, none of these upper bound classes are in TFZPP .
- Find a TFZPP upper bound for the following problem, which we call $\text{RAZBOROV-SMOLENSKY}$ [Raz87, Smo87]: The input is an $\text{AC}^0[2]$ circuit C of depth d and size at most $2^{n^{1/10d}}$ that attempts to compute MAJ (the Majority function), and the goal is to output an instance $x \in \{0, 1\}^n$ such that $C(x) \neq \text{MAJ}(x)$. Since MAJ is average-case hard against such circuits, this problem sits in TFZPP . This problem is trivially solvable in deterministic quasi-polynomial time (note that the naïve algorithm runs in $2^{O(n)}$ time while the input size is $2^{n^{\Omega(1)}}$), hence we are interested in the regime where only polynomial-time reductions are allowed. We are not aware of any syntactic subclass of TFZPP that contains this problem.
- Is $\text{BIJ-LOSSY-CODE} \in \text{PPAD}$? Observe that BIJ-LOSSY-CODE is equivalent to END-OF-LINE with half of the vertices designated as distinguished sources. The simple argument showing $\text{LOSSY-CODE} \in \text{PPADS}$, that is, output the solution returned by the PPAD solver on the original graph, does not work here, since the PPAD solver may return one of the distinguished sources. Recently, Goldberg and Hollender [GH21] gave an involved proof that END-OF-LINE with k distinguished sources belongs to PPAD when $k = \text{polylog}(N)$, where N denotes the size of the graph. However, their argument does not extend to the case that $k = \text{poly}(N)$ since the PPAD instance they constructed has size $N^{\Omega(k)}$.

- Can BINARY-EMPTY-CHILD be separated from BIJ-LOSSY-CODE?
- What is the relationship between DENSE-LINEAR-ORDERING and BIJ-LOSSY-CODE?

Acknowledgments

We thank Robert Robere, Yuhao Li, and Ben Davis for extensive discussions about the Nephew problem and TFZPP. As well, we thank the reviewers for suggestions which improved the presentation of this paper. Noah Fleming was supported by an NSERC Discovery grant and the Swedish Research Council under grant number 2025-06762. Stefan Grosser was supported by the NSERC CGS D fellowship. Siddhartha Jain was supported by Scott Aaronson’s Berkeley CIQC grant and an Amazon AI Fellowship. Jiawei Li was supported by Scott Aaronson’s Open Philanthropy grant. Morgan Shirley was supported by an NSERC grant and by Knut and Alice Wallenberg grant KAW 2023.0116. Weiqiang Yuan was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00026.

References

- [AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *J. Comput. Syst. Sci.*, 74(3):323–334, 2008. doi:10.1016/J.JCSS.2007.06.025.
- [AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. doi:10.4007/annals.2004.160.781.
- [AM25] Aryan Agarwala and Ian Mertz. Bipartite matching is in catalytic logspace. In *FOCS*, 2025. To appear. arXiv:2504.09991.
- [AT14] Albert Atserias and Neil Thapen. The ordering principle in a fragment of approximate counting. *ACM Trans. Comput. Log.*, 15(4):29:1–29:11, 2014. doi:10.1145/2629555.
- [BCE⁺98] Paul Beame, Stephen A. Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. *J. Comput. Syst. Sci.*, 57(1):3–19, 1998. doi:10.1006/JCSS.1998.1575.
- [BCK⁺14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *STOC*, pages 857–866. ACM, 2014. doi:10.1145/2591796.2591874.
- [BFI23] Sam Buss, Noah Fleming, and Russell Impagliazzo. TFNP characterizations of proof systems and monotone circuits. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPICs*, pages 30:1–30:40. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ITCS.2023.30.
- [BG01] Maria Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Comput. Complex.*, 10(4):261–276, 2001. doi:10.1007/S000370100000.
- [BGG01] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer Science & Business Media, 2001.

- [BGSD25] Huck Bennett, Surendra Ghentiyala, and Noah Stephens-Davidowitz. The more the merrier! On total coding and lattice problems and the complexity of finding multi-collisions. In *16th Innovations in Theoretical Computer Science Conference*, volume 325 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 14, 22. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2025. doi:[10.4230/lipics.itcs.2025.14](https://doi.org/10.4230/lipics.itcs.2025.14).
- [BHP01] R. C. Baker, G. Harman, and J. Pintz. The difference between consecutive primes. II. *Proc. London Math. Soc. (3)*, 83(3):532–562, 2001. doi:[10.1112/plms/83.3.532](https://doi.org/10.1112/plms/83.3.532).
- [BJ12] Samuel R Buss and Alan S Johnson. Propositional proofs and reductions between np search problems. *Annals of Pure and Applied Logic*, 163(9):1163–1182, 2012.
- [BKT14] Samuel R. Buss, Leszek Aleksander Kołodziejczyk, and Neil Thapen. Fragments of approximate counting. *J. Symb. Log.*, 79(2):496–525, 2014. doi:[10.1017/JSL.2013.37](https://doi.org/10.1017/JSL.2013.37).
- [BO06] Joshua Buresh-Oppenheim. On the TFNP complexity of factoring. *Unpublished*, 2006.
- [Bus97] Samuel R. Buss. Bounded arithmetic, cryptography and complexity. *Theoria*, 63(3):147–167, 1997. doi:[10.1111/j.1755-2567.1997.tb00745.x](https://doi.org/10.1111/j.1755-2567.1997.tb00745.x).
- [CD06] Xi Chen and Xiaotie Deng. Settling the complexity of two-player Nash equilibrium. In *FOCS*, pages 261–272. IEEE Computer Society, 2006. doi:[10.1109/FOCS.2006.69](https://doi.org/10.1109/FOCS.2006.69).
- [CdRN⁺23] Jonas Conneryd, Susanna F. de Rezende, Jakob Nordström, Shuo Pang, and Kilian Risse. Graph colouring is hard on average for Polynomial Calculus and Nullstellensatz. In *FOCS*, pages 1–11. IEEE, 2023. doi:[10.1109/FOCS57990.2023.00007](https://doi.org/10.1109/FOCS57990.2023.00007).
- [CHLR23] Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *STOC*, pages 1058–1066. ACM, 2023. doi:[10.1145/3564246.3585147](https://doi.org/10.1145/3564246.3585147).
- [CHR24] Lijie Chen, Shuichi Hirahara, and Hanlin Ren. Symmetric exponential time requires near-maximum circuit size. In *STOC’24—Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1990–1999. ACM, New York, [2024] ©2024. doi:[10.1145/3618260.3649624](https://doi.org/10.1145/3618260.3649624).
- [CJSW24] Lijie Chen, Ce Jin, Rahul Santhanam, and Ryan Williams. Constructive separations and their consequences. *TheoretCS*, 3, 2024. doi:[10.46298/THEORETICS.24.3](https://doi.org/10.46298/THEORETICS.24.3).
- [CK98] Mario Chiari and Jan Krajíček. Witnessing functions in bounded arithmetic and search problems. *J. Symb. Log.*, 63(3):1095–1115, 1998. doi:[10.2307/2586729](https://doi.org/10.2307/2586729).
- [CLMP25] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. In *STOC*, pages 554–564. ACM, 2025. doi:[10.1145/3717823.3718112](https://doi.org/10.1145/3717823.3718112).
- [CLO⁺23] Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1261–1270. IEEE, 2023. doi:[10.1109/FOCS57990.2023.00074](https://doi.org/10.1109/FOCS57990.2023.00074).
- [CLO24] Lijie Chen, Jiatu Li, and Igor C. Oliveira. Reverse mathematics of complexity lower bounds. In *FOCS*, pages 505–527. IEEE, 2024. doi:[10.1109/FOCS61266.2024.00040](https://doi.org/10.1109/FOCS61266.2024.00040).

- [CLW20] Lijie Chen, Xin Lyu, and R. Ryan Williams. Almost-everywhere circuit lower bounds from non-trivial derandomization. In *FOCS*, pages 1–12. IEEE, 2020. doi:[10.1109/FOCS46700.2020.00009](https://doi.org/10.1109/FOCS46700.2020.00009).
- [Cra36] Harald Cramér. On the order of magnitude of the difference between consecutive prime numbers. *Acta Arithmetica*, 2:23–46, 1936. URL: <http://eudml.org/doc/205441>.
- [CTW23] Lijie Chen, Roei Tell, and Ryan Williams. Derandomization vs refutation: A unified framework for characterizing derandomization. In *FOCS*, pages 1008–1047. IEEE, 2023. doi:[10.1109/FOCS57990.2023.00062](https://doi.org/10.1109/FOCS57990.2023.00062).
- [DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009. doi:[10.1137/070699652](https://doi.org/10.1137/070699652).
- [DR23] Ben Davis and Robert Robere. Colourful TFNP and propositional proofs. In Amnon Ta-Shma, editor, *38th Computational Complexity Conference, CCC 2023, July 17–20, 2023, Warwick, UK*, volume 264 of *LIPICs*, pages 36:1–36:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:[10.4230/LIPICs.CCC.2023.36](https://doi.org/10.4230/LIPICs.CCC.2023.36).
- [FGH⁺24] Lukáš Folwarczný, Mika Göös, Pavel Hubáček, Gilbert Maystre, and Weiqiang Yuan. One-way functions vs. TFNP: simpler and improved. In *15th Innovations in Theoretical Computer Science Conference*, volume 287 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 50, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2024. doi:[10.4230/lipics.itcs.2024.50](https://doi.org/10.4230/lipics.itcs.2024.50).
- [FGHS23] John Fearnley, Paul Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: $\text{CLS} = \text{PPAD} \cap \text{PLS}$. *J. ACM*, 70(1):7:1–7:74, 2023. doi:[10.1145/3568163](https://doi.org/10.1145/3568163).
- [FGMS20] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *J. Comput. Syst. Sci.*, 114:1–35, 2020. doi:[10.1016/J.JCSS.2020.05.007](https://doi.org/10.1016/J.JCSS.2020.05.007).
- [FGPR24] Noah Fleming, Stefan Grosser, Toniann Pitassi, and Robert Robere. Black-box PPP is not Turing-closed. In Bojan Mohar, Igor Shinkar, and Ryan O’Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24–28, 2024*, pages 1405–1414. ACM, 2024. doi:[10.1145/3618260.3649769](https://doi.org/10.1145/3618260.3649769).
- [FIM25] Noah Fleming, Deniz Imrek, and Christophe Marciot. Provably total functions in the polynomial hierarchy. In Srikanth Srinivasan, editor, *40th Computational Complexity Conference, CCC 2025, August 5–8, 2025, Toronto, Canada*, volume 339 of *LIPICs*, pages 28:1–28:40. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:[10.4230/LIPICs.CCC.2025.28](https://doi.org/10.4230/LIPICs.CCC.2025.28).
- [FKP19] Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic proofs and efficient algorithm design. *Found. Trends Theor. Comput. Sci.*, 14(1-2):1–221, 2019. doi:[10.1561/04000000086](https://doi.org/10.1561/04000000086).
- [GG11] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, TR11-136, 2011. URL: <https://eccc.weizmann.ac.il/report/2011/136/>.

- [GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. doi:[10.1145/6490.6503](https://doi.org/10.1145/6490.6503).
- [GGNS23] Karthik Gajulapalli, Alexander Golovnev, Satyajeet Nagargoje, and Sidhant Saraogi. Range avoidance for constant depth circuits: Hardness and algorithms. In *APPROX/RANDOM*, volume 275 of *LIPICs*, pages 65:1–65:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:[10.4230/LIPICS.APPROX/RANDOM.2023.65](https://doi.org/10.4230/LIPICS.APPROX/RANDOM.2023.65).
- [GH21] Paul W. Goldberg and Alexandros Hollender. The hairy ball problem is PPAD-complete. *Journal of Computer and System Sciences*, 122:34–62, 2021. doi:[10.1016/j.jcss.2021.05.004](https://doi.org/10.1016/j.jcss.2021.05.004).
- [GHJ⁺22] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and TFNP. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1150–1161, 2022. doi:[10.1109/FOCS54457.2022.00111](https://doi.org/10.1109/FOCS54457.2022.00111).
- [GHJ⁺24] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in TFNP. *SIAM J. Comput.*, 53(3):573–587, 2024. doi:[10.1137/22M1498346](https://doi.org/10.1137/22M1498346).
- [GKRS19] Mika Göös, Pritish Kamath, Robert Robere, and Dmitry Sokolov. Adventures in monotone complexity and TFNP. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPICs*, pages 38:1–38:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:[10.4230/LIPICS.ITCS.2019.38](https://doi.org/10.4230/LIPICS.ITCS.2019.38).
- [GL25] Surendra Ghentiyala and Zeyong Li. Hierarchies within TFNP: building blocks and collapses. *CoRR*, 2025. arXiv:[2507.21550](https://arxiv.org/abs/2507.21550).
- [GLW25] Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. *ACM Trans. Comput. Theory*, 17(2):14:1–14:23, 2025. doi:[10.1145/3718745](https://doi.org/10.1145/3718745).
- [GP18] Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. *J. Comput. Syst. Sci.*, 94:167–192, 2018. doi:[10.1016/J.JCSS.2017.12.003](https://doi.org/10.1016/J.JCSS.2017.12.003).
- [Gry19] Svyatoslav Gryaznov. Notes on resolution over linear equations. In *CSR*, volume 11532 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2019. doi:[10.1007/978-3-030-19955-5_15](https://doi.org/10.1007/978-3-030-19955-5_15).
- [Han04] Jiří Hanika. *Search Problems and Bounded Arithmetic*. PhD thesis, Charles University, Prague, 2004.
- [HKKS20] Pavel Hubáček, Chethan Kamath, Karel Král, and Veronika Slívová. On average-case hardness in TFNP from one-way functions. In *Theory of cryptography. Part III*, volume 12552 of *Lecture Notes in Comput. Sci.*, pages 614–638. Springer, Cham, [2020] ©2020. doi:[10.1007/978-3-030-64381-2_22](https://doi.org/10.1007/978-3-030-64381-2_22).
- [HKT24] Pavel Hubáček, Erfan Khaniki, and Neil Thapen. TFNP intersections through the lens of feasible disjunction. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical*

- Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPICs*, pages 63:1–63:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:[10.4230/LIPICs.ITCS.2024.63](https://doi.org/10.4230/LIPICs.ITCS.2024.63).
- [HL22] Max Hopkins and Ting-Chun Lin. Explicit lower bounds against $\Omega(n)$ -rounds of sum-of-squares. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 662–673. IEEE, 2022. doi:[10.1109/FOCS54457.2022.00069](https://doi.org/10.1109/FOCS54457.2022.00069).
 - [HV25] Edward A. Hirsch and Ilya Volkovich. Upper and lower bounds for the linear ordering principle. *CoRR*, 2025. arXiv:[2503.19188](https://arxiv.org/abs/2503.19188).
 - [HY20] Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *SIAM J. Comput.*, 49(6):1128–1172, 2020. doi:[10.1137/17M1118014](https://doi.org/10.1137/17M1118014).
 - [IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229. ACM, 1997. doi:[10.1145/258533.258590](https://doi.org/10.1145/258533.258590).
 - [Jeř04] Emil Jeřábek. Dual weak pigeonhole principle, Boolean complexity, and derandomization. *Ann. Pure Appl. Log.*, 129(1-3):1–37, 2004. doi:[10.1016/j.apal.2003.12.003](https://doi.org/10.1016/j.apal.2003.12.003).
 - [Jeř05] Emil Jeřábek. *Weak pigeonhole principle and randomized computation*. PhD thesis, Charles University in Prague, 2005.
 - [Jeř07a] Emil Jeřábek. Approximate counting in bounded arithmetic. *J. Symb. Log.*, 72(3):959–993, 2007. doi:[10.2178/JSL/1191333850](https://doi.org/10.2178/JSL/1191333850).
 - [Jeř07b] Emil Jeřábek. On independence of variants of the weak pigeonhole principle. *J. Log. Comput.*, 17(3):587–604, 2007. doi:[10.1093/LOGCOM/EXM017](https://doi.org/10.1093/LOGCOM/EXM017).
 - [Jeř16] Emil Jeřábek. Integer factoring and modular square roots. *J. Comput. Syst. Sci.*, 82(2):380–394, 2016. doi:[10.1016/J.JCSS.2015.08.001](https://doi.org/10.1016/J.JCSS.2015.08.001).
 - [JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988. doi:[10.1016/0022-0000\(88\)90046-3](https://doi.org/10.1016/0022-0000(88)90046-3).
 - [KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos H. Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, volume 185 of *LIPICs*, pages 44:1–44:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:[10.4230/LIPICs.ITCS.2021.44](https://doi.org/10.4230/LIPICs.ITCS.2021.44).
 - [KMPS25] Michal Koucký, Ian Mertz, Edward Pyne, and Sasha Sami. Collapsing catalytic classes. In *FOCS*, 2025. To appear. arXiv:[2504.08444](https://arxiv.org/abs/2504.08444).
 - [Kor21] Oliver Korten. The hardest explicit construction. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science—FOCS 2021*, pages 433–444. IEEE Computer Soc., Los Alamitos, CA, 2021. doi:[10.1109/FOCS52979.2021.00051](https://doi.org/10.1109/FOCS52979.2021.00051).
 - [Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *CCC*, volume 234 of *LIPICs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:[10.4230/LIPICs.CCC.2022.37](https://doi.org/10.4230/LIPICs.CCC.2022.37).

- [Kor25] Oliver Korten. Range avoidance and the complexity of explicit constructions. *Bull. EATCS*, 145, 2025. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/825>.
- [KP24] Oliver Korten and Toniann Pitassi. Strong vs. weak range avoidance and the linear ordering principle. In *FOCS*, pages 1388–1407. IEEE, 2024. doi:10.1109/FOCS61266.2024.00089.
- [Kra95] Jan Krajíček. Extensions of models of PV. In *Logic Colloquium*, volume 11 of *Lecture Notes in Logic*, pages 104–114. Springer, 1995. doi:10.1007/978-3-662-22108-2_8.
- [Kra04] Jan Krajíček. Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds. *J. Symb. Log.*, 69(1):265–286, 2004. doi:10.2178/jsl/1080938841.
- [Kri85] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985. doi:10.1007/BF00265682.
- [KT22] Leszek Aleksander Kolodziejczyk and Neil Thapen. Approximate counting and NP search problems. *J. Math. Log.*, 22(3):2250012:1–2250012:31, 2022. doi:10.1142/S021906132250012X.
- [Li24] Zeyong Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *STOC*, pages 2000–2007. ACM, 2024. doi:10.1145/3618260.3649615.
- [LLR24] Jiawei Li, Yuhao Li, and Hanlin Ren. Metamathematics of resolution lower bounds: A TFNP perspective. *CoRR*, abs/2411.15515, 2024. arXiv:2411.15515.
- [LO87] J. C. Lagarias and Andrew M. Odlyzko. Computing $\pi(x)$: An analytic method. *J. Algorithms*, 8(2):173–191, 1987. doi:10.1016/0196-6774(87)90037-X.
- [LOS21] Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *STOC*, pages 303–316. ACM, 2021. doi:10.1145/3406325.3451085.
- [LPR24] Yuhao Li, William Pires, and Robert Robere. Intersection classes in TFNP and proof complexity. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA*, volume 287 of *LIPICs*, pages 74:1–74:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ITCS.2024.74.
- [LPT24] Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness. In *FOCS*, pages 1–13. IEEE, 2024. doi:10.1109/FOCS61266.2024.00095.
- [Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987. doi:10.1007/3-540-48184-2_32.
- [Mer23] Ian Mertz. Reusing space: Techniques and open problems. *Bull. EATCS*, 141, 2023. URL: <http://eatcs.org/beatcs/index.php/beatcs/article/view/780>.

- [Mül21] Moritz Müller. Typical forcings, NP search problems and an extension of a theorem of riis. *Ann. Pure Appl. Log.*, 172(4):102930, 2021. URL: <https://doi.org/10.1016/j.apal.2020.102930>, doi:10.1016/J.APAL.2020.102930.
- [NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.
- [OS17] Igor C. Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. In *STOC*, pages 665–677, 2017. doi:10.1145/3055399.3055500.
- [Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.
- [Pic15] Ján Pich. Circuit lower bounds in bounded arithmetics. *Ann. Pure Appl. Log.*, 166(1):29–45, 2015. doi:10.1016/J.APAL.2014.08.004.
- [Pot20] Aaron Potechin. Sum of squares bounds for the ordering principle. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPICs*, pages 38:1–38:37. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.38.
- [PPY23] Amol Pasarkar, Christos H. Papadimitriou, and Mihalis Yannakakis. Extremal combinatorics, iterated pigeonhole arguments and generalizations of PPP. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPICs*, pages 88:1–88:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ITCS.2023.88.
- [PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. In *FOCS*, pages 989–1007. IEEE, 2023. doi:10.1109/FOCS57990.2023.00061.
- [PS23] Ján Pich and Rahul Santhanam. Towards $P \neq NP$ from Extended Frege lower bounds. *Electron. Colloquium Comput. Complex.*, TR23-199, 2023. URL: <https://eccc.weizmann.ac.il/report/2023/199>.
- [PT19] Pavel Pudlák and Neil Thapen. Random resolution refutations. *Comput. Complex.*, 28(2):185–239, 2019. doi:10.1007/S00037-019-00182-7.
- [Pud15] Pavel Pudlák. On the complexity of finding falsifying assignments for Herbrand disjunctions. *Arch. Math. Log.*, 54(7-8):769–783, 2015. doi:10.1007/S00153-015-0439-6.
- [PWW88] 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(4):1235–1244, 1988. doi:10.1017/S0022481200028061.
- [Pyn24] Edward Pyne. Derandomizing logspace with a small shared hard drive. In *CCC*, volume 300 of *LIPICs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.CCC.2024.4.

- [Raz87] Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.
- [Rii01] Søren Riis. A complexity gap for tree resolution. *Comput. Complex.*, 10(3):179–209, 2001. [doi:10.1007/S00037-001-8194-Y](https://doi.org/10.1007/S00037-001-8194-Y).
- [RRV02] Ran Raz, Omer Reingold, and Salil P. Vadhan. Extracting all the randomness and reducing the error in Trevisan’s extractors. *J. Comput. Syst. Sci.*, 65(1):97–128, 2002. [doi:10.1006/JCSS.2002.1824](https://doi.org/10.1006/JCSS.2002.1824).
- [RSW22] Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *FOCS*, pages 640–650. IEEE, 2022. [doi:10.1109/FOCS54457.2022.00067](https://doi.org/10.1109/FOCS54457.2022.00067).
- [Smo87] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82. ACM, 1987. [doi:10.1145/28395.28404](https://doi.org/10.1145/28395.28404).
- [STV01] Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001. [doi:10.1006/JCSS.2000.1730](https://doi.org/10.1006/JCSS.2000.1730).
- [TCH12] Terence Tao, Ernest Croot, III, and Harald Helfgott. Deterministic methods to find primes. *Mathematics of Computation*, 81(278):1233–1246, 2012. [doi:10.1090/S0025-5718-2011-02542-1](https://doi.org/10.1090/S0025-5718-2011-02542-1).
- [Tha02] Neil Thapen. *The weak pigeonhole principle in models of bounded arithmetic*. PhD thesis, University of Oxford, 2002.
- [Tha24] Neil Thapen. How to fit large complexity classes into TFNP. *CoRR*, abs/2412.09984, 2024. [arXiv:2412.09984](https://arxiv.org/abs/2412.09984).
- [Uma03] Christopher Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003. [doi:10.1016/S0022-0000\(03\)00046-1](https://doi.org/10.1016/S0022-0000(03)00046-1).

A Herbrandization

Herbrandization is a basic construction in logic. Roughly speaking, any $\text{TF}\Sigma_2^P$ problem L can be captured by a logical formula $\forall x \exists y \forall z \varphi(x, y, z)$,¹² where the task of L is, given x , to find a y such that $\forall z \varphi(x, y, z)$ holds. To Herbrandize this formula, we add another function h (treated as an input oracle) and consider the formula $\forall x \exists y \varphi(x, y, h(y))$. One can then define a TFNP problem by treating h as a first-order variable like x , where the task is, given x and h , to find a y such that $\varphi(x, y, h(y))$ holds.

¹²Formally, they are called “ $\forall\Sigma_2^b$ -formulas”, where the subscript 2 stands for two alternations ($\forall\exists\forall$) and the superscript b stands for “bounded”, i.e., the lengths of x, y, z are polynomially related and φ is a polynomial-time predicate.

$\text{TF}\Sigma_2^P$ search problem	TFNP problem via Herbrandization
$\forall x \exists y \forall z \varphi(x, y, z)$	$\forall x, h \exists y \varphi(x, y, h(y))$
AVOID: $\forall D \exists x \forall y D(y) \neq x$	LOSSY-CODE: $\forall C, D \exists x D(C(x)) \neq x$
(1): $\forall F \exists x \forall y (F(F(y)) \neq F(x) \vee F(y) = x)$	NEPHEW: $\forall F, G \exists x (F(F(G(x))) \neq F(x) \vee F(G(x)) = x)$

Table 1: Some examples of TFNP problems via Herbrandization.

B Proof Complexity Characterizations of Randomized Reductions

In this section we prove [Theorem 3.4](#), which we restate next.

Theorem B.1. If a proof system \mathcal{P} is characterized by the total search problems reducible to $R \in \text{TFNP}^{dt}$, then $r\mathcal{P}$ is characterized by the total search problems that are randomized-reducible to R .

Proof. Fix a complexity c $r\mathcal{P}$ proof \mathcal{D} of a CNF formula H . We will construct a complexity $\Theta(c)$ randomized reduction from SEARCH_H to a complete problem SEARCH_F for \mathcal{C} . On input $x \in \{0, 1\}^n$ the reduction first samples (Π, B) from the distribution \mathcal{D} given by the $r\mathcal{P}$ proof, where Π is a complexity c \mathcal{P} -proof of $H \wedge B$. Since by assumption \mathcal{P} is characterized by \mathcal{C} , this implies that there is a complexity $O(c)$ reduction $\mathcal{T} = (T, \{T_o\})$ from $\text{SEARCH}_{H \wedge B}$ to SEARCH_F . Relabel each leaf of an output decision tree T_o in \mathcal{T} , which is labelled with a clause of B , by \perp , indicating a failure event of the randomized reduction.

Observe that property (1) (correctness) of a randomized reduction to SEARCH_H is satisfied. It remains to argue that property (2) (error probabilities) is also satisfied. This follows since the reduction was constructed from a $r\mathcal{P}$ proof. Indeed, the probability that the reduction fails is the probability that we sampled a reduction $\mathcal{T} = (T, \{T_o\})$ and the leaf of the output decision tree that we arrive at when following x is labelled by \perp . By construction, we arrive at a \perp leaf only if we falsify a clause of the corresponding CNF B . However, by the definition of an $r\mathcal{P}$ proof, for every x the probability that every clause in B is satisfied by x is at least $2/3$.

For the converse, let SEARCH_F be a \mathcal{C} -complete problem and let \mathcal{D} be a randomized reduction from SEARCH_H to SEARCH_F of complexity c . We will argue that each $\mathcal{T} \sim \mathcal{D}$ is a deterministic reduction from $\text{SEARCH}_{H \wedge B}$ to SEARCH_F for some CNF formula B of width $O(c^2)$. Because \mathcal{C} is characterized by \mathcal{P} , we will obtain a \mathcal{P} proof of $H \wedge B$, and putting these together, a $r\mathcal{P}$ proof of H .

Let $F = C_1 \wedge \dots \wedge C_m$ and let $\mathcal{T} \sim \mathcal{D}$ where $\mathcal{T} := (\{T_i\}_{i \in [n]}, \{T_o\}_{o \in [m]})$. Following [\[BF123\]](#), let the *reduced formula* $F_{\mathcal{T}}$ be the CNF formula obtained as follows: for each clause $C \in F$, let the decision tree T^C be obtained by sequentially running the decision trees T_i for each $i \in \text{vars}(C)$ to obtain an assignment $\alpha \in \{0, 1\}^{\text{vars}(C)}$. If $C(\alpha) = b \in \{0, 1\}$ then label this a “ b -leaf”, for $b \in \{0, 1\}$. Say that a root-to-leaf path $p \in T^C$ is a b -path if it ends at a b -leaf. Define

$$C(\mathcal{T}) := \bigwedge_{\text{0-path } p \in T^C} \neg p$$

In words, $C(\mathcal{T})$ says that the clause C , after substituting \mathcal{T} for the variables, is never falsified. The reduced CNF formula is

$$F_{\mathcal{T}} := \bigwedge_{o \in [m]} \left(C_o(\mathcal{T}) \vee \bigwedge_{p \in T_o} \neg p \right).$$

$F_{\mathcal{T}}$ formalizes the definition of a reduction (Equation 2) by \mathcal{T} to SEARCH_F —if we falsify $C_o(\mathcal{T})$ and follow path p in T_o then the label of the leaf of p is a valid solution to the search problem reducing to SEARCH_F by \mathcal{T} . Hence, \mathcal{T} is a reduction from a CNF formula $H \wedge B$ to F iff each clause of $F_{\mathcal{T}}$ is a weakening¹³ of a clause of $H \wedge B$.

Then, knowing H , we can recover B as follows: let B be the set of all clauses of $F_{\mathcal{T}}$ which are not a weakening of any clause of H . Then \mathcal{T} is a reduction from $\text{SEARCH}_{H \wedge B}$ to SEARCH_F . Note that the width of the clauses in B is at most $O(c^2)$ as every clause in F has width $O(c)$, the trees in \mathcal{T} have depth $O(c)$, and we have substituted the decision trees for the variables of the clause.

It remains to argue that for every $x \in \{0, 1\}^n$, $\Pr_{(\Pi, B) \sim \mathcal{D}}[B(x) = 1] \geq 2/3$. This is immediate from the fact that \mathcal{T} is a randomized reduction to SEARCH_F . Indeed, each clause $K \in F_{\mathcal{T}}$ comes from some $C_o(\mathcal{T}) \vee \neg p$ for $p \in T_o$ and some $o \in [m]$. That is, $K = \neg p^* \vee \neg p$ for some 0-path $p^* \in T^C$. Hence, by the definition of a randomized reduction,

$$\begin{aligned} 2/3 &\leq \Pr_{\mathcal{T} \sim \mathcal{D}} [\forall o \in [m] : (o, T(x)) \notin \text{SEARCH}_F \vee T_o(x) \neq \perp] \\ &= \Pr_{\mathcal{T} \sim \mathcal{D}} [\forall o \in [m] : C_o(\mathcal{T}(x)) \neq 0 \vee T_o(x) \neq \perp] \\ &= \Pr_{\mathcal{T} \sim \mathcal{D}} [\forall o \in [m], \forall \text{ 0-paths } p^* \in T^{C_o}, \forall \perp\text{-paths } p \in T_o : \neg p^*(x) = 1 \vee \neg p(x) = 1] \\ &= \Pr_{\mathcal{T} \sim \mathcal{D}} [\forall K \in B : K(x) = 1]. \end{aligned} \quad (B \text{ are clauses of } F_{\mathcal{T}})$$

□

C Proof of Theorem 6.2

Weak designs. We say that $I_1, I_2, \dots, I_m \subseteq [d]$ is a *weak* (ℓ, ρ) -design if:

- For every $i \leq m$, $|I_i| = \ell$; and
- For every $i \leq m$,

$$\sum_{j < i} 2^{|S_i \cap S_j|} \leq \rho \cdot (m - 1).$$

Theorem C.1 ([RRV02]). For every $\ell, m \in \mathbb{N}$ and $\rho > 1$, there is a weak (ℓ, ρ) -design $S_1, S_2, \dots, S_m \subseteq [d]$ with

$$d = \left\lceil \frac{\ell}{\ln \rho} \right\rceil \cdot \ell.$$

Moreover, such a family can be found in deterministic time $\text{poly}(m, d)$.

List-decodable codes. A pair of functions (Enc, Dec) is called an $(L, 1/2 - \varepsilon)$ -list-decodable code if:

- $\text{Enc} : \{0, 1\}^n \rightarrow \{0, 1\}^{2^\ell}$ and $\text{Dec} : \{0, 1\}^{2^\ell} \rightarrow (\{0, 1\}^n)^L$ are computable in deterministic polynomial time, and
- for every $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^{2^\ell}$ such that y is $(1/2 - \varepsilon)$ -close to $\text{Enc}(x)$, $\text{Enc}(x)$ appears in the list $\text{Dec}(y)$.

Theorem C.2 ([STV01]). For every $n \in \mathbb{N}$ and $\varepsilon > 0$, there exists an $(L, 1/2 - \varepsilon)$ -list-decodable code with $\ell = O(\log(n/\varepsilon))$ and $L = \text{poly}(1/\varepsilon)$.

¹³A clause C is a weakening of a clause D if the literals of D are a subset of the literals of C .

In what follows, for notational convenience, we will think of length- 2^ℓ strings $f \in \{0, 1\}^{2^\ell}$ as (the truth tables of) ℓ -bit Boolean functions $f : \{0, 1\}^\ell \rightarrow \{0, 1\}$.

Theorem 6.2. Let $n, m \in \mathbb{N}$, $\varepsilon > 0$, and $\rho > 1$ be parameters. Let $d := O\left(\frac{\log^2(nm/\varepsilon)}{\log \rho}\right)$ and $k := d + (\rho + 1)(m - 1) + O(\log(m/\varepsilon))$. Let $N := 2^n$, $M := 2^m$, $K := 2^k$, and $D := 2^d$.

Then there is a (k, ε) -reconstructive generator $\text{PRG} : [N] \times [D] \rightarrow [M]$ with $\text{poly}(\rho mn/\varepsilon)$ -time deterministic oracle algorithms $G_<, H_<, G_>, H_>, \text{Comp}, \text{Decomp}$ such that the following holds. For every set $S \subseteq [M]$, every $f \in [N]$, and every input w witnessing that either

$$[val] \times [D] \xrightleftharpoons[G_<^S(f, -)]{H_<^S(f, -)} (S \dot{\cup} [\varepsilon M]) \times [D] \quad \text{or} \quad S \times [D] \xrightleftharpoons[G_>^S(f, -)]{H_>^S(f, -)} ([val] \dot{\cup} [\varepsilon M]) \times [D]$$

is not an injection-surjection pair (where $val := |S \cap \text{PRG}_f| \cdot \frac{M}{D}$), we have $\text{Comp}^S(f, w) \in [K]$ and

$$\text{Decomp}^S(\text{Comp}^S(f, w)) = f.$$

Proof. Let ε' be the biggest (inverse) power of 2 such that $\varepsilon' \leq \varepsilon/m$. Let (Enc, Dec) be an $(L, 1/2 - \varepsilon')$ -list-decodable code guaranteed by [Theorem C.2](#) with $\ell = O(\log n/\varepsilon') = O(\log(nm/\varepsilon))$ and $L \leq \text{poly}(1/\varepsilon') \leq \text{poly}(m/\varepsilon)$. Let $I_1, I_2, \dots, I_m \subseteq [d]$ be a weak (ℓ, ρ) -design guaranteed by [Theorem C.1](#) with $d = O(\ell^2/\log \rho)$. Given $f \in \{0, 1\}^n$ and $z \in \{0, 1\}^d$ as inputs, the generator first computes $\tilde{f} := \text{Enc}(f)$, and then outputs

$$\text{PRG}(f, z) := (\tilde{f}(z|_{I_1}), \tilde{f}(z|_{I_2}), \dots, \tilde{f}(z|_{I_m})).$$

The classical proof of Nisan–Wigderson [\[NW94\]](#) shows that for every $S \subseteq \{0, 1\}^m$ and every $f \in \{0, 1\}^n$ that is worst-case hard against S -oracle circuits, f provides an additive approximation of $|S|$, i.e.,

$$\left| \frac{|S \cap \text{PRG}_f|}{D} - \frac{|S|}{M} \right| \leq \varepsilon.$$

The *proof* of the above fact goes through a *hybrid* argument and considers the following intermediate generators. For each $0 \leq i \leq m$, let $\text{Hyb}_i : \{0, 1\}^d \times \{0, 1\}^m \rightarrow \{0, 1\}^m$ denote the generator that takes $z \in \{0, 1\}^d$ and $r \in \{0, 1\}^m$ as inputs, and outputs

$$\text{Hyb}_i(z, r) := (\tilde{f}(z|_{I_1}), \dots, \tilde{f}(z|_{I_i}), r_{i+1}, \dots, r_m).$$

Let $V_i := \{(z, r) : \text{Hyb}_i(z, r) \in S\}$; intuitively, $|V_i|$ is an estimation of $|S| \cdot 2^d$ by the generator Hyb_i . Suppose f is hard, then for every $1 \leq i \leq m$, $|V_{i-1}|$ is close to $|V_i|$. It follows that $|V_0|$ is close to $|V_m|$. Since $|V_0| = 2^d \cdot |S|$ and $|V_m| = |S \cap \text{PRG}_f| \cdot 2^m = val \cdot 2^d$, val is a good estimation of $|S|$.

In a nutshell, the proof of [\[Jeř07a, Theorem 2.7\]](#) proceeds by constructing injection-surjection pairs witnessing $|V_{i-1}| \lesssim |V_i|$ and $|V_{i-1}| \gtrsim |V_i|$ for each i . Composing all these injection-surjection pairs gives the final injection-surjection pairs witnessing $|V_0| \lesssim |V_m|$ and $|V_0| \gtrsim |V_m|$ respectively.

Fix $1 \leq i \leq m$, we now aim to construct injection-surjection pairs witnessing the inequalities $|V_{i-1}| \lesssim |V_i|$ and $|V_{i-1}| \gtrsim |V_i|$. That is, letting $\text{diff} := \varepsilon' \cdot 2^{m+d}$, we will construct injection-surjection pairs $V_i \xrightleftharpoons[G_i]{H_i} V_{i-1} \dot{\cup} [\text{diff}]$ and $V_{i-1} \xrightleftharpoons[G'_i]{H'_i} V_i \dot{\cup} [\text{diff}]$ that depends on f . Moreover, if f is indeed a “hard function”, then these injection-surjection pairs will be valid, i.e., $G_i \circ H_i$ is the identity map on V_i and $G'_i \circ H'_i$ is the identity map on V_{i-1} .

Let $z \in \{0, 1\}^d$ and $r \in \{0, 1\}^m$, we denote $z' := z|_{I_i}$ and $\text{aux} := (z|_{[d] \setminus I_i}, r|_{[m] \setminus i})$. Note that there is a one-to-one correspondence between (z, r) and (z', r_i, aux) . Now given i, aux , we define

$$X_{i, \text{aux}} := \{(z', r_i) : (\tilde{f}(z|_{I_1}), \tilde{f}(z|_{I_2}), \dots, \tilde{f}(z|_{I_{i-1}}), r_i, r_{i+1}, \dots, r_m) \in S\} \text{ and}$$

$$Y_{i,\text{aux}} := \{(z', b) : (z', \tilde{f}(z')) \in X_{i,\text{aux}}\}.$$

Then, $V_{i-1} = \bigcup_{\text{aux}} (X_{i,\text{aux}} \times \{\text{aux}\})$ and $V_i = \bigcup_{\text{aux}} (Y_{i,\text{aux}} \times \{\text{aux}\})$. To show that $|V_{i-1}| \approx |V_i|$, it suffices to show that $|X_{i,\text{aux}}| \approx |Y_{i,\text{aux}}|$ for every aux (under the assumption that f is “hard”); this is exactly what the next claim shows.

Claim C.3. If $||X_{i,\text{aux}}| - |Y_{i,\text{aux}}|| > \varepsilon' 2^{\ell+1}$, then given (i, aux) and additional $\rho \cdot (m-1) + 2$ advice bits, it is possible to recover a string \tilde{f}_{apx} that is $(1/2 + \varepsilon')$ -close to \tilde{f} in deterministic $\text{poly}(\rho m, 2^\ell)$ time with oracle access to S . Moreover, given (i, aux) and \tilde{f} , the additional advice bits can be computed in deterministic $\text{poly}(\rho m, 2^\ell)$ time with oracle access to S .

Proof Sketch. For every $b \in \{0, 1\}$, define a string $\tilde{f}^b \in \{0, 1\}^{2^\ell}$, where for every $z' \in \{0, 1\}^\ell$, the z' -th bit of \tilde{f}^b is 1 if and only if $(z', b) \in X_{i,\text{aux}}$. It can be shown that

$$|X_{i,\text{aux}}| - |Y_{i,\text{aux}}| = \Delta(\tilde{f}^1, \tilde{f}) - \Delta(\tilde{f}^0, \tilde{f}),$$

where $\Delta(\cdot, \cdot)$ denotes the Hamming distance of two binary strings. Since $||X_{i,\text{aux}}| - |Y_{i,\text{aux}}|| > \varepsilon' 2^{\ell+1}$, there must be some $b \in \{0, 1\}$ such that $\Delta(\tilde{f}^b, \tilde{f}) \notin [(1/2 - \varepsilon')2^\ell, (1/2 + \varepsilon')2^\ell]$.

Now suppose that \tilde{f} is fixed and (i, aux) is given. For each $j < i$, $\tilde{f}(z|_{I_j})$ is a function over z' that only depends on $|S_i \cap S_j|$ bits of z' . Hence, the truth table of this function can be recorded in $2^{|S_i \cap S_j|}$ bits. If we write down the truth tables of $\tilde{f}(z|_{I_j})$ for every $j < i$ as advice, this only costs

$$\sum_{j < i} 2^{|S_i \cap S_j|} \leq \rho \cdot (m-1)$$

advice bits. We append two additional advice bits $b, b' \in \{0, 1\}$, where b indicates that $\Delta(\tilde{f}^b, \tilde{f}) \notin [(1/2 - \varepsilon')2^\ell, (1/2 + \varepsilon')2^\ell]$ and b' indicates whether $\Delta(\tilde{f}^b, \tilde{f})$ is above $1/2$ or not. It is easy to see that we can recover a string that is $(1/2 + \varepsilon')$ -close to \tilde{f} given (i, aux) and these advice bits; moreover, these advice bits can be computed in deterministic polynomial time given (i, aux) and f as inputs. \diamond

Composing the above claim with the list-decodable code (Enc, Dec) , we obtain the following corollary:

Corollary C.4. If $||X_{i,\text{aux}}| - |Y_{i,\text{aux}}|| > \varepsilon' 2^{\ell+1}$, then given (i, aux) and additional $\rho \cdot (m-1) + \log L + 2$ advice bits, it is possible to compute f in deterministic $\text{poly}(\rho m, 2^\ell)$ time with oracle access to S . Moreover, given (i, aux) and f , the additional advice bits can be computed in deterministic $\text{poly}(\rho m, 2^\ell)$ time with oracle access to S . \diamond

Let $\text{Decomp}^S(i, \text{aux}, \alpha)$ be the procedure for computing f from (i, aux) and the advice bits α as asserted in Corollary C.4. We now say f is “hard” if f is not in the range of Decomp^S . (Note that Decomp^S takes $k := \log m + d - \ell + m - 1 + \rho(m-1) + \log L + 2 \leq d + (\rho+1)(m-1) + O(\log(m/\varepsilon))$ bits. As long as this is less than n bits, a hard f must exist.)

Suppose that we are given some f that is hard. We can create injection-surjection pairs between $X_{i,\text{aux}}$ and $Y_{i,\text{aux}}$ by brute force; this only takes deterministic $\text{poly}(2^\ell)$ time with oracle access to S . As an example, we construct

$$Y_{i,\text{aux}} \xrightleftharpoons[G_{i,\text{aux}}]{H_{i,\text{aux}}} X_{i,\text{aux}} \dot{\cup} [\varepsilon' \cdot 2^{\ell+1}].$$

- $G_{i,\text{aux}}(v)$: If $v \in X_{i,\text{aux}}$ then let p be the integer such that v is the (lexicographically) p -th smallest element of $X_{i,\text{aux}}$ (the smallest element is the 0-th); if $v \in [\varepsilon' \cdot 2^{\ell+1}]$ then let $p := |X_{i,\text{aux}}| + v$. Return the p -th smallest element of $Y_{i,\text{aux}}$; if $|Y_{i,\text{aux}}| \geq p$ then return an arbitrary element (say the smallest one).

- $H_{i,\text{aux}}(v)$: Suppose that v is the p -th smallest element of $Y_{i,\text{aux}}$. If $p < |X_{i,\text{aux}}|$ then return the p -th smallest element of $X_{i,\text{aux}}$; otherwise return $p - |X_{i,\text{aux}}| \in [\varepsilon' \cdot 2^{\ell+1}]$.
- It is straightforward to verify that $G_{i,\text{aux}} \circ H_{i,\text{aux}}$ is the identity map and that $G_{i,\text{aux}}$ and $H_{i,\text{aux}}$ are computable in deterministic $\text{poly}(2^\ell)$ time.

We can similarly construct $X_{i,\text{aux}} \xrightleftharpoons[G'_{i,\text{aux}}]{H'_{i,\text{aux}}} Y_{i,\text{aux}} \dot{\cup} [\varepsilon' 2^{\ell+1}]$ such that $G'_{i,\text{aux}} \circ H'_{i,\text{aux}}$ is the identity map and $G'_{i,\text{aux}}$ and $H'_{i,\text{aux}}$ are computable in deterministic $\text{poly}(2^\ell)$ time. Now we describe the functions G_i, H_i, G'_i, H'_i .

- Let v be the input of G_i . If $v \in V_{i-1}$ then write $v = (z, r) = (z', r_i, \text{aux})$; if $v \in [\text{diff}]$ then let $\text{aux} := \lfloor v / (\varepsilon' 2^{\ell+1}) \rfloor$ (treated as both a number in $[2^{m+d-\ell-1}]$ and a length- $(m+d-\ell-1)$ string) and $v' := v - \text{aux} \cdot \varepsilon' 2^{\ell+1}$. Assuming f is hard, we have $v \in X_{i,\text{aux}} \dot{\cup} [\varepsilon' 2^{\ell+1}]$. Let $u := G_{i,\text{aux}}(v) \in Y_{i,\text{aux}}$, write $u = (z^u, r_i^u)$ and return $G_i(v) := (z^u, r_i^u, \text{aux})$.
- Let $u \in V_i$ be the input of H_i , and write $u := (z', r_i, \text{aux})$ where $(z', r_i) \in Y_{i,\text{aux}}$. We can compute $v := H_{i,\text{aux}}(z', r_i) \in X_{i,\text{aux}} \dot{\cup} [\varepsilon' 2^{\ell+1}]$. If $v \in X_{i,\text{aux}}$ then we write $v = (z^v, r_i^v)$ and return $(z^v, r_i^v, \text{aux}) \in V_{i-1}$; if $v \in [\varepsilon' 2^{\ell+1}]$ then we return $\text{aux} \cdot \varepsilon' 2^{\ell+1} + v \in [\text{diff}]$.
- The definitions of G'_i, H'_i are analogous.

It is easy to see that if f is indeed hard, then $G_i \circ H_i$ is the identity map. However, if f is *not hard*, there is no guarantee that $G_i \circ H_i$ is the identity map. Nevertheless we still gain something: Given any witness $u \in V_i$ such that $G_i(H_i(u)) \neq u$, if we write $u = (z', r_i, \text{aux})$ then we have $||X_{i,\text{aux}}| - |Y_{i,\text{aux}}|| > \varepsilon' 2^{\ell+1}$. By [Corollary C.4](#), we can compute (i, aux, α) from this witness u deterministically such that $\text{Decomp}^S(i, \text{aux}, \alpha) = f$, i.e., we found a *witness for the non-hardness of f* as well! In summary, let f be a purported hard function, we can *either* use f to perform approximate counting and obtain injection-surjection pairs (G_i, H_i) *or*, if (G_i, H_i) fails to be an injection-surjection pair, *exploit this failure* to compress f .

Finally, we can compose the functions $\{G_i\}$, $\{H_i\}$, $\{G'_i\}$, and $\{H'_i\}$ to obtain $G_{<}, H_{<}, G_{>}, H_{>}$.

- Let $v \in (S \dot{\cup} [\varepsilon M]) \times [D] = V_0 \dot{\cup} [m \cdot \text{diff}]$ be the input of $G_{<}$. For each i from 1 to m , if currently we have $v \in V_{i-1} \dot{\cup} [\text{diff}]$, then we update $v \leftarrow G_i(v)$; otherwise $v \in [\text{diff}, m \cdot \text{diff}]$ and we update $v \leftarrow v - \text{diff}$.
- Let $v \in V_m = [\text{val}] \times [D]$ be the input of $H_{<}$. For each i from m down to 1, if currently we have $v \in V_i$, then we update $v \leftarrow H_i(v)$; otherwise v is some number in $[m \cdot \text{diff}]$ and we update $v \leftarrow v + \text{diff}$.
- The definitions of $G_{>}, H_{>}$ are analogous.

It is easy to see that the functions

$$[\text{val}] \times [D] \xrightleftharpoons[G_{<}]{H_{<}} (S \dot{\cup} [\varepsilon M]) \times [D]$$

satisfy the following property: given any $w \in [\text{val}] \times [D]$ such that $G_{<}(H_{<}(w)) \neq w$, we can compute in deterministic $\text{poly}(n, 2^\ell) = \text{poly}(n/\varepsilon)$ time a “compression” $\text{Comp}^S(f, w)$ of f that decompresses to f via Decomp^S . Furthermore, $G_{<}$ and $H_{<}$ themselves can be computed in deterministic $\text{poly}(n/\varepsilon)$ time. The conclusions for $G_{>}$ and $H_{>}$ can be proved similarly. \square