NEPHEW $\leq$ LOSSY

Proof: GPT-6 Astra
Exposition: Hanlin Ren, with Ti*k*Z help from Astra

October 20, 2026

The total search problem NEPHEW was defined in [FGJ+26] as a candidate problem in TFZPP that does not reduce to LOSSY. These two problems are defined as follows (in the decision tree setting):

Definition 1 (NEPHEW, [FGJ+26])**.**
Input: A "parent" function $f : [N] \to [N]$ and a "nephew" function $g : [N] \to [N]$
Output: Any element $x \in [N]$ that violates either of the following two "NEPHEW axioms": (1) $f(g(x)) \neq x$; (2) $f(f(g(x))) = f(x)$.

Definition 2 (LOSSY, [Kor22])**.**
Input: A "compressor" function $C : [2N] \to [N]$ and a "decompressor" function $D : [N] \to [2N]$
Output: Any element $x \in [2N]$ such that $D(C(x)) \neq x$.

In this note, we show that contrary to the conjecture in [FGJ+26], NEPHEW actually reduces to LOSSY:

Theorem 3. *There is a mapping reduction from* NEPHEW *to* LOSSY*.*

Intuition. Let $h(x) := g(f(g(x)))$. It is known that assuming the NEPHEW axioms, starting from any root r, the g-children and h-children forms a binary tree (not just a DAG). To reduce NEPHEW to LOSSY, we need to construct a compressor $C : [2M] \to [M]$ and a decompressor $D : [M] \to [2M]$. The compressor is simple: Given a string in $\{\mathtt{G}, \mathtt{H}\}^{O(\log N)}$, follow the $\{\mathtt{G}, \mathtt{H}\}$-path denoted by this string, and output the final node it ends at. However, to define the decompressor, we need to efficiently choose the correct children to walk down.

Before describing the decompressor, let us recall a useful property of NEPHEW. Suppose that $v \in [N]$ and all NEPHEW axioms are satisfied, then:

- $f(g(v)) \neq f(h(v))$: this is because otherwise $f(g(v)) = f(g(f(g(v))))$, and hence $v' := f(g(v))$ violates the NEPHEW axiom $f(g(v')) \neq v'$.
- $f(f(g(v))) = f(f(h(v))) = f(v)$: $f(f(g(v))) = f(v)$ is a NEPHEW axiom; for $f(f(h(v))) = f(v)$, expand $f(f(h(v))) = f(f(g(f(g(v))))) = f(f(g(v))) = f(v)$.

A useful intuition to have in mind is that the g-edges ($v \to g(v)$) and h-edges ($v \to h(v)$) can be "lifted", via f-edges, into reverse f-edges ($f(v) \leftarrow f(g(v))$) and ($f(v) \leftarrow f(h(v))$) (see Figure 2).

Now we describe the decompressor. Given an output $z \in [N]$ of the compressor, let us construct the f-path starting from z: $f(z)$, $f(f(z))$, $f(f(f(z)))$, etc. Since z was obtained from a path of g-edges and h-edges starting from the root r, the "lift" of this path is the above f-path from $f(z)$ to $f(r)$. (And if all the NEPHEW axioms are satisfied, then the top of this path must be $f(r)$.) Hence we can recover

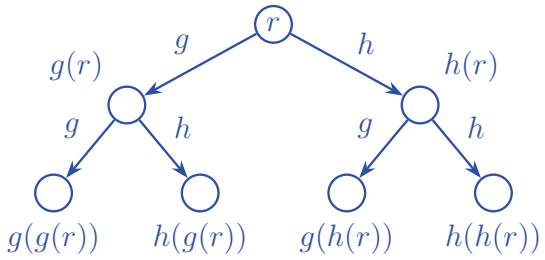


Figure 1: The g- and h-edges form a binary tree.

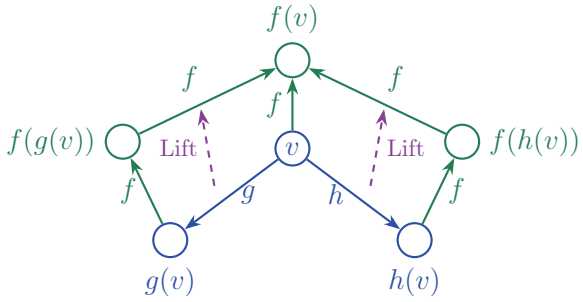


Figure 2: The g- and h-edges "lift" to f-edges.

the $\{\mathtt{G},\mathtt{H}\}$-path leading to z as follows. Start from r; whenever at a vertex w, find the children (either $g(w)$ or $h(w)$) whose f-edge points to the correct vertex on the f-path.

Here is a more formal proof.

Proof of Theorem 3. Fix an arbitrary root $r \in [N]$ and let $\ell := 2\lceil \log N \rceil$, we define the compressor $C : \{\mathtt{G},\mathtt{H}\}^\ell \to [N]$ and the decompressor $D : [N] \to \{\mathtt{G},\mathtt{H}\}^\ell$ as follows.

Algorithm 1 $C : \{\mathtt{G},\mathtt{H}\}^\ell \to [N]$

```
function C(s)
    v_0 ← r
    for i ← 1 to ℓ do
        if s_i = G then
            v_i ← g(v_{i−1})
        else                                   ▷ s_i = H
            v_i ← h(v_{i−1})
    return v_ℓ
```

Algorithm 2 $D : [N] \to \{\mathtt{G},\mathtt{H}\}^\ell$

```
function D(z)
    w_ℓ ← z
    for i ← ℓ to 1 do
        w_{i−1} ← f(w_i)
    u_0 ← r; ans ← ε                           ▷ ε is the empty string
    for i ← 1 to ℓ do
        if f(g(u_{i−1})) = w_{i−1} then
            u_i ← g(u_{i−1}); ans ← ans ∘ G
        else
            u_i ← h(u_{i−1}); ans ← ans ∘ H
    return ans
```

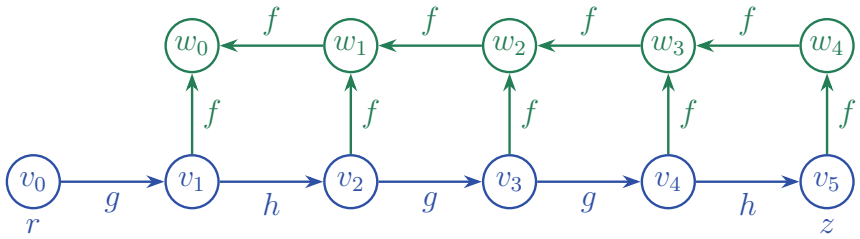


Figure 3: Running $z \leftarrow C(\mathtt{GHGGH})$ and $D(z)$.

The compressor C: Given a string $s \in \{\mathtt{G},\mathtt{H}\}^\ell$, starting from r we walk down the binary tree defined by the g and h pointers according to s. Then return the node $z \in [N]$ that we end up with.

The decompressor D: Given a node $z \in [N]$, the decompressor $D(z)$ first computes the path $f(z), f(f(z))$, etc. Then we walk down the tree from the root r for ℓ steps. To decide whether we go to the $\mathtt{G}$-child or the $\mathtt{H}$-child of the current node, simply check which child has a correct f value on the chain. Finally, return the $\{\mathtt{G},\mathtt{H}\}$-path we encountered.

For a more precise description of C and D (as well as some intermediate variables used in the proof below), see the pseudocode above.

Suppose that $s \in \{\mathtt{G},\mathtt{H}\}^\ell$ and $D(C(s)) \neq s$. Let $v_0 = r, v_1, \ldots, v_\ell$ denote the values of $v_0, \ldots, v_\ell$ in the execution of $C(s)$, then v_ℓ is the returned value of $C(s)$. Denote $z := v_\ell$, and let $\{w_i\}, \{u_i\}$ denote the values of $\{w_i\}$, $\{u_i\}$ in the execution of $D(z)$.

We first verify if $f(w_i) = f(v_i)$ for every $0 \leq i \leq \ell$. If this is not the case then we can solve NEPHEW as follows. Let $0 \leq i < \ell$ be the largest index such that $f(w_i) \neq f(v_i)$, then $f(v_{i+1}) = f(w_{i+1})$. If $v_{i+1} = g(v_i)$, then $f(w_i) = f(f(w_{i+1})) = f(f(v_{i+1})) = f(f(g(v_i))) \neq f(v_i)$ and v_i is a solution for NEPHEW. On the other hand, if $v_{i+1} = h(v_i)$ then $f(f(h(v_i))) \neq f(v_i)$ and we can find a solution for NEPHEW similarly.

Now, since $D(C(s)) \neq s$, let i be the first index such that $D(C(s))$ and s differs at the i-th character, then $u_{i-1} = v_{i-1}$. If the i-th bit of s is $\mathtt{G}$, then

$$f(g(u_{i-1})) = f(g(v_{i-1})) = f(v_i) = f(w_i) = w_{i-1},$$

and the i-th bit of $D(C(s))$ is $\mathtt{G}$ as well, a contradiction. Hence, the i-th bit of s is $\mathtt{H}$ and the i-th bit of $D(C(s))$ is $\mathtt{G}$. This means that

$$f(g(v_{i-1})) = f(g(u_{i-1})) = f(w_i) = f(v_i) = f(h(v_{i-1})) = f(g(f(g(v_{i-1})))),$$

which implies that $x := f(g(v_{i-1}))$ is a NEPHEW solution violating the axiom that $f(g(x)) \neq x$. □

The same argument shows that the totality of NEPHEW can be proven in $\mathsf{APC}_1(f, g)$. Here is a sketch: In $\mathsf{PV}_1(f, g)$ we can define the oracle circuits C and D. Let $s \in \{\mathtt{G},\mathtt{H}\}^\ell$ be any string. Assuming NEPHEW axioms, we can use a length induction on a polynomial-time predicate to prove that $f(w_i) = f(v_i)$ for every $i \in [\ell]$ (recall that ℓ is a length). Given this, we can use another length induction on a polynomial-time predicate to prove that every bit of s and $D(C(s))$ are equal, contradicting the retraction weak pigeonhole principle.

References

[FGJ+26] Noah Fleming, Stefan Grosser, Siddhartha Jain, Jiawei Li, Hanlin Ren, Morgan Shirley, and Weiqiang Yuan. "Total Search Problems in ZPP". In: *ITCS*. Vol. 362. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026, 60:1–60:26 (cit. on p. 1).

[Kor22] Oliver Korten. "Derandomization from Time-Space Tradeoffs". In: *CCC*. Vol. 234. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 37:1–37:26 (cit. on p. 1).