Electronic Colloquium on Computational Complexity, Report No. 123 (2025)

# Hierarchies within TFNP: building blocks and collapses

Surendra Ghentiyala* Zeyong Li†

**Abstract**

We initiate the study of complexity classes $\mathsf{A}^{\mathsf{B}}$ where $\mathsf{A}$ and $\mathsf{B}$ are both $\mathsf{TFNP}$ subclasses. For example, we consider complexity classes of the form $\mathsf{PPP}^{\mathsf{PPP}}$, $\mathsf{PPAD}^{\mathsf{PPA}}$, and $\mathsf{PPA}^{\mathsf{PLS}}$. We define complete problems for such classes, and show that they belong in $\mathsf{TFNP}$. These definitions require some care, since unlike a class like $\mathsf{PPA}^{\mathsf{NP}}$, where the $\mathsf{NP}$ oracle defines a function, in $\mathsf{PPA}^{\mathsf{PPP}}$, the oracle is for a search problem with many possible solutions. Intuitively, the definitions we introduce quantify over all possible instantiations of the $\mathsf{PPP}$ oracle.

With these notions in place, we then show that several $\mathsf{TFNP}$ subclasses are self-low. In particular, $\mathsf{PPA}^{\mathsf{PPA}} = \mathsf{PPA}$, $\mathsf{PLS}^{\mathsf{PLS}} = \mathsf{PLS}$, and $\mathsf{LOSSY}^{\mathsf{LOSSY}} = \mathsf{LOSSY}$. These ideas introduce a novel approach for classifying computational problems within $\mathsf{TFNP}$, such as the problem of deterministically generating large primes.

*Cornell University. Email: sg974@cornell.edu. This work is supported in part by the NSF under Grants Nos. CCF-2122230 and CCF-2312296, a Packard Foundation Fellowship, and a generous gift from Google.

†National University of Singapore. Email: li.zeyong@u.nus.edu. Supported by NRF grant NRF-NRFI09-0005.

ISSN 1433-8092

# Contents

# 1 Introduction

Oracles and oracle classes are an indispensable tool in complexity theory. For example, the polynomial hierarchy $\mathsf{PH}$, which can be defined in terms of oracles ($\Sigma_0^{\mathsf{P}} = \mathsf{P}, \Sigma_{i+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$), is related to a wide variety of complexity classes and important problems. Just to name a few, the Sipser-Lautemann theorem [Lau83] states that $\mathsf{BPP} \subseteq \Sigma_2^{\mathsf{P}} \cap \Pi_2^{\mathsf{P}}$, and the circuit minimization problem is known to be in $\Sigma_2^{\mathsf{P}}$ but not known to be in $\mathsf{NP}$.

Furthermore, the assumption that the polynomial hierarchy does not collapse—that $\Sigma_i^{\mathsf{P}} \neq \Sigma_{i+1}^{\mathsf{P}}$ for any integer $i > 0$—is now a standard assumption in complexity theory. For example, the original proof that the graph isomorphism problem is not $\mathsf{NP}$-complete is based on the assumption that the polynomial hierarchy does not collapse to $\Sigma_2^{\mathsf{P}}$ [BHZ87].

Oracles are also useful as evidence that two complexity classes $\mathsf{A}$ and $\mathsf{A}'$ are not equal. Since unconditional separations between complexity classes are often challenging, oracle separations, the existence of an oracle $\mathcal{O}$ such that $\mathsf{A}^{\mathcal{O}} \neq \mathsf{A}'^{\mathcal{O}}$, are often viewed as (weak) evidence that two complexity classes are indeed separate.

While oracles and oracle classes have been a mainstay of theoretical computer science since their introduction in [Tur39], one class of problems has only recently begun to be explored through oracles: $\mathsf{TFNP}$ (total function nondeterministic polynomial time).

Informally, $\mathsf{TFNP}$ is the class of efficiently verifiable search problems for which a solution always exists. One should have in mind a problem like Pigeon and its associated complexity class, $\mathsf{PPP}$ (Polynomial Pigeonhole Principle).

**Definition 1.1.** The search problem Pigeon is defined as follows. Given a $\mathrm{poly}(n)$ size circuit $C : \{0,1\}^n \to \{0,1\}^n$, output one of the following.

1. $x$ s.t. $C(x) = 0^n$;
2. distinct $x_1, x_2$ s.t. $C(x_1) = C(x_2)$.

$\mathsf{PPP}$ is defined as all search problems which are many-to-one reducible to Pigeon.

Notice that a solution to Pigeon always exists since if $C$ is surjective, then a type 1 solution exists, and if $C$ is not surjective, a type 2 solution exists by the pigeonhole principle. Pigeon therefore always has a solution and is what we refer to as a total function. Furthermore, any solution to Pigeon is clearly efficiently verifiable since it simply requires evaluating $C$ at most twice followed by some polynomial time computation. Observe that the names Pigeon and the polynomial pigeonhole principle are fitting, since Pigeon can be viewed as the search problem analogue of the pigeonhole principle.

$\mathsf{PPP}$ is only one of many $\mathsf{TFNP}$ subclasses, some others include $\mathsf{PPA}$, $\mathsf{PPAD}$, $\mathsf{PPADS}$, and $\mathsf{PLS}$. Just as $\mathsf{PPP}$ can be viewed as the algorithmization of the pigeonhole principle, each of these $\mathsf{TFNP}$ subclasses can be viewed as the algorithmization of a distinct mathematical principle (e.g. parity arguments for $\mathsf{PPA}$). Since $\mathsf{TFNP}$ is believed to not have any complete problems [Pud15], our understanding of $\mathsf{TFNP}$ is primarily advanced by understanding the complexity of these $\mathsf{TFNP}$ subclasses, each of which is defined with respect to some complete problems.

While oracle separations between $\mathsf{TFNP}$ subclasses have been well studied [BCE+95, Mor, BOM04, GHJ+24, JLRX24], the idea of raising $\mathsf{TFNP}$ subclasses to oracles and studying the result complexity classes in its own right is relatively new. [KKMP21] introduced $\mathsf{TFNP}^{\Sigma_i^{\mathsf{P}}}$ as a total

function analogue of PH. They also considered complexity classes like $\mathsf{PPP}^{\Sigma_i^P}$, where the input circuit $C$ in Definition 1.1 is allowed to have $\Sigma_i^P$ oracle gates.

While complexity classes like $\mathsf{PPP}^{\mathsf{NP}}$ are natural to define and easy to reason about, verifying a solution requires an NP oracle (assuming $\mathsf{NP} \neq \mathsf{coNP}$). In particular, we do not know how to efficiently verify when an NP oracle query is a NO instance. One natural attempt to get around this issue is to switch the NP oracle into a complete problem for some subclasses in TFNP such as PIGEON. Now that every oracle query has an efficiently verifiable solution, one can efficiently evaluate a $C^{\text{PIGEON}}$ circuit when given solutions to the oracle queries as auxiliary inputs. As such, these complexity classes like $\mathsf{PPP}^{\mathsf{PPP}}$ now also stay in TFNP!

In this work, we introduce TFNP subclasses which have oracle access to some TFNP problem. For example, $\mathsf{PPP}^{\mathsf{PPP}}$ would have as its complete problem $\text{PIGEON}^{\text{PIGEON}}$ which is the same as PIGEON except that the input circuit $C$ is allowed to have PIGEON oracle gates. Such a notion is somewhat more difficult to define and work with since unlike a $\Sigma_i^P$ oracle gate, which encodes a decision problem with exactly one output, a PIGEON instance may have an exponential number of possible outputs, which defines a relation rather than a function. The behavior of the PIGEON oracle gate is therefore underspecified. Hence, it is somewhat unclear exactly how one should think of the oracle gates unless the oracle problem is a search problem with unique solutions (like finding all prime factors of an integer). Informally, we will resolve this issue by quantifying over all possible functions consistent with the relation defined by the oracle problem.

Under the definition above, one can naturally define hierarchies of subclasses of TFNP, such as the PPA hierarchy defined as $\mathsf{PPA}^1 = \mathsf{PPA}, \mathsf{PPA}^i = \mathsf{PPA}^{\mathsf{PPA}^{i-1}}$ for $i > 0$, and ask about the complexity of these hierarchies. We will show that, under our definition, several classic TFNP subclasses are self-low. In other words, their corresponding hierarchies collapse to the first level.

### 1.1 Our Results

**Definitions.** We begin by defining $\text{A}^{\text{B}}$ when A is a TFNP circuit problem (Definition 3.1) and B is any TFNP problem (Definition 3.5, Definition 4.2). Informally, a circuit problem is one where the input consists of a circuit $C$ and some other input $a$, and an answer to the problem can be verified using black-box queries to $C$. $\text{A}^{\text{B}}$ takes as input a poly$(n)$ size circuit $C^{\text{B}}$ with oracle gates for B and some other input $a$, and outputs a solution $y, w_{1,1}, \ldots, w_{\text{poly}(n),\text{poly}(n)}$. Informally, one should view $y$ as a solution to A on input $(C^{\text{B}}, a)$. However, verifying $y$ as a solution requires evaluating $C^{\text{B}}$, which requires evaluating B oracle gates. This is where $w_{1,1}, \ldots, w_{\text{poly}(n),\text{poly}(n)}$ come in. We use $w_{i,j}$ as the solution to the $j^{\text{th}}$ B oracle gate query on the $i^{\text{th}}$ time the verifier for A makes a query to $C^{\text{B}}$. In some sense, the solution to $\text{A}^{\text{B}}$ on input $(C^{\text{B}}, a)$ has the same form as the solution to A but also includes the auxiliary information required to evaluate $C^{\text{B}}$ to verify a solution.

**Robustness of Definition.** We then go on to show some desirable properties of our definition, indicating that we have indeed arrived at the "correct" definition of $\text{A}^{\text{B}}$. Assume A is a TFNP circuit problem (Definition 3.1) and B is any TFNP problem, and $\mathsf{A}$ and $\mathsf{B}$ are the set of total search problems reducible to A and B respectively. By defining $\mathsf{A}^{\mathsf{B}}$ as the set of total search problems reducible to $\text{A}^{\text{B}}$ under many-to-one $\mathsf{FP}^{\text{B}}$ reductions, we observe that $\mathsf{A}^{\mathsf{B}}$ is in TFNP (Theorem 4.1) and that $\mathsf{A}^{\mathsf{B}}$ is robust to the choice of complete problem for $\mathsf{A}$ or $\mathsf{B}$ (Theorem 4.4, Theorem 4.7).

**Main Theorems.** Our main technical contributions are as follows.

**Theorem 5.4.** $\mathsf{PPA}^{\mathsf{PPA}} = \mathsf{PPA}$.

**Theorem 6.5.** $\mathsf{PLS}^{\mathsf{PLS}} = \mathsf{PLS}$.

**Theorem 7.2.** $\mathsf{LOSSY}^{\mathsf{LOSSY}} = \mathsf{LOSSY}$.

**Further Applications.** In Section 8, we show how to apply our new TFNP subclasses to study the complexity of well known problems. One of the consequences of the fact that PPA is self-low and that factoring is (likely) in PPA means that one can generally assume access to a factoring oracle when reducing a problem to a PPA-complete problem (assuming the generalized Riemann hypothesis). As an instantiation of this technique we show Theorem 8.5. Let Factor be the problem of finding a non-trivial prime factor of an integer (or declaring none exist) and let Weak-Bertrand be the problem of generating a prime between $2^n$ and $2^{32n}$ given $1^n$ as input. The following two theorems may provide a new way of attacking the longstanding open problem of pinpointing the complexity of Weak-Bertrand in TFNP.

**Theorem 8.5.** *If* Weak-Bertrand *is in* $\mathsf{PPA}^{\text{Factor}}$*, then the generalized Riemann hypothesis implies that* Weak-Bertrand *is in* PPA.

**Theorem 8.4.** *Under the generalized Riemann hypothesis,* Weak-Bertrand *is in* $\mathsf{LOSSY}^{\mathsf{PPA}}$, $\mathsf{LOSSY}^{\mathsf{PPP}}$, $\mathsf{PPADS}^{\mathsf{PPA}}$, *and* $\mathsf{PPADS}^{\mathsf{PPP}}$.

## 1.2 Related Work

The idea of taking a complete circuit problem for a TFNP subclass, letting the input circuit have oracle gates, and then studying the resulting complexity class was first introduced by [KKMP21]. They consider $\mathsf{PPP}^{\mathsf{NP}}$ for which the complete problem is finding a collision in a circuit $C^{\mathsf{NP}} : [2^n] \to [2^n - 1]$. They also define an algorithmic variant of the Sauer-Shelah lemma which they call Shattering, and show that it is in $\mathsf{PPP}^{\mathsf{NP}}$.

[Kor22] also considered TFNP problems assuming access to an oracle. In particular, they consider Lossy with access to an MCSP (minimum circuit size problem) oracle and showed that solving $\text{Lossy}^{\text{MCSP}}$ allows one to generate hard truth tables. This is closely related to our work, except MCSP is not a TFNP problem. [Kor22] also considered Lossy with a factoring oracle. They showed that if one can solve $\text{Lossy}^{\text{Factor}}$, then one can deterministically generate large primes, a longstanding open problem. We note, however, that [Kor22] merely assumes the existence of such oracles. They did not consider if the search problem associated to the oracle is in TFNP or what the complexity of solving $\text{Lossy}^{\text{Factor}}$ may be.

[LLR24] studied the TFNP class in the decision tree model coined as rwPHP(PLS), which are problems reducible to the retraction weak pigeonhole principle where the retraction function is in PLS. To a certain extent one could interpret this class as $\mathsf{LOSSY}^{\mathsf{PLS}}$ in our language, but in the decision tree model, or in the fully black-box setting. [LLR24] showed that this class captures the problem of proving certain restricted lower bounds for Resolution proofs.

Turing-closure is arguably the concept most closely related to self-lowness. We say that a TFNP subclass A is Turing closed if $\mathsf{FP}^{\mathsf{A}} = \mathsf{A}$, or equivalently, the existence of a Turing reduction to A implies the existence of a many-to-one reduction to A. Recall that we define $\mathsf{A}^{\mathsf{B}}$ as the set of problems reducible to $\text{A}^{\text{B}}$ under many-to-one $\mathsf{FP}^{\text{B}}$ reductions, where A and B are complete problems for A and B respectively. Proving self-lowness $\mathsf{A}^{\mathsf{A}} = \mathsf{A}$ implicitly requires A to be Turing

closed in the first place, since we are allowing $\mathsf{FP}^{\mathrm{A}}$ reductions. In other words, self-lowness is a stronger property than Turing-closure under our definition.

The Turing-closure of PLS, PPA, PPAD, and PPADS was shown in [BJ12]. LOSSY was shown to be Turing-closed in [LPT24]. On the other hand, [FGPR24] showed that PPP is not Turing-closed under black-box reductions.

### 1.3 Open questions

1. Are PPAD, PPADS, CLS, and UEOPL self-low?
2. The newly defined complexity class PLC (polynomial long choice) is meant to capture the combinatorial principle of the iterated pigeonhole principle [PYP22]. [PYP22] ask if $\mathsf{PLC} \subseteq \mathsf{FP}^{\mathsf{PPP}}$? We believe one should also ask more general question like is $\mathsf{PLC} \subseteq \mathsf{PPP}^{\mathsf{PPP}}$ or $\mathsf{PPP}^{\mathsf{PPP}} \subseteq \mathsf{PLC}$ or if the two classes are incomparable.

## 2 Preliminaries

### 2.1 TFNP

We begin by formally defining a search problem and TFNP search problems.

**Definition 2.1.** A search problem is a binary relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ where we say that $y$ is a solution to $x$ iff $(x, y) \in \mathcal{R}$.

**Definition 2.2** (TFNP)**.** A total NP search (TFNP) problem is a relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ such that the following properties hold.

- Polynomial: For all $(x, y) \in \mathcal{R}$, $|y| \leq \mathrm{poly}(|x|)$.
- Totality: For all inputs $x$, there is a solution $o$ such that $(x, o) \in \mathcal{R}$.
- FNP membership: There exists a $\mathrm{poly}(|x|, |o|)$ time algorithm $V$ such that $V(x, o) = 1$ if and only if $(x, o) \in \mathcal{R}$.

When dealing with TFNP problems, we are generally concerned with many-to-one reductions. One should think of these as reductions with a single oracle call.

**Definition 2.3.** Let $\mathcal{R}, \mathcal{Q}$ be TFNP problems. A many-to-one reduction from $\mathcal{R}$ to $\mathcal{Q}$ is defined as two polynomial time computable functions $f, g$ such that for all $x \in \{0,1\}^*, y \in \{0,1\}^*$, the following holds.

$$(x, g(y)) \in \mathcal{R} \impliedby (f(x), y) \in \mathcal{Q}$$

Alternatively, there is the notion of a Turing reduction, where one can make multiple oracle calls. Informally, we say that a class is Turing-closed if Turing reductions give us no more power than many-to-one reductions.

**Definition 2.4** ([FGPR24])**.** We say that a search problem $\mathcal{R}$ is Turing-closed if any problem which is polynomial time reducible to $\mathcal{R}$ via multiple calls to an oracle for a problem in $\mathcal{R}$ is also polynomial time reducible to $\mathcal{R}$ using a single call to an oracle for a problem in $\mathcal{R}$. We say a complexity class with a complete problem is Turing-closed if its complete problem is Turing closed.

Quite crucially for us, a variety of important TFNP subclasses are known to be Turing-closed.

**Lemma 2.5** ([BJ12])**.** PPA, PPAD, PPADS, *and* PLS *are Turing-closed.*

**Lemma 2.6** ([LPT24])**.** LOSSY *is Turing-closed.*

### 2.2 Some TFNP subclasses

Here we review some TFNP subclasses and give some informal intuition regarding their structures. All of these classes of interest will involve a polynomial size circuit implicitly encoding an exponential size object. The goal will be to find (an efficiently verifiable) structure which must exist in this exponential size object. We note that we will freely switch between a binary string or set ($\{0,1\}^n$) and its integer representation ($[2^n]$). We encourage the reader to not be overly concerned with this technical detail.

**Definition 2.7** (PPA and Bipartite-mod-2)**.** The problem Bipartite-mod-2 is defined as follows. Given a circuit $C : \{0,1\} \times \{0,1\}^n \to \mathsf{Set}_{\leq 2}(\{0,1\} \times \{0,1\}^n)$ (where $\mathsf{Set}_{\leq 2}(S)$ denotes some encoding of subsets of $S$ with size at most 2), representing a bipartite graph on the vertex set $(\{0\} \times \{0,1\}^n, \{1\} \times \{0,1\}^n)$ with $|C(00^n)| = 1$ find either of the following.

1. $x \neq 00^n$ such that $|C(x)| = 1$
2. $x, y$ such that $y \in C(x)$ but $x \notin C(y)$

PPA is defined as all TFNP problems which are many-to-one reducible to the problem Bipartite-mod-2.

The circuit for Bipartite-mod-2 should be viewed as implicitly encoding a bipartite graph on vertices $\{0,1\} \times \{0,1\}^n$. We think of all vertices in $0 \times \{0,1\}^n$ as being on the left of this graph and all vertices in $1 \times \{0,1\}^n$ as being on the right of this graph. $C(x)$ outputs a set of size at most 2 which is the set of vertices connected to $x$. We can ensure syntactically that edges on the vertices on the left are only connected to vertices on the right and vice versa by modifying the circuit $C$. We elide this minor technical detail and assume that the circuit $C$ satisfies this property. A solution to Bipartite-mod-2 is a vertex which does not have exactly 1 neighbor (a type 1 solution). Alternatively it is a witness that the circuit does not encode a graph since $y \in C(x)$ implies $(x, y)$ is an edge in the implicitly defined graph, which should imply that $(y, x)$ is also an edge in that graph and therefore that $x \in C(y)$ (a type 2 solution). To see that Bipartite-mod-2 is total assume that $C$ encodes a bipartite graph (otherwise, a type 2 solution to Bipartite-mod-2 exists), consider the sum of the degrees of all vertices $0 \times \{0,1\}^n$, call it $a$. Similarly, call the sum of the degrees of all nodes $1 \times \{0,1\}^n$ $b$. Notice $a = b$. If all vertices except $00^n$ have degree 0 mod 2, then $a = |C(00^n)| \neq 0 \pmod 2$, and $b = 0 \pmod 2$, which contradicts the fact that $a = b$. Therefore, there must be some $x \neq 00^n$ such that $|C(x)| = 1$.

**Definition 2.8** (PPAD and EndOfLine)**.** The problem EndOfLine is defined as follows. Given $S : \{0,1\}^n \to \{0,1\}^n, P : \{0,1\}^n \to \{0,1\}^n$ such that $S(0) \neq 0$ and $P(0) = 0$, output $x$ such that $P(S(x)) \neq x$ or $x \neq 0$ such that $S(P(x)) \neq x$. PPAD is defined as all TFNP problems which are many-to-one reducible to EndOfLine.

The circuit for EndOfLine should be viewed as specifying a directed graph. The input gives us a successor circuit $S$ and a predecessor circuit $P$. We say that the graph implicitly defined by $S, P$ has an edge from $x$ to $y$ if $S(x) = y$ and $P(y) = x$. Notice that there is no edge leading to 0 in any such graph since $P(0) = 0$. Therefore, there must be a node $x$ which has no outgoing edges, which implies $P(S(x)) \neq x$. This can be thought of as a sink of a line in the graph defined by $S, P$. Notice that EndOfLine also allows for a solution $x$ such that $S(P(x)) \neq x$. This should be interpreted as the beginning of a new line in the graph implicitly encoded by $S, P$ (one other than the one starting at 0), a node which has an edge out but no incoming edges.

**Definition 2.9** (PPADS and SinkOfLine)**.** The problem SinkOfLine is defined as follows. Given $S : \{0,1\}^n \to \{0,1\}^n, P : \{0,1\}^n \to \{0,1\}^n$ such that $S(0) \neq 0$ and $P(0) = 0$, output $x$ such that $P(S(x)) \neq x$. PPADS is defined as all TFNP problems which are many-to-one reducible to SinkOfLine.

SinkOfLine is almost the same as EndOfLine except that we only allow one type of solution: a sink in the graph implicitly defined by $S, P$. Beginnings of a new line are no longer solutions. One can also define the following useful PPADS-complete problem (under blackbox reductions) which we will use in Section 8.

**Definition 2.10.** The problem Injective-Pigeon is defined as follows. Given $C : [2^n] \to [2^n - 1], D : [2^n - 1] \to [2^n]$, output $x$ such that $D(C(x)) \neq x$.

**Definition 2.11** (PLS and Sink-of-DAG)**.** The Sink-of-DAG problem is defined as follows. Given a poly$(n)$ size circuits $S : [2^n] \to [2^n]$ and $V : [2^n] \to [2^n]$ such that $S(0) \neq 0$, find $v$ such that $S(v) \neq v$ and either $S(S(v)) = S(v)$ or $V(S(v)) \leq V(v)$. PLS is the set of all TFNP problems which are many-to-one reducible to Sink-of-DAG.

Sink-of-DAG should be viewed as encoding a gradient ascent problem. There are two circuits, a successor circuit and a value circuit. At every point $v$ in the space, we hope that the successor function $S$ leads us to a (different) point which has a higher value, $(V(S(v)) > V(v))$. A solution to Sink-of-DAG is a point such that this condition is violated ($S(v) \neq v$ but $V(S(v)) \leq V(v)$), or one which acts as a sink in the gradient ascent process ($S(v) \neq v$ but $S(S(v)) = S(v)$).

**Definition 2.12** (PWPP and Weak-Pigeon)**.** The Weak-Pigeon problem is defined as follows. Given a poly$(n)$ size circuits $C : \{0,1\}^n \to \{0,1\}^{n-1}$, output distinct $x_1, x_2 \in \{0,1\}^n$ such that $C(x_1) = C(x_2)$. PWPP is the set of all TFNP problems which are many-to-one reducible to Weak-Pigeon.

PWPP should be considered the algorithmic analogue of the weak pigeonhole principle. We know that a collision exists in $C$ since $C$ is compressing, Weak-Pigeon asks us to find a collision.

**Definition 2.13.** The problem $f(n)$-Lossy is defined as follows. Given $C : \{0,1\}^n \to \{0,1\}^{f(n)}$ and $D : \{0,1\}^n \to \{0,1\}^{f(n)}$, output $x \in \{0,1\}^n$ such that $D(C(x)) \neq x$. We refer to $(n/2)$-Lossy simply as Lossy. LOSSY is defined as all TFNP problems which are many-to-one reducible to Lossy.

[Kor22] defined Lossy (which they call Lossy Code), but did not define the complexity class LOSSY. We believe this is the correct and natural definition for LOSSY. One should view the inputs to $f(n)$-Lossy as consisting of a compressor circuit $C$ and a decompressor circuit $D$. The

goal is to find a string that is not compressible by this compression scheme. Such a string must exist since the compression scheme is lossy. The following lemma shows that the compression factor of $C$ and $D$ does not matter (up to a polynomial factor)

**Lemma 2.14** ([Kor22])**.** *$f(n)$-*Lossy *is many-one equivlent to* Lossy *for any efficiently computable $f(n) < n$ and $f(n) = \text{poly}(n)$.*

**Definition 2.15** (Factor)**.** Factor is defined as follows. Given an $n$ bit integer $x$, output $0^{n-1}$ if $x$ is prime. Otherwise, output $y \in \{0,1\}^{n-1}$ such that $y$ divides $x$.

Notice that an $n$ bit composite number has an $n-1$ bit non-trivial divisor. Therefore, the size of the solution to Factor on an $n$ bit input is bounded above by $n-1$. Furthermore, as was shown in [AKS04], testing if $x$ is prime can be done in polynomial time.

# 3 Definition

## 3.1 Oracle gates

We will be considering some TFNP problem A given access to an oracle for a TFNP problem B. However, this notion will only make sense (at least as we define it) when A is what we term a "circuit problem".

**Definition 3.1.** We say A is a circuit problem if the input to A is a $\text{poly}(n)$ size circuit $C : \{0,1\}^n \to \{0,1\}^{\text{poly}(n)}$ and possibly some other $\text{poly}(n)$ size input $a \in \{0,1\}^{\text{poly}(n)}$.

We note that the complete problems for all the major TFNP problems are in fact circuit problems (Section 2.2). There is a minor subtlety that some problems, like Sink-of-DAG, take as input more than one circuit $S, V$. In these cases, we treat the circuits as a single circuit $SV$, where $SV(x) = S(x)\|V(x)$. As an example, Sink-of-DAG is a circuit problem since it has as input circuits $S$ and $V$ (which we treat as a single circuit). We assume a canonical gate evaluation order over circuits. Since B is a TFNP problem, the length of any solution on input $x$ is bounded by some polynomial $p(|x|)$. We will insist that all B oracle gates $G$ in $C^{\text{B}}$ have the form $G : \{0,1\}^z \to \{0,1\}^{p(|z|)}$. This is to ensure that a valid solution always fits within the number of output wires of $G$.

**Example 3.2.** *To illustrate some of the concepts around* TFNP *classes with oracles, we will review concepts by instantiating them with the* Weak-Pigeon$^{\text{Factor}}$ *instance of Figure 1. For the remainder of this section, we refer to the circuit defined in Figure 1 as $T : \{0,1\}^6 \to \{0,1\}^5$.*

## 3.2 Evaluating oracle circuit with auxiliary oracle answers: $C^*$

We now reiterate why defining $\text{A}^{\text{B}}$ for TFNP problems A and B, is more challenging than defining $\text{A}^{\text{B}}$ when B is a decision problem, like SAT. When the oracle gates in $C^{\text{SAT}}$ are SAT gates, the behavior of the circuit on any input $x$ is well-defined since every input to an oracle gate has a unique output (0 or 1). However, when the oracle gates in $C^{\text{Pigeon}}$ are gates for a search problem (e.g., Pigeon), the behavior of the circuit $C^{\text{Pigeon}}$ on an input $x$ may not be well-defined, since the evaluation of the oracle gates could have many possible solutions. If the oracle gates of $C$ are for a problem with unique solutions (e.g., finding all prime factors of a number), this problem does

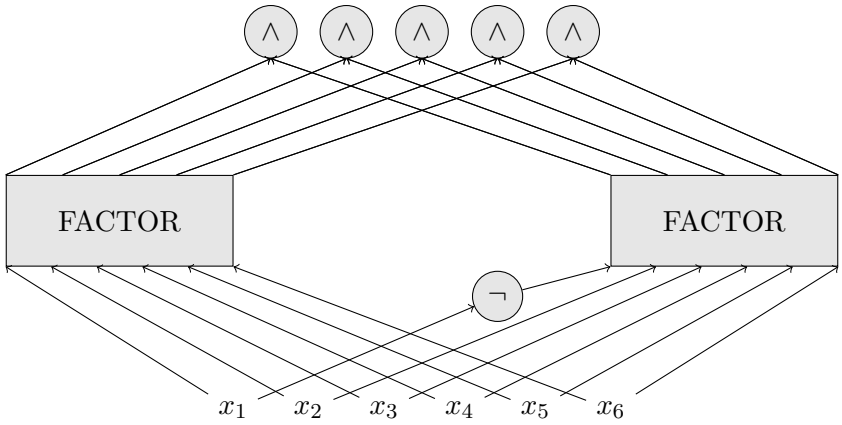


Figure 1: A WEAK-PIGEON$^{\text{FACTOR}}$ instance

not occur, but we want to work with full generality. Towards addressing this issue, we define $C^*$ which simulates $C^{\mathrm{B}}$ with auxiliary inputs that serve as solutions to its oracle gates.

**Definition 3.3.** Let $C^{\mathrm{O}} : \{0,1\}^n \to \{0,1\}^m$ be an oracle circuit where O is a TFNP problem, $C^{\mathrm{O}}$ contains $t$ oracle gates, and the $i^{\text{th}}$ oracle gate of $C^{\mathrm{O}}$ has an $s_i$-bit output. We now define

$$C^* : \{0,1\}^n \times \prod_{i=1}^{t} \{0,1\}^{s_i} \to \{0,1\}^m \ .$$

$C^*$ on input $(x, w_1, \ldots, w_t)$ simulates the evaluation of $C^{\mathrm{O}}$ on $x$. At the $i^{th}$ instance when $C^{\mathrm{O}}$ must evaluate an oracle gate, say on input $u$, we check if $w_i$ has a suffix that is a valid solution to O on input $u$. If it is not, then $C^*(x, w_1, \ldots, w_t) = \perp$ and the simulation terminates; otherwise, the simulation of $C^{\mathrm{O}}$ continues by using $w_i$ as the output of oracle gate $i$. If the simulation does not terminate prematurely and obtains a simulated $m$-bit output of $C^{\mathrm{O}}(x)$, $C^*(x, w_1, \ldots, w_t)$ returns that output.

$C^*(x, w_1, \ldots, w_t)$ is the evaluation of $C^{\mathrm{O}}$ on $x$ given that the output of oracle gate $i$ is $w_i$. In other words, $(x, w_1, \ldots, w_t)$ contains all the information necessary to lock in an evaluation of $C^{\mathrm{O}}$ on $x$.

We note that $w_i$ may not itself be a solution to the query made to oracle gate $i$ during the evaluation of $C^{\mathrm{O}}(x)$, but some suffix of $w_i$ is a valid solution. We do this because oracle gates have a fixed number of output wires but we still wish to allow for variable length solutions to queries to oracle gates. This is without loss of generality, as the circuit can determine for itself which suffix is a solution to the query it made to the oracle gate. Also for the rest of the presentation, when we say $w_i$ is a solution for some oracle query $u$, we mean that some suffix $w_i'$ where $w_i = r \| w_i'$ is a solution to $u$. Moreover, for some circuit $C_u$ defined on $\{0,1\}^{|w_i'|}$, we abuse the notation and define $C_u(w) := r \| C_u(w_i')$ for simplicity.

**Example 3.4** (Continuation of Example 3.2)**.** *We now provide some intuition for what $T^*$ does by probing it on a few values. We assume that the gates of $T$ are evaluated from left to right. In the following examples, we freely switch between integers and their binary representations. Consider the evaluation of $T^*(10, 2, 17)$. We begin by simulating $T$ on the input integer* 10*, which is* 001010

*in binary. This leads us to query* 001010 *at the left* FACTOR *gate. We see that* 2 *is indeed a factor of* 10*, so we continue the simulation of* $T$ *by assuming that the* LEFT *oracle gate outputs* 2 *(*00010 *in binary). The simulation then queries* 101010 *(42 in binary) at the right* FACTOR *gate. We see that* 17 *is not a factor of 42, so* $T^*(10, 2, 17) = \perp$.

*Consider the evaluation of* $T^*(10, 2, 21)$. *Again, we begin by simulating* $T$ *on the input integer* 10*, which is* 001010 *in binary. This leads us to query* 001010 *at the left* FACTOR *gate. We see that* 2 *is indeed a factor of* 10*, so we continue the simulation of* $T$ *by assuming that the* LEFT *oracle gate output* 2 *(*00010 *in binary). The simulation next queries* 101010 *(42 in binary) at the right* FACTOR *gate. We see that* 21 *is indeed a factor of* 42*, so the right* FACTOR *gate outputs* 21 *(*10101 *in binary). Therefore,* $T^*(10, 2, 21) = 00010 \wedge 10101 = 00000$.

### 3.3 The full definition

We are now ready to define what a solution to $\mathrm{A}^{\mathrm{B}}$ looks like.

**Definition 3.5.** Let A and B be TFNP problems and A be a circuit problem with a black-box verifier $V : \{0,1\}^n \times \{0,1\}^{\mathrm{poly}(n)} \to \{0,1\}$, one which only queries the circuit input to A in a black-box manner. The input to $\mathrm{A}^{\mathrm{B}}$ is defined the same as A except the circuit input $C^{\mathrm{B}}$ of A contains $t$ B oracle gates. The verifier for $\mathrm{A}^{\mathrm{B}}$, $V' : \{0,1\}^n \times \{0,1\}^{\mathrm{poly}(n)} \to \{0,1\}$ is defined as follows. $V'$ takes as input $(C^{\mathrm{B}}, a), (y, w_{1,1}, \ldots, w_{\mathrm{poly}(n),t})$. $V'$ simulates the computation of $V((C^{\mathrm{B}}, a), y)$ and on the $i^{th}$ evaluation of $C^{\mathrm{B}}$, say on input $x$, the black-box verifier $V'$ receives $C^*(x, w_{i,1}, w_{i,2}, \ldots, w_{i,t})$. If at any point, $C^*(x, w_{i,1}, w_{i,2}, \ldots, w_{i,t}) = \perp$ or $V'$ finds the sub-witnesses $w_{i,j}$ to not be *internally consistent* (which we define below), $V'$ outputs 0. Otherwise, $V'$ outputs the result of $V$ after the simulation of $V$ terminates.

We say that the $w_{1,1}, \ldots, w_{\mathrm{poly}(n),t}$ are not *internally consistent* (with respect to the simulation) if in the execution of $V'$, there exist two oracle queries made on the same input $u$ and the simulation uses answer $w_{a,b}$ to the first oracle gate query and $w_{a',b'}$ as the answer to the second oracle gate query and $w_{a,b} \neq w_{a',b'}$.

**Internal Consistency** A crucial feature in Definition 3.5 above is the *internal consistency* requirement. Informally, it enforces that the oracle B behaves consistently on the inputs observed by $V'$. Ultimately we wish $C^{\mathrm{B}}$ to define a function (at least from the perspective of the verifier $V'$). This is because the proof of totality of many classic TFNP subclasses relies on the circuit being a function! For example, it only makes sense to instantiate the pigeonhole principle in PIGEON if the underlying circuit defines a function. While the definition of $C^*$ allows the verifier $V'$ to lock in one evaluation of $C^{\mathrm{B}}$, it does not guarantee that $C^{\mathrm{B}}$ behaves consistently across multiple evaluations done by $V'$. By requiring the oracle B to behave consistently across multiple evaluations in $V'$, we also obtain consistent behaviors of $C^{\mathrm{B}}$ as desired. One may of course consider the possibility of enforcing consistency only on $C^{\mathrm{B}}$ but not on B. But we will see in Section 4 that our definition enjoys many other nice properties, indicating that it is more likely to be the "correct" definition.

*Remark* 3.6. If we only enforce the consistency only on $C^{\mathrm{B}}$ but not on B, reduction algorithm might break in the following scenario: Suppose that we are reducing $A^{\mathrm{B}}$ to $C^{\mathrm{B}}$ in a black-box manner. We construct a circuit for $C^{\mathrm{B}}$ consisting of $A^{\mathrm{B}}$-gates. Since we only enforce consistency on $C^{\mathrm{B}}$, a solution for $C^{\mathrm{B}}$ could lead to inconsistent evaluations of the $A^{\mathrm{B}}$-gates on the same input, which could cause the reduction to break.

In particular, a solution to $\mathrm{A}^{\mathrm{B}}$ is a solution to A along with all the answers to the B oracle gate queries that are made when verifying such a solution. We allow any input/output of the B gates as long as it is actually a solution to B and the behavior is consistent across multiple evaluations of B. In some sense, we are quantifying over all instantiations (i.e. all functions consistent with the relation defined by B) of a B oracle. Under Definition 3.5, a reduction from some problem C to $\mathrm{A}^{\mathrm{B}}$ should work for all instantiations of the oracle gates for B. An alternative perspective is that we are delegating the work of fixing the B oracle to the solution $y, w_{1,1}, \ldots, w_{\mathrm{poly}(n),t}$, particularly the $w_{1,1}, \ldots, w_{\mathrm{poly}(n),t}$.

**Example 3.7** (Continuation of Example 3.4)**.** *Using Definition 3.5, we see that a solution to* $\textsc{Weak-Pigeon}^{\textsc{Factor}}$ *on input* $T$ *would be distinct* $x_1, x_2$ *and internally consistent* $w_1, w_2, w_3, w_4$ *such that* $T^*(x_1, w_1, w_2) \neq \perp$ *and* $T^*(x_1, w_1, w_2) = T^*(x_2, w_3, w_4)$*. We will now try to find solutions to our* $\textsc{Weak-Pigeon}^{\textsc{Factor}}$ *instance* $T$*. One seemingly obvious solution is* $(x_1, x_2) = (0, 32), (w_1, w_2) = (0, 2), (w_3, w_4) = (8, 0)$*. This appears to be a solution since* $T^*(0, 0, 2) = 0$ *and* $T^*(32, 8, 0) = 0$*. However, observe that this solution is not internally consistent. In particular, when evaluating* $T^*(0, 0, 2)$*, we assume that* Factor *on input* 32 *evaluates to 2, but when evaluating* $T^*(32, 8, 0)$*, we assume that* Factor *on input* 32 *evaluates to* 8*.*

*An actual solution for* $\textsc{Weak-Pigeon}^{\textsc{Factor}}$ *on instance* $T$ *(which we leave to the reader to confirm) is* $(x_1, x_2) = (0, 32), (w_1, w_2) = (0, 2), (w_3, w_4) = (2, 0)$*.*

### 3.4 Another helpful evaluation: $C_*$

We now define an alternative way to evaluate $C^{\mathrm{O}}$ on $x$ that is significantly different from $C^*$. While $C^*$ may be easier to reason about, $C_*$ will be useful in enforcing internal consistency of our solutions.

**Definition 3.8.** Let $C^{\mathrm{O}} : \{0,1\}^n \to \{0,1\}^m$ be an oracle circuit where O is a TFNP problem, $C^{\mathrm{O}}$ contains $t$ oracle gates, and the $i^{\text{th}}$ oracle gate of $C^{\mathrm{O}}$ has an $s_i$-bit output. We now define

$$C_* : \{0,1\}^n \times \prod_{i=1}^{t} \{0,1\}^{s_i} \to \{0,1\}^m \ .$$

We maintain an ongoing set $M$, initially empty. $C_*$ on input $(x, w_1, \ldots, w_t)$ simulate the evaluation of $C^{\mathrm{O}}$ on $x$. At the $i^{th}$ instance when $C^{\mathrm{O}}$ must evaluate an oracle gate $G_i$, say on input $u$, if 0 is a valid solution to $G_i$ on $u$, we continue simulating $C^{\mathrm{O}}$ using 0 as the output of oracle gate $G_i$. Otherwise, we find the minimum $j$ such that $w_j$ is a solution for O on input $u$ and update $M \leftarrow M \cup \{j\}$. We refer to this as the $i^{\text{th}}$ witness used to evaluate $C_*(x, w_1, \ldots, w_t)$. If no such $j$ exists when evaluating oracle gate $i$, let $m$ be the smallest index such that $m \notin M$ and we output $(m, u)$. We refer to this as an error output. Otherwise, we continue simulating $C^{\mathrm{O}}$ by using $w_j$ as the output of oracle gate $i$. If the simulation does not terminate prematurely and obtains a simulated $m$-bit output of $C^{\mathrm{O}}(x)$, $C_*(x, w_1, \ldots, w_t)$ returns that output. We refer to this as a result output. We say $w_j$ was used in the evaluation of $C_*(x, w_1, \ldots, w_t)$ if $j \in M$ when the procedure for $C_*$ terminates.

The key difference between $C^*$ and $C_*$ is that instead of using the subsequent $w_i$ as the solution to an oracle gate query in the simulation of $C^{\mathrm{O}}$ as we did in $C^*$, we find the first $w_i$ which is sufficient to answer the oracle gate query and use that in the simulation of $C^{\mathrm{O}}$. We refer to this as

evaluation by first witnesses since we use the first solution possible whenever evaluating an oracle gate. We note that such evaluation naturally enforces internal consistency.

**Example 3.9** (Continuation of Example 3.7)**.** *To understand how $T_*$ is evaluated, consider $T_*(10, 2, 21)$. We begin by simulating $M$ on the input 10, or 001010 in binary. We query the left* Factor *gate on 10 and find that 2 is indeed a factor of 10 (and in particular, the first value which is a factor among $2, 21$), so we assume that the left* Factor *gate outputs 00010. We next evaluate the right* Factor *gate on 42 and find that 2 is a factor of 42 (and in particular, the first value which is a factor among $2, 21$), so we continue the evaluation assuming that the right* Factor *gate outputs 2. This leads the simulation of $C$ to output $00010 \wedge 00010 = 00010$, or 2. Therefore $T_*(10, 2, 21) = 2$. The observant reader will note that this is a different result from is the same result as in Section 3.2, where $T^*(10, 2, 21) = 0$.*

*Finally, let us consider an example where $T_*$ outputs a err result. Consider $T_*(14, 7, 9)$. We query the left* Factor *gate on 14 and find that 7 is indeed a factor of 14 (and in particular, the first value which is a factor among $7, 9$), so we assume that the left* Factor *gate outputs 00111. We also add 1 to our set $M$. We next evaluate the right* Factor *gate on 46 and find that neither 7 nor 9 are factors of 46. We therefore terminate and $T_*(14, 7, 9) = (2, 46)$, indicating that $w_2$ is the next unused witnesses.*

# 4 Robustness of Definition

Having defined $\mathrm{A}^{\mathrm{B}}$ for TFNP problems, we critique our definitions and show that they have several desirable properties. We first show that $\mathrm{A}^{\mathrm{B}}$ is in TFNP.

**Theorem 4.1.** *If* A *and* B *are* TFNP *problems and* A *is a circuit problem with a black-box verifier, then $\mathrm{A}^{\mathrm{B}} \in$ TFNP.*

*Proof.* The efficiency of the verifier for $\mathrm{A}^{\mathrm{B}}$ and the fact that $\mathrm{A}^{\mathrm{B}}$ has polynomially bounded solutions are inherited from the efficiency of the verifier for A. To show totality, let us consider a problem $\mathrm{B}'$ defined as $\{(x, y) : y$ is the lexicographically first solution to B on input $x\}$. Notice then that $\mathrm{A}^{\mathrm{B}'}$ is total since A is total. But of course, any solution to $\mathrm{A}^{\mathrm{B}'}$ is a solution to $\mathrm{A}^{\mathrm{B}}$. Therefore, $\mathrm{A}^{\mathrm{B}}$ is total. □

**Definition 4.2.** Let $\mathsf{A}, \mathsf{B} \in$ TFNP, let A and B be canonical complete problems for $\mathsf{A}$ and $\mathsf{B}$ respectively, and let A be a circuit problem with a black-box verifier. $\mathsf{A}^{\mathsf{B}}$ is then defined as all problems which are many-to-one reducible to $\mathrm{A}^{\mathrm{B}}$ under $\mathsf{FP}^{\mathsf{B}}$ reductions.

**Observation 4.3.** *Let $\mathsf{A}, \mathsf{B} \in$ TFNP, let* A *and* B *be canonical complete problems for $\mathsf{A}$ and $\mathsf{B}$ respectively, and let* A *be a circuit problem with a black-box verifier. $\mathsf{A}^{\mathsf{B}} \subseteq$ TFNP.*

To justify our definition, we now show that our choice for the complete problem for $\mathsf{A}$ does not matter as long as there exists a black-box reduction to/from that problem from/to the canonical complete problem for $\mathsf{A}$. We note that Theorem 4.4 relies crucially on the fact that solutions to $\mathrm{A}^{\mathrm{B}}$ are internally consistent.

**Theorem 4.4.** *Let $\mathrm{A}_1, \mathrm{A}_2$ and* B *be* TFNP *problems and $\mathrm{A}_1, \mathrm{A}_2$ are circuit problems. If $\mathrm{A}_1$ is black-box many-one reducible to $\mathrm{A}_2$, then ${\mathrm{A}_1}^{\mathrm{B}}$ is black-box many-one reducible to ${\mathrm{A}_2}^{\mathrm{B}}$ under $\mathsf{FP}^{\mathrm{B}}$ reduction.*

*Proof.* Let $(f, g)$ be a pair of polynomial-time reduction algorithms such that on an $\mathrm{A}_1$ instance $C_1$, $f(C_1) = C_2$ generates an $\mathrm{A}_2$ instance $C_2$. And given solution $\pi_2$ for $C_2$, $g(C_2, \pi_2) = \pi_1$ outputs a solution $\pi_1$ for $C_1$. In particular, we emphasize that $(f, g)$ are black-box reduction algorithms. I.e., they evaluate $C_1$ on polynomially many inputs and construct $C_2$ as a circuit with $C_1$-gates.

We define a pair of $\mathsf{FP}^{\mathrm{B}}$ reduction algorithms $(f_1^{\mathrm{B}}, g_1^{\mathrm{B}})$. In particular, the reduction algorithms maintain a polynomial-sized table $T$ containing query-answer pairs to B.

On an $A_1^{\mathrm{B}}$ instance $C_1^{\mathrm{B}}$, $f_1^{\mathrm{B}}$ starts by instantiating an empty lookup table $T$. Next, it simulates $f$: whenever $f$ would evaluate $C_1^{\mathrm{B}}$ and hence query the oracle $B$ on some input $x$, it checks if $T$ contains the query-answer pair. If yes, it returns the answer stored in $T$. Otherwise, it uses its B oracle to obtain a query-answer pair, stores it in $T$ and returns the answer. At the end of its simulation of $f$, it constructs a circuit $C'$ (with $C_1^{\mathrm{B}}$-gates). It further modifies $C'$ as follows: the table $T$ is hardwired into $C'$ and any B gate in $C'$ is modified to always look for an answer from $T$ if available. The modified circuit $C_2^{\mathrm{B}}$ would be the output of $f_1^{\mathrm{B}}$.

Given a solution $(\pi_2, w_2)$ to $C_2^{\mathrm{B}}$, $g_1^{\mathrm{B}}$ starts by simulating the verifier for $\mathrm{A}_2^{\mathrm{B}}$ and adding the query-answer pairs from $w_2$ to the table $T$. Note that due to the additional modification in $C_2^{\mathrm{B}}$, there will be no inconsistency between $w_2$ and $T$. Next, it simulates $g$ and similarly maintains the table $T$ in the same manner as $f_1^{\mathrm{B}}$. At the end of the simulation, it outputs $\pi_1$ and oracle answers $w_1$ for verifying the solution.

Internal consistency follows from the use of table $T$ and it remains to show the correctness of the reduction.

Consider any function $b$ that is a restriction of B and is consistent with $T$. Since $b$ is a function, we may view $C_1^b$ as a vanilla circuit by hardwiring the truth table of $b$. Let $C_2^b := f(C_1^b)$. Since $f_1^{\mathrm{B}}$ simulates $f$ and $b$ is consistent with $T$, $C_2^b$ is the same as $C_2^{\mathrm{B}}$ with the oracle gates switched and hence $\pi_2$ is a solution to $C_2^b$. Since $g_1^{\mathrm{B}}$ simulates $g$ and by correctness of $(f, g)$, $\pi_1$ is a solution to $C_1^b$. Therefore, $\pi_1$ with the associated oracle answers $w_1$ must be a solution for $C_1^{\mathrm{B}}$ since $C_1^b$ and $C_1^{\mathrm{B}}$ have the same behavior with respect to the verifier. □

**Corollary 4.5.** *Let* $\mathrm{A}_1, \mathrm{A}_2$, B, *and* C *be* TFNP *problems. If all the following hold, then* $\mathrm{A}_1^{\mathrm{B}}$ *reduces to* C *under many-one reductions.*

1. $\mathrm{A}_1$ *is black-box many-one reducible to* $\mathrm{A}_2$.
2. $\mathrm{A}_2^{\mathrm{B}}$ *has a Turing reduction to* C.
3. B *has a Turing reduction to* C
4. C *is Turing-closed.*

*Proof.* By Theorem 4.4, $\mathrm{A}_1^{\mathrm{B}}$ reduces to $\mathrm{A}_2^{\mathrm{B}}$ under $\mathsf{FP}^{\mathrm{B}}$ reductions. Since B has a Turing reduction to C, $\mathrm{A}_1^{\mathrm{B}}$ reduces to ${\mathrm{A}_2}^{\mathrm{B}}$ under $\mathsf{FP}^{\mathrm{C}}$ reductions. This combined with the fact that $\mathrm{A}_2^{\mathrm{B}}$ has a Turing reduction to C implies $\mathrm{A}_1^{\mathrm{B}}$ reduces to C under $\mathsf{FP}^{\mathrm{C}}$ reductions. Since C is Turing-closed, this implies that $\mathrm{A}_1^{\mathrm{B}}$ reduces to C under many-one reductions. □

**Observation 4.6.** *We note that* $\mathsf{FP}^{\mathrm{B}}$ *reductions are allowed in Definition 4.2. As* PPP *is not believed to be Turing-closed [FGPR24], i.e.* $\mathsf{FP}^{\mathsf{PPP}} \neq \mathsf{PPP}$*, it indicates that* PPP *is likely not self-low.*

We now show that our choice for the complete problem for B also does not matter as long as there exists a reduction to/from that problem from/to the canonical complete problem for B.

**Theorem 4.7.** *Let* $\mathrm{B}_1, \mathrm{B}_2$ *and* A *be* TFNP *problems. If* $\mathrm{B}_1$ *is many-one reducible to* $\mathrm{B}_2$*, then* $\mathrm{A}^{\mathrm{B}_1}$ *is many-one reducible to* $\mathrm{A}^{\mathrm{B}_2}$*.*

*Proof.* Let $(f, g)$ be the many-to-one reduction from $\mathrm{B}_1$ to $\mathrm{B}_2$. We now show a reduction from $\mathrm{A}^{\mathrm{B}_1}$ to $\mathrm{A}^{\mathrm{B}_2}$. Let $C^{\mathrm{B}_1}$ be the input to $\mathrm{A}^{\mathrm{B}_1}$. Let $G$ be a circuit with a $\mathrm{B}_2$ oracle gate defined as follows. $G$ on input $x$, computes $f(x)$, feeds $f(x)$ into a $\mathrm{B}_2$-gate, and applies $g$ to the output of its $\mathrm{B}_2$-gate.

On an input $C_1^{\mathrm{B}_1}$, the reduction from $\mathrm{A}^{\mathrm{B}_1}$ to $\mathrm{A}^{\mathrm{B}_2}$ simply replaces each $\mathrm{B}_1$ gate with $G$ and outputs the resulting circuit $C_2^{\mathrm{B}_2}$. Given $(\pi, w_1, \ldots, w_t)$ as a solution for $C_2^{\mathrm{B}_2}$, the reduction outputs $(\pi, g(w_1), \ldots, g(w_t))$ as a solution.

By the correctness of the reduction $(f, g)$, $g(w_i)$ are proper solutions to the $\mathrm{B}_1$ oracle queries. Furthermore, by our construction of $C_2^{\mathrm{B}_2}$, any evaluation on $C_1^{\mathrm{B}_1}$ assisted by $g(w_i)$ would be exactly the same as evaluation on $C_2^{\mathrm{B}_2}$ assisted by $w_i$. As such, $(\pi, g(w_1), \ldots, g(w_t))$ is a valid solution to $C_1^{\mathrm{B}_1}$.

It remains to verify that internal consistency is preserved. Assume towards contradiction that $g(w_i) \neq g(w_j)$ are solutions to the same $\mathrm{B}_1$ query $x$ and hence $w_i \neq w_j$. However, $w_i$ and $w_j$ are solutions to the same $\mathrm{B}_1$ query $f(x)$ in $C_2^{\mathrm{B}_2}$. This contradicts the internal consistency of the solution $(\pi, w_1, \ldots, w_t)$.

□

# 5 PPA Self-lowness

We show that PPA is self-low. To do so, we make use of the following PPA-complete problem.

**Definition 5.1.** The problem Lonely is defined as follows. The input is a circuit $C : \{0,1\}^n \to \{0,1\}^n$. If $C(0) \neq 0$ (0 is not unpaired) output anything. Otherwise, find $w \neq 0$ such that either $C(w) = w$ (a type 1 solution) or $C(C(w)) \neq w$ (a type 2 solution) and output $w$.

**Definition 5.2.** The problem Lonely+ is defined as follows. Given a circuit $C : \{0,1\}^n \to \{0,1\}^n$, we say $a, b \in \{0,1\}^n$ are matched if $C(a) = b$ and $C(b) = a$. The input is $C$ and $u \in \{0,1\}^n$. If $C(u) \neq u$ ($u$ is not unpaired) output $0^n$. Otherwise, find $w \neq u$ such that either $C(w) = w$ or $C(C(w)) \neq w$ and output $w$.

**Lemma 5.3.** $\textsc{Lonely}^{\textsc{Lonely}}$ *reduces to* Lonely+ *under black-box Turing reductions..*

*Proof.* We show the reduction from $\textsc{Lonely}^{\textsc{Lonely}}$ to Lonely+. Let $C : \{0,1\}^n \to \{0,1\}^n$ be the input circuit to the reduction. We assume that $C$ has $t$ oracle gates, where the $i^{\text{th}}$ oracle gate has an output of size $s_i$.

We now define

$$C' : \{0,1\}^n \times \prod_{i=1}^{t} \{0,1\}^{s_i} \times \prod_{i=1}^{t} \{0,1\}^{s_i} \times \{0,1\} \to \{0,1\}^n \times \prod_{i=1}^{t} \{0,1\}^{s_i} \times \prod_{i=1}^{t} \{0,1\}^{s_i} \times \{0,1\} \ .$$

The reduction first computes $a_1, \ldots, a_t$ such that $a_1, \ldots, a_t$ are consistent for $C^*(0, a_1, \ldots, a_t)$ and $C^*(0, a_1, \ldots, a_t) \neq \perp$ by calling its Lonely+ oracle. If $C^*(0, a_1, \ldots, a_t) \neq 0$, the reduction outputs $(0, a_1, \ldots, a_t)$.

We define $C'(0, a_1, \ldots, a_t, 0, \ldots, 0, 0) = (0, a_1, \ldots, a_t, 0, \ldots, 0, 0)$. $C'$ on any other input $(x_1, w_1, \ldots, w_{2t}, b)$ behaves as follows: Let $x_2 = C^*(x_1, w_1, \ldots, w_t)$ and let $x_3 = C^*(x_2, w_{t+1}, \ldots, w_{2t})$. If any of the bad events:

- $x_1 = 0$;
- $x_2 = \perp$;
- $x_3 = \perp$;
- Internal consistency is violated with respect to the three evaluations $C^*(0, a_1, \ldots, a_t) = 0$, $C^*(x_1, w_1, \ldots, w_t) = x_2$, $C^*(x_2, w_{t+1}, \ldots, w_{2t}) = x_3$. I.e. different solutions are used for the same oracle query across the three evaluations,

occurs, $C'$ outputs $(x_1, w_1, \ldots, w_{2t}, b \oplus 1)$ and we call this type 1 output. Otherwise, $C'$ outputs $(x_2, w_{t+1}, \ldots, w_{2t}, w_1, \ldots, w_t, b)$ and we call this type 2 output.

The reduction then calls its LONELY+ oracle on $C'$, $u = (0, a_1, \ldots, a_t, 0, \ldots, 0, 0)$, gets back an answer $(v_1, w_1, \ldots, w_{2t}, b)$. Let $v_2 = C^*(v_1, w_1, \ldots, w_t)$ and $v_3 = C^*(v_2, w_{t+1}, \ldots, w_{2t})$. If $v_1 = v_2$, the reduction outputs $v_1, w_1, \ldots, w_t$. If $v_2 = 0$, the reduction outputs $v_1, w_1, \ldots, w_t, a_1, \ldots, a_t$. If $v_1 \neq v_3$, the reduction outputs $v_1, w_1, \ldots, w_{2t}$.

The reduction is clearly many-to-one and runs in polynomial time. To see correctness, suppose $v = (v_1, w_1, \ldots, w_{2t}, b)$ is a solution to $C'$. We start by noting that if $v$ falls into any of the bad events, $C'(v) \neq v$ and $C'(C'(v)) = v$ by construction, and could not be a solution. In other words, we have $v_1 \neq 0$, $v_2 = C^*(v_1, w_1, \ldots, w_t)$, $v_3 = C^*(v_2, w_{t+1}, \ldots, w_{2t})$ and internal consistency satisfied.

We consider the following scenarios:

1. $0 \neq v_1 = v_2$. In this case, $v_1, w_1, \ldots, w_t$ is a desired solution as $C^*(v_1, w_1, \ldots, w_t) = v_1$. We further note that $C'(v) = v$ must fall in this case, and we only need to consider $C'(C'(v)) \neq v$ for the rest of the cases.
2. $v_2 = 0$. We have $v_2 = C^*(v_1, w_1, \ldots, w_t) = 0$ and $v_1 \neq 0 = C^*(0, a_1, \ldots, a_t)$. Hence, $v_1, w_1, \ldots, w_t, a_1, \ldots, a_t$ is a desired solution.
3. $v_1 \neq v_3$. In this case, $v_1, w_1, \ldots, w_{2t}$ is a desired solution. One can verify that $C^*(C^*(v_1, w_1, \ldots, w_t), w_{t+1}, \ldots, w_{2t}) = v_3 \neq v_1$.
4. Finally we show that the remaining scenario where $v_2 \neq 0$, $v_1 = v_3$ and $C'(C'(v)) \neq v$ is impossible. In particular, if $v_1 = v_3$, then the evaluation $C'(v_2, w_{t+1}, \ldots, w_{2t}, w_1, \ldots, w_t)$ would not fall into bad events, since we know that $C^*(v_3, w_1, \ldots, w_t)$ is valid and internally consistent. As such, $C'(v_2, w_{t+1}, \ldots, w_{2t}, w_1, \ldots, w_t) = (v_3, w_1, \ldots, w_{2t}) = (v_1, w_1, \ldots, w_{2t})$. This contradicts that $C'(C'(v)) \neq v$.

□

**Theorem 5.4.** $\mathsf{PPA}^{\mathsf{PPA}} = \mathsf{PPA}$.

*Proof.* BIPARTITE-MOD-2 trivially reduces to BIPARTITE-MOD-2$^{\text{BIPARTITE-MOD-2}}$.

We now show the other direction. Note that all the following conditions are satisfied.

1. BIPARTITE-MOD-2 has a black-box many-one reduction to LONELY.
2. LONELY$^{\text{LONELY}}$ has a Turing reduction to LONELY+ by Lemma 5.3.

3. LONELY is many-one reducible to LONELY+.

4. LONELY+ is Turing-closed since PPA is Turing-closed.

Therefore, Corollary 4.5 tells us BIPARTITE-MOD-$2^{\text{LONELY}}$ has a many-one reduction to LONELY+, which has a many-one reduction to BIPARTITE-MOD-2. Finally, Theorem 4.7 tells us BIPARTITE-MOD-$2^{\text{BIPARTITE-MOD-2}}$ reduces to BIPARTITE-MOD-$2^{\text{LONELY}}$. Chaining these reduction lets us conclude BIPARTITE-MOD-$2^{\text{BIPARTITE-MOD-2}}$ reduces to BIPARTITE-MOD-2, as desired. □

## 6 PLS Self-lowness

We now show that PLS is self-low. To do so, we work with the PLS-complete problems ITER and ITER2. We observe (without proof) that ITER is PLS-complete under black-box reductions.

**Definition 6.1.** The problem ITER is defined as follows. The input is $S : [2^n] \to [2^n]$. If $S(0) = 0$, output 0. Otherwise, output $x$ s.t. $S(x) > x$ and $S(S(x)) \leq S(x)$.

**Definition 6.2.** The problem ITER2 is defined as follows. The input is $S : [2^n] \to [2^n]$. If $S(0) = 0$, output 0. Otherwise, output any of the following solutions.

1. $x$ s.t. $S(x) < x$,

2. $x$ s.t. $S(x) > x$ and $S(S(x)) \leq S(x)$.

**Lemma 6.3.** ITER2 *is* PLS*-complete.*

*Proof.* We first reduce ITER to ITER2. Given an instance $S$ of ITER, we let $S'(x)$ be $x$ if $S(x) < x$ and $S(x)$ otherwise. We feed $S'$ to our ITER2 to get back a solution $y$ which we output. Notice that the solution we get back must be a type 2 solution, $y$ s.t. $S'(y) > y$ and $S'(S'(y)) \leq S'(y)$. If $S'(y) > y$, then $S(y) = S'(y)$. Therefore, since $S'(S'(y)) \leq S'(y)$, $S'(S(y)) \leq S(y)$. This implies that $S(S(y)) \leq S(y)$, since by construction $S' \geq S$ for all inputs. Therefore, $y$ is a type 2 solution to ITER.

We now reduce ITER2 to ITER. Given an instance $S$ of ITER2, the reduction calls its ITER oracle on $S$ to get back an answer $y$ which it outputs. Since $y$ is the output of the oracle call, $S(y) > y, S(S(y)) \leq S(y)$. Therefore, $y$ is a type 2 solution to ITER2. □

**Lemma 6.4.** ITER$^{\text{ITER2}}$ *reduces to* ITER *under black-box Turing reductions.*

*Proof.* We show the reduction from ITER$^{\text{ITER2}}$ to ITER. Let $S$ be the input to the reduction. We assume that $S$ has $t$ oracle gates, where the $i^{\text{th}}$ oracle gate has an output of size $s_i$. Let $S^2$ be the circuit which simply composes $S$ with itself. Note that $S^2$ has $2t$ oracle gates.

The reduction first computes $a_1, \ldots, a_t$ such that $a_1, \ldots, a_t$ are consistent for $S^*(0, a_1, \ldots, a_t)$ and $S^*(0, a_1, \ldots, a_t) \neq \perp$ by calling its ITER oracle. If $S^*(0, a_1, \ldots, a_t) = 0$, the reduction outputs $(0, a_1, \ldots, a_t)$.

The reduction then constructs

$$S' : \{0,1\}^n \times \prod_{j=1}^{t} \{0,1\}^{s_j} \times \prod_{j=1}^{t} \{0,1\}^{s_j} \to \{0,1\}^n \times \prod_{j=1}^{t} \{0,1\}^{s_j} \times \prod_{j=1}^{t} \{0,1\}^{s_j} .$$

Let $y = S^*(x, w_1, \ldots, w_t)$ and if $y$ is a result output, let $z = S^*(y, w_{t+1}, \ldots, w_{2t})$. We now define $S'$ whose behavior is split into four cases.

1. If $y = \perp$ or internal consistency is violated in $S^*(x, w_1, \ldots, w_t)$, output $(x, w_1, \ldots, w_{2t})$.

2. If $y \neq \perp$, the evaluation of $S^*(x, w_1, \ldots, w_t)$ is internally consistent, and $y \leq x$, output $(x, w_1, \ldots, w_{2t})$.

3. Consider when $y \neq \perp$, the evaluation of $S^*(x, w_1, \ldots, w_t)$ is internally consistent, $y > x$, and one of the following occurs: $z = \perp$ or the evaluation of $S^{2^*}(x, w_1, \ldots, w_{2t})$ violates internal consistency. Let $m$ be the smallest index in $\{t+1, \ldots, 2t\}$ such that $w_m$ is not a solution to oracle gate query $m$ in the evaluation of $S^{2^*}(x, w_1, \ldots, w_{2t})$ or $w_m$ violates internal consistency in the evaluation of $S^{2^*}(x, w_1, \ldots, w_{2t})$ becuase gate query $m$ is the same as gate query $i$ for some $i < m$ and $w_m \neq w_i$. If $w_m$ violates internal consistency, output $(x, w_1, \ldots, w_{m-1}, w_i, 0, \ldots, 0)$. Otherwise $w_m$ is not a valid solution to an oracle gate query $u$ which encodes a ITER2 query $H : \{0,1\}^q \to \{0,1\}^q$, output $(x, w_1, \ldots, w_{m-1}, H(w_m), 0, \ldots, 0)$.

4. Say $y \neq \perp$, $y > x$, $z \neq \perp$, and the evaluation of $S^{2^*}(x, w_1, \ldots, w_{2t})$ is internally consistent. Output $(y, w_{t+1}, \ldots, w_{2t}, 0, \ldots, 0)$.

The reduction calls its ITER oracle on $S'$ to get back an answer $(x, w_1, \ldots, w_{2t})$ which would be the output of our reduction.

The reduction clearly runs in polynomial time since $S'$ translates to a polynomial size circuit and all other operations run in polynomial time. We now show the correctness. Let $v = (x, w_1, \ldots, w_{2t})$ be the answer the oracle returned. The following two equations must hold.

$$S'(v) > v \tag{1}$$

$$S'(S'(v)) \leq S'(v) \tag{2}$$

We will divide our proof of correctness by which cases are used to evaluate $S'(v)$ and $S'(S'(v))$. Before we dive into the case analysis, we note that $S'$ evaluated as case 1 and case 2 is the identity and could not satisfy Equation (1). Hence $S'(v)$ has to be evaluated using either case 3 or case 4. $S'$ evaluated as case 4 is strictly increasing. Hence $S'(S'(v))$ could not be evaluated as case 4.

1. $S'(v)$ is evaluated using case 3 and $S'(S'(v))$ is evaluated using case 1 or case 2. This case cannot happen. Since $S'(x, w_1, \ldots, w_{2t})$ was evaluated using case 3, $S^*(x, w_1, \ldots, w_t) \neq \perp$ and its evaluation is internally consistent. Let $S'(x, w_1, \ldots, w_{2t}) = (x, w'_1, \ldots, w'_{2t})$. Notice that $w_i = w'_i$ for all $i$ in $[1, t]$. Therefore, $S^*(x, w'_1, \ldots, w'_t) \neq \perp$ and its evaluation is internally consistent. Therefore, $S'(x, w'_1, \ldots, w'_{2t})$ will not be evaluated using case 1 or case 2.

2. $S'(v)$ is evaluated using case 3 and $S'(S'(v))$ is evaluated using case 3. This case cannot happen. Let $m$ be the identified index in $S'(x, w_1, \ldots, w_{2t})$ and $m'$ be the identified index in $S'(S'(x, w_1, \ldots, w_{2t}))$. Note that $m \leq m'$. Consider first when $m < m'$. Then $S'(x, w_1, \ldots, w_{2t}) = (x, w_1, \ldots, w_{m-1}, \hat{w}_m, 0, \ldots, 0)$ and $S'(x, w_1, \ldots, w_{m-1}, \hat{w}_m, 0, \ldots, 0) = (x, w_1, \ldots, w_{m-1}, \hat{w}_m, 0, \ldots, 0, \hat{w}_{m'}, 0, \ldots, 0)$ for some $\hat{w}_m$ and $\hat{w}_{m'} \neq 0$. Therefore, $S'(S'(x, w_1, \ldots, w_{2t})) > S'(x, w_1, \ldots, w_{2t})$, violating Equation (2). If $m = m'$, this can only happen when $w_m$ and $H(w_m)$ are not solutions to $u$. $S'(x, w_1, \ldots, w_{2t}) = (x, w_1, \ldots, w_{m-1}, H(w_m), 0, \ldots, 0)$ and $S'(S'(x, w_1, \ldots, w_{2t})) = (x, w_1, \ldots, w_{m-1}, H(H(w_m)), 0, \ldots, 0)$. But since $w'_m$ was not a solution to ITER2 on instance $H$, either $H(w'_m) = w'_m$ or $H(H(w'_m)) > H(w'_m)$. But we know from the Equation (1) that $H(w'_m) > w'_m$, therefore $H(H(w'_m)) > H(w'_m)$. Which implies $S'(S'(x, w_1, \ldots, w_{2t})) > S'(x, w_1, \ldots, w_{2t})$, contradicting Equation (2).

3. $S'(v)$ is evaluated using case 4 and $S'(S'(v))$ is evaluated using case 1. This case cannot happen. In particular, since $S'(x, w_1, \ldots, w_{2t})$ is evaluated using case 4, we know $S^*(y, w_{t+1}, \ldots, w_{2t}) \neq \perp$ and is internally consistent.

4. $S'(v)$ is evaluated using case 4 and $S'(S'(v))$ is evaluated using case 2. Notice that $S'(x, w_1, \ldots, w_{2t}) = (y, w_{t+1}, \ldots, w_{2t}, 0, \ldots, 0)$ where $S^*(y, w_{t+1}, \ldots, w_{2t}) = z$. Therefore, $S'(S'(x, w_1, \ldots, w_{2t}))$ being evaluated using case 2 means that $z \leq y$. The reduction therefore outputs a valid solution in this case.

5. $S'(v)$ is evaluated using case 4 and $S'(S'(v))$ is evaluated using case 3. Let $S'(x, w_1, \ldots, w_{2t}) = (y, w_{t+1}, \ldots, w_{2t}, 0, \ldots, 0)$. Evaluation by case 3 implies that $S'(y, w_{t+1}, \ldots, w_{2t}, 0, \ldots, 0) = (y, w_{t+1}, \ldots, w_{2t}, 0, \ldots, 0, \hat{w}_m, 0, \ldots)$. As such, $S'(S'(x, w_1, \ldots, w_{2t})) > S'(x, w_1, \ldots, w_{2t})$, contradicting Equation (2).

□

**Theorem 6.5.** $\mathsf{PLS}^{\mathsf{PLS}} = \mathsf{PLS}$.

*Proof.* Sink-of-DAG trivially reduces to Sink-of-DAG$^{\text{Sink-of-DAG}}$.

We now show the other direction. Note that all the following conditions are satisfied.

1. Sink-of-DAG has a black-box many-one reduction to Iter.
2. Iter$^{\text{Iter2}}$ has a Turing reduction to Iter by Lemma 6.4.
3. Iter2 is many-one reducible to Iter.
4. Iter is Turing-closed since PLS is Turing-closed.

Therefore, Corollary 4.5 tells us Sink-of-DAG$^{\text{Iter2}}$ has a many-one reduction to Iter, which has a many-one reduction to Sink-of-DAG. Finally, by Theorem 4.7, Sink-of-DAG$^{\text{Sink-of-DAG}}$ has a many-one reduction to Sink-of-DAG$^{\text{Iter2}}$. Chaining these reductions gives us a reduction from Sink-of-DAG$^{\text{Sink-of-DAG}}$ to Sink-of-DAG.

□

# 7 LOSSY Self-lowness

In this section, we show LOSSY is self-low.

**Lemma 7.1.** Lossy$^{\text{Lossy}}$ *reduces to* $(n-1)$-Lossy *under many-one reductions.*

*Proof.* Let $C : \{0,1\}^n \to \{0,1\}^{n/2}, D : \{0,1\}^{n/2} \to \{0,1\}^n$ be the circuits that act as input to our Lossy$^{\text{Lossy}}$ problem. Say that $C, D$ collectively have $t$ Lossy gates, where the $i^{\text{th}}$ gate has a $s_i$ bit output.We further assume without loss of generality that any input $c : \{0,1\}^q \to \{0,1\}^{q/2}, d : \{0,1\}^{q/2} \to \{0,1\}^q$ to a Lossy oracle gate has the form $q \geq 100 \log(t + 100)$. This can be achieved by padding and applying Lemma 2.14.

We now construct

$$C' : \{0,1\}^n \times \prod_i \{0,1\}^{s_i} \to \{0,1\}^{n+\sum s_i - 1},$$

$$D' : \{0,1\}^{n+\sum s_i - 1} \to \{0,1\}^n \times \prod_i \{0,1\}^{s_i} \ .$$

Let $N := n + \sum_{i=1}^{t} s_i$. Let $D \circ C : \{0,1\}^n \to \{0,1\}^n$ be the composed circuit which consists of $C$ followed by $D$. Let $x_2 = C_*(x_1, w_1, \ldots, w_t)$ and $x_3 = (D \circ C)_*(x_1, w_1, \ldots, w_t)$.

$C'(x_1, w_1, \ldots, w_t)$ computes $x_3 = (D \circ C)_*(x_1, w_1, \ldots, w_t)$. If $x_3$ is an error output $(m, z)$ where $z$ encodes a Lossy instance $c : \{0,1\}^q \to \{0,1\}^{q/2}, d : \{0,1\}^{q/2} \to \{0,1\}^q$, $C'$ outputs $(1, m, x_1, \ldots, w_{m-1}, c(w_m), w_{m+1} \ldots, w_t, 0, \ldots, 0) \in \{0,1\}^{N-1}$. Otherwise, $C'$ outputs $(0, x_2, w_1, \ldots, w_t, 0, \ldots, 0) \in \{0,1\}^{N-1}$ where $x_2 = C_*(x_1, w_1, \ldots, w_t)$.

Next we now define $D'$. We first consider the case when input to $D'$ has the form $(0, x_2, w_1, \ldots, w_t, 0, \ldots, 0)$. $D'$ computes $x_3 = D_*(x_2, w_1, \ldots, w_t)$. If $x_3$ is an err output, $D'$ outputs 0. Otherwise, $D'$ outputs $(x_3, w_1, \ldots, w_t)$.

When input to $D'$ has the form $(1, m, x_1, \ldots, w_t, 0, \ldots, 0)$. $D'$ computes $(D \circ C)_*(x_1, w_1, \ldots, w_{m-1}, 0^{s_m}, w_{m+1}, \ldots, w_t)$. If this results in a result output, $D'$ outputs 0. Say it results in an error output $(m', z)$ where $z$ encodes a Lossy instance $c : \{0,1\}^q \to \{0,1\}^{q/2}, d : \{0,1\}^{q/2} \to \{0,1\}^q$. $D'$ outputs $(x_1, \ldots, w_{m-1}, d(w_m), w_{m+1}, \ldots, w_t)$.

The reduction feeds $C', D'$ to its Lossy oracle to get back $(v_1, w_1, \ldots, w_t)$. The reduction outputs $v_1$ as well as witnesses used to evaluate $(D \circ C)_*(v_1, w_1, \ldots, w_t)$ in order.

The reduction clearly runs in polynomial time. Notice also that since $c$ compresses by at least $q/2 \geq 50 \log(t + 100)$ bits and $m$ requires exactly $\log_2(t)$ bits to specify, $C'$ compresses by at least 1 bit. To show correctness, let $v_2 = C_*(v_1, w_1, \ldots, w_t)$ and $v_3 = (D \circ C)_*(v_1, w_1, \ldots, w_t)$. We consider two cases.

1. $v_3$ is a result output. Then $C'(v_1, w_1, \ldots, w_t) = (0, v_2, w_1, \ldots, w_t, 0, \ldots, 0)$. Notice that by construction, $D'(0, v_2, w_1, \ldots, w_t) = (v_3, w_1, \ldots, w_t)$. By assumption, $v_1 \neq v_3$. Therefore, $v_1$ as well as the witnesses among $w_1, \ldots, w_t$ used to evaluate $D(C(v_1))$ are a solution to our $\text{Lossy}^{\text{Lossy}}$ instance. The witnesses are all consistent since the first valid $w_i$ among $w_1, \ldots, w_t$ is used to evaluate the oracle gates at every step.

2. $v_3$ is an error output $(m, z)$. This cannot happen. Say $z$ encodes a Lossy instance $c : \{0,1\}^q \to \{0,1\}^{q/2}, d : \{0,1\}^{q/2} \to \{0,1\}^q$. Then $C'(v_1, w_1, \ldots, w_t) = (1, m, x_1, \ldots, c(w_m), \ldots, w_t, 0, \ldots, 0)$. Notice that since $m \notin M$ for the evaluation of $v_3 = (D \circ C)_*(v_1, w_1, \ldots, w_t)$, $(D \circ C)_*(v_1, w_1, \ldots, w_{m-1}, 0^{s_m}, w_{m+1}, \ldots, w_t)$ should have the exact same behaviour as $(D \circ C)_*(v_1, w_1, \ldots, w_t)$ and output $(m, z)$ and $w'_m$ is not a solution to Lossy on $c, d$. This is because $C_*$ always tries 0 as a solution to an oracle gate. As such, $D'(1, m, v_1, w_1, \ldots, c(w_m), \ldots, w_t, 0, \ldots, 0)$ evaluates to $(v_1, w_1, \ldots, d(c(w_m)), \ldots, w_t) = (v_1, w_1, \ldots, w_t)$. Therefore, $(v_1, w_1, \ldots, w_m, \ldots, w_t)$ is not a solution to our oracle call to Lossy on $C', D'$.

□

**Theorem 7.2.** $\mathsf{LOSSY}^{\mathsf{LOSSY}} = \mathsf{LOSSY}$.

*Proof.* Lossy trivially reduces to $\text{Lossy}^{\text{Lossy}}$.

$\text{Lossy}^{\text{Lossy}}$ has a many-one reduction to $(n-1)$-Lossy by Lemma 7.1, which reduces to Lossy by Lemma 2.14. Chaining the reductions tells us $\text{Lossy}^{\text{Lossy}}$ reduces to Lossy.

□

# 8 Further Applications

In this section, we demonstrate the potential of our new definitions of TFNP subclasses with TFNP oracles for developing better understanding of important computational problems.

Notably, our result that PPA is self-low provides a potential way to classify the problem of deterministically generating large primes inside TFNP. We first define the necessary problems.

**Definition 8.1** (Weak-Bertrand)**.** Given a string $1^n$, output a $32n$ bit prime $p$ such that $p > 2^n$.

We refer to this problem as Weak-Bertrand since Bertrand's postulate tells us that there always exists a prime between $2^n$ and $2^{n+1}$. The problem Bertrand would ask us to find such a prime. In Weak-Bertrand, we are asking for a prime between $2^n$ and $2^{32n}$. We now review relevant results.

**Lemma 8.2** ([Jeř16])**.** *Under the generalized Riemann hypothesis,* Factor *is in* PPA *and* PPP*.*

**Lemma 8.3** ([Kor22])**.** Weak-Bertrand *reduces to* Lossy$^{\text{Factor}}$ [1]*.*

We are able to leverage Lemma 8.2 and Lemma 8.3 to give a classification of Weak-Bertrand into our newly defined classes. Furthermore, the fact that PPA is self-low and Factor is in PPA (under the generalized Riemann hypothesis) suggests that Weak-Bertrand (or even Bertrand) may be easily reducible to a PPA-complete problem.

**Theorem 8.4.** *Under the generalized Riemann hypothesis,* Weak-Bertrand *is in* $\mathsf{LOSSY}^{\mathsf{PPA}}$*,* $\mathsf{LOSSY}^{\mathsf{PPP}}$*,* $\mathsf{PPADS}^{\mathsf{PPA}}$*, and* $\mathsf{PPADS}^{\mathsf{PPP}}$*.*

*Proof.* Both the fact that Weak-Bertrand is in $\mathsf{LOSSY}^{\mathsf{PPA}}$ and $\mathsf{LOSSY}^{\mathsf{PPP}}$ follow directly by combining Lemma 8.3 with Lemma 8.2. Observing that Lossy has a trivial relativizing reduction to the PPADS-complete problem Injective-Pigeon then implies Weak-Bertrand is in $\mathsf{PPADS}^{\mathsf{PPA}}$ and $\mathsf{PPADS}^{\mathsf{PPP}}$. □

**Theorem 8.5.** *If* Weak-Bertrand *is in* $\mathsf{PPA}^{\text{Factor}}$*, then the generalized Riemann hypothesis implies that* Weak-Bertrand *is in* PPA*.*

*Proof.* Apply Lemma 8.2 and Theorem 5.4. □

One interpretation of the above theorem is as follows: if PPA is sufficiently powerful to capture or 'derandomize' LOSSY, then it also captures Weak-Bertrand.

# 9 Acknowledgements

The authors would like to thank Karthik Gajulapalli, Sidhant Saraogi, and Noah Stephens-Davidowitz for many helpful discussions and feedback on an earlier draft of this manuscript. The authors would also like to thank the anonymous referees for useful comments.

[1] Technically, [Kor22] showed this for Lossy given access to an oracle which outputs all prime factors of a number rather than one non-trivial factor. We observe that one can obtain all factors of a number by simply applying Factor multiple times.

## References


[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004. 7

[BCE+95] Paul Beame, Stephen Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of np search problems. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 303–314, 1995. 1

[BHZ87] Ravi B Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987. 1

[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. 4, 5

[BOM04] Josh Buresh-Oppenheim and Tsuyoshi Morioka. Relativized np search problems and propositional proof systems. In *Proceedings. 19th IEEE Annual Conference on Computational Complexity, 2004.*, pages 54–67. IEEE, 2004. 1

[FGPR24] Noah Fleming, Stefan Grosser, Toniann Pitassi, and Robert Robere. Black-box ppp is not turing-closed. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1405–1414, 2024. 4, 12

[GHJ+24] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Separations in proof complexity and tfnp. *Journal of the ACM*, 71(4):1–45, 2024. 1

[Jeř16] Emil Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016. 19

[JLRX24] Siddhartha Jain, Jiawei Li, Robert Robere, and Zhiyang Xun. On pigeonhole principles and ramsey in tfnp. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 406–428. IEEE, 2024. 1

[KKMP21] Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. 1, 3

[Kor22] Oliver Korten. Derandomization from time-space tradeoffs. In *37th Computational Complexity Conference (CCC 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022. 3, 6, 7, 19

[Lau83] Clemens Lautemann. Bpp and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983. 1

[LLR24] Jiawei Li, Yuhao Li, and Hanlin Ren. Metamathematics of resolution lower bounds: A tfnp perspective, 2024. 3

[LPT24] Jiatu Li, Edward Pyne, and Roei Tell. Distinguishing, predicting, and certifying: On the long reach of partial notions of pseudorandomness. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–13, 2024. 4, 5

[Mor] T Morioka. Classification of search problems and their definability in bounded arithmetic, master's thesis, university of toronto, toronto, canada, 2001. 1

[Pud15] Pavel Pudlák. On the complexity of finding falsifying assignments for herbrand disjunctions. *Archive for Mathematical Logic*, 54(7):769–783, 2015. 1

[PYP22] Amol Pasarkar, Mihalis Yannakakis, and Christos Papadimitriou. Extremal combinatorics, iterated pigeonhole arguments, and generalizations of ppp. *arXiv preprint arXiv:2209.07625*, 2022. 4

[Tur39] Alan Mathison Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society, Series 2*, 45:161–228, 1939. 1



ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il