Range Avoidance for Constant Depth Circuits: Hardness and Algorithms

Karthik Gajulapalli
Georgetown University, Washington, DC, USA

Alexander Golovnev
Georgetown University, Washington, DC, USA

Satyajeet Nagargoje
Georgetown University, Washington, DC, USA

Sidhant Saraogi
Georgetown University, Washington, DC, USA

Abstract

Range Avoidance (Avoid) is a total search problem where, given a Boolean circuit $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^m$, $m > n$, the task is to find a $y \in \{0,1\}^m$ outside the range of C. For an integer $k \geq 2$, NC^0_k-Avoid is a special case of Avoid where each output bit of C depends on at most k input bits. While there is a very natural randomized algorithm for Avoid, a deterministic algorithm for the problem would have many interesting consequences. Ren, Santhanam, and Wang (FOCS 2022) and Guruswami, Lyu, and Wang (RANDOM 2022) proved that explicit constructions of functions of high formula complexity, rigid matrices, and optimal linear codes, reduce to NC^0_4-Avoid, thus establishing conditional hardness of the NC^0_4-Avoid problem. On the other hand, NC^0_2-Avoid admits polynomial-time algorithms, leaving the question about the complexity of NC^0_3-Avoid open.

We give the first reduction of an explicit construction question to NC^0_3-Avoid. Specifically, we prove that a polynomial-time algorithm (with an NP oracle) for NC^0_3-Avoid for the case of $m = n + n^{2/3}$ would imply an explicit construction of a rigid matrix, and, thus, a super-linear lower bound on the size of log-depth circuits.

We also give deterministic polynomial-time algorithms for all NC^0_k-Avoid problems for $m \geq n^{k-1}/\log(n)$. Prior work required an NP oracle, and required larger stretch, $m \geq n^{k-1}$.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases Boolean function analysis, Explicit Constructions, Low-depth Circuits, Range Avoidance, Matrix Rigidity, Circuit Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.65

Category RANDOM

Related Version *arXiv Version*: `https://arxiv.org/abs/2303.05044`

Acknowledgements We would like to thank Justin Thaler, Sam King, and anonymous reviewers for their helpful comments on our paper.

1 Introduction

The Range Avoidance (Avoid) problem is: given a Boolean circuit $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^m$ for some *stretch* $m > n$, find an element $y \in \{0,1\}^m$ outside the range of C. By the pigeonhole principle, such a y always exists. This problem was first introduced by Kleinberg, Korten, Mitropolsky, and Papadimitriou [15] as a complete problem for the class APEPP (Abundant Polynomial Empty Pigeonhole Principle). Informally, APEPP contains total search problems where the existence of a solution follows via the union bound (such as Shannon's classical proof that most functions require circuits of exponential size).

Korten [16] proved that a deterministic algorithm for Avoid would imply explicit constructions of objects that are central to the field of computational complexity, and would resolve several long-standing open problems. Such objects include functions of high circuit complexity, rigid matrices, pseudorandom generators, and Ramsey graphs. The key idea is that there is a succinct way to encode all "easy" objects (such as descriptions of functions of low circuit complexity) in the input space of a small circuit that acts as a decoder. Then a solution to the Avoid problem yields a "hard" object (such as a function of high circuit complexity), implying an explicit construction. In fact, the aforementioned works [15, 16] showed that even a deterministic algorithm *with an* NP *oracle* solving Avoid in polynomial time would lead to breakthrough results in complexity theory.

The only known deterministic algorithm for Avoid is the trivial brute force algorithm running in time $2^n \cdot \text{poly}(n, |\mathsf{C}|)$.[1] No better algorithms are known for Avoid even if the algorithm is allowed to use an NP oracle. On the other hand, using both an NP oracle and randomness, one can solve Avoid in polynomial time: Pick a random string $y \in \{0,1\}^m$, and simply check if $y \in \text{Range}(\mathsf{C})$ using the NP oracle. This shows that Avoid $\in \mathsf{FZPP}^{\mathsf{NP}}$, and, using the standard trick of simulating randomness by non-uniformity, Avoid $\in \mathsf{FP}^{\mathsf{NP}}/\mathsf{Poly}$.[2] Interestingly, for the class of polynomial-time algorithms with an NP oracle, the Avoid problem is equally hard for all values of stretch $n+1 \leq m \leq \text{poly}(n)$ [15].

For a class of circuits $\mathcal{C}$, the $\mathcal{C}$-Avoid problem is a special case of Avoid where each output is computed by a circuit from $\mathcal{C}$. Recent works by Ren, Santhanam, and Wang [19] and Guruswami, Lyu, and Wang [9] proved that efficient algorithms for $\mathcal{C}$-Avoid even for certain simple circuit classes $\mathcal{C}$ would be sufficient for getting various explicit constructions. Later, Chen, Huang, Li, and Ren [6] re-derived the best known lower bounds against ACC^0 circuits from an efficient algorithm for a certain $\mathcal{C}$-Avoid problem. While this suggests that designing efficient algorithms for Avoid problems is a promising approach to various explicit construction questions, the work of Ilango, Li, and Williams [10] proves barriers for designing polynomial time algorithms under certain cryptographic assumptions.

Let NC^1 denote the class of Boolean fan-in-2 circuits of depth $O(\log(n))$, and NC^0_k denote the class of Boolean functions where each output depends on at most k inputs for a constant k. [19] used perfect encodings of [13, 14, 2] to reduce NC^1-Avoid to NC^0_4-Avoid in polynomial time. Consequently, [9] reduced most of the aforementioned explicit constructions in [16] (and several new ones!) to NC^1-Avoid, and, thus, to NC^0_4-Avoid. In particular, polynomial-time *deterministic* algorithms (even with an NP oracle) for NC^0_4-Avoid would now imply breakthrough results in complexity theory.

[9] gave a polynomial-time algorithm solving NC^0_2-Avoid for any stretch $m \geq n+1$. As mentioned above, NC^0_4-Avoid might be hard to solve efficiently. This leaves the question about the complexity of NC^0_3-Avoid open.

▶ **Open Problem 1** ([9]). *Can we reduce explicit construction problems to solving* NC^0_3*-Avoid? Or can we solve* NC^0_3*-Avoid in polynomial time?*

Unlike the case of the general Avoid problem, NC^0_k-Avoid may be much easier for large

[1] This trivial algorithm is (conditionally) tight for a related problem studied in [15], where the range of C has size much smaller than 2^{n+1}, and is given by a circuit computing a function from $[N]$ to $[M]$. [15] gives a deterministic reduction from SAT on n variables to Avoid for a circuit $\mathsf{C}\colon [2^n] \to [2^n + 2^{o(n)}]$ running in subexponential time. Thus, under the Exponential Time Hypothesis [12, 11], this problem does not admit deterministic (and randomized) algorithms running in time $2^{o(n)}$.

[2] Here, the complexity classes $\mathsf{FP}, \mathsf{FE}, \mathsf{FZPP}$ are simply the functional analogs of the decision classes $\mathsf{P}, \mathsf{E}, \mathsf{ZPP}$.

values of the stretch m. Indeed, on one hand, NC^0_k-Avoid for small stretch $m = n + o(n)$ is capable of encoding hard explicit construction problems [19, 9]. On the other hand, NC^0_k-Avoid for $m = \Omega(n^k)$ is easily solvable in polynomial time: since the number of distinct functions depending on at most k out of n inputs is $O(n^k)$, every such instance of the problem must have two outputs computing identical functions. Assigning different values to these outputs solves NC^0_k-Avoid.

[9] presented an algorithm solving NC^0_k-Avoid for stretch $m \geq \Omega(n^{k-1})$ in polynomial time with an NP oracle.[3] This improvement on the trivial algorithm suggests a natural question of whether one can solve NC^0_k-Avoid for even smaller values of stretch m.

▶ **Open Problem 2.** *Design a polynomial-time algorithm (with an* NP *oracle) solving* NC^0_k-Avoid *with* n *inputs and stretch* $m = o(n^{k-1})$ *for* $k \geq 3$.

1.1 Our Results

The classical result of Shannon [20] shows that most Boolean functions of n variables require Boolean circuits of exponential size. Despite that, the best known lower bound on the size of a circuit (or even a circuit of logarithmic depth, i.e., NC^1) for a function in P (or even E^{NP}) is $3.1n - o(n)$ proven by Li and Yang [17]. A central problem in circuit complexity is to prove a super-linear lower bound on the number of gates of NC^1 circuits computing an explicit function [22, 3, Frontier 3].

Similarly, for the class of *linear* NC^1 circuits – NC^1 circuits where each gate computes the XOR (or its negation) of its two inputs – no super-linear lower bound on the complexity of an explicit linear map $M \in \mathbb{F}_2^{n\times n}$ is known. The best lower bound against linear circuits is $3n - o(n)$ proven by Chashkin [5].

In our first result (Theorem 9), we answer Open Problem 1 by showing that a polynomial-time algorithm for NC^0_3-Avoid would imply an explicit construction of a map requiring linear NC^1 circuits of super-linear size (thus, demonstrating the hardness of NC^0_3-Avoid).

▶ **Theorem 1.** *An* FP *(resp.* $\mathsf{FP}^{\mathsf{NP}}$*) algorithm for* NC^0_3-Avoid *with stretch* $m = n + O(n^{2/3})$ *implies an explicit construction of a linear map in* FP *(resp.* $\mathsf{FP}^{\mathsf{NP}}$*) that cannot be computed by linear* NC^1 *circuits of size* $o(n \log\log(n))$.

Our proof of Theorem 1 first reduces an explicit construction of a rigid matrix to NC^0_3-Avoid (Theorem 9). A matrix $M \in \mathbb{F}_2^{n\times n}$ is called (r, s)-rigid if it cannot be written as a sum $M = L + S$ of a rank-r matrix L and a matrix S with at most s non-zeros per row. In a seminal work, Valiant [22] introduced an approach for proving super-linear lower bounds on the size of linear NC^1 circuits via matrix rigidity. Valiant proved that an $(\varepsilon n, n^{\varepsilon})$-rigid matrix $M \in \mathbb{F}_2^{n\times n}$ for any constant $\varepsilon > 0$ requires linear NC^1 circuits of size $\Omega(n \log\log(n))$. Theorem 1 now follows straightforwardly as a corollary of Theorem 9.

The best known constructions of rigid matrices do not yet achieve the parameters sufficient for Valiant's circuit lower bound. [7, 18, 21] construct an $\left(r, \Omega\left(\frac{n}{r}\log\left(\frac{n}{r}\right)\right)\right)$-rigid matrix in polynomial time, [8] gives an $\left(r, \Omega\left(\frac{n^2}{r^2\log(n)}\right)\right)$-rigid matrix in time $2^{O(n)}$ for $r \geq \sqrt{n}$, and [1, 4] give $(2^{\varepsilon\log(n)/\log\log(n)}, \Omega(n))$-rigid matrices in polynomial time with an NP oracle. However, even an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for NC^0_3-Avoid with stretch $m = n + n^{12/17-\varepsilon}$ for any constant $\varepsilon > 0$ would already improve on these known constructions of rigid matrices.

[3] The algorithm of [9] does not use the full power of $\mathsf{FP}^{\mathsf{NP}}$: it outputs a hitting set $H \subseteq \{0,1\}^m$ such that for every NC^0_k function C, at least one point $y \in H$ is outside the range of C. Only then the algorithm looks at the input function and finds a solution $y \in H$ using the NP oracle.

In fact, we reduce the problem of constructing explicit rigid matrices to a problem that we call degree-2-Avoid, where each output computes a degree-2 polynomial of the inputs. Following Ren, Santhanam, and Wang's approach [19], this problem can be reduced to NC^0_3-Avoid using the perfect encoding scheme of Applebaum, Ishai, and Kushilevitz [2].[4]

On the algorithmic side, we make partial progress towards resolving Open Problem 2. We first give a simple deterministic polynomial-time algorithm for NC^0_3-Avoid for stretch $m \geq \binom{n}{2}/3 + 2n$ (presented in Appendix A). This algorithm already improves on the best known algorithm for NC^0_3, as it does not use an NP oracle. Then, in Theorem 4 we extend this algorithm to solve NC^0_k-Avoid for all constant k. Recall that the current best algorithms for this problem solve the case where $m \geq \Omega(n^{k-1})$ in polynomial time using an NP oracle [9]. We improve this result in two directions: our algorithm does not use an NP oracle, and it works in polynomial time for stretch $m \geq n^{k-1}/\log(n)$.

▶ **Theorem 2.** *There is a deterministic polynomial-time algorithm that solves the* NC^0_k*-Avoid problem with n inputs and stretch m for every $k \geq 3$ and $m \geq n^{k-1}/\log(n)$.*

1.2 Proof Overview

1.2.1 Hardness of NC^0_3-Avoid

Valiant [22] proved that linear NC^1 circuits with a linear number of gates can only compute non-rigid linear maps $M \in \mathbb{F}_2^{n \times n}$, i.e., maps M that can be written as a sum $M = Q + S$, where $\mathrm{rank}(Q) \leq \varepsilon n$ and each row of S has at most n^δ ones in it. For the rest of the section, our non-rigid matrices can be written as the sum of a matrix with rank $\leq n/10$ and a matrix with row sparsity at most $n^{0.1}$. Therefore, constructing a rigid matrix would imply a super-linear lower bound on the size of linear NC^1 circuits computing it.

To reduce an explicit construction of an $n \times n$ rigid matrix to solving an instance of NC^0_3-Avoid, we design an NC^0_3 function $f\colon \{0,1\}^{p(n)} \to \{0,1\}^{n^2}$, for some polynomial $p(n) < n^2$, such that for every non-rigid matrix $M \in \{0,1\}^{n \times n}$, there exists $x \in \{0,1\}^{p(n)}$ satisfying $f(x) = M$. Now, any solution $M' \in \{0,1\}^{n \times n}$ to the NC^0_3-Avoid problem for the function f must be a rigid matrix.

Before constructing such an NC^0_3 function f, we first design a function $g\colon \mathbb{F}_2^{n^2/2} \to \mathbb{F}_2^{n^2}$, where each output bit of g is a degree-2 polynomial of the inputs, and the range of g contains all non-rigid matrices. A solution to the degree-2-Avoid problem for the function g would give us a rigid matrix. Following [19], we can then apply a perfect encoding scheme [13, 14, 2] to g, and obtain an NC^0_3 function f, as required (see Lemma 6). Effectively, this reduces solving Avoid on g to solving Avoid on f.

Now we construct a degree-2 function $g\colon \mathbb{F}_2^{n^2/2} \to \mathbb{F}_2^{n^2}$ whose inputs encode all non-rigid matrices, i.e., for every non-rigid matrix M, there is an $x \in \{0,1\}^{n^2/2}$ such that $f(x) = M$. A non-rigid matrix M can be written as $M = LR + S$, where $L, R^T \in \mathbb{F}_2^{n \times n/10}$, and each row of S contains at most $n^{0.1}$ ones. The first $n^2/5$ inputs of the function g will correspond to the elements of L and R. Note that every entry of LR is a degree-2 function of the entries of L and R since it just computes the inner product of a row in L and a column in R. Now, for each $n^{0.1}$-sparse row of S, we show how to encode it using $n^{0.6}$ inputs and a degree 2 function. Repeating this procedure for each row of S will finish the proof.

---

[4] Ren, Santhanam, and Wang use the following definition of perfect encodings: A function $\widehat{f}$ is a perfect encoding of a function f if there exists a polynomial time algorithm Dec such that for all x, y: $\mathrm{Dec}(y) = f(x) \iff \exists r, y = \widehat{f}(x, r)$.

We interpret the $n^{0.1}$-sparse row with n entries as a $\sqrt{n} \times \sqrt{n}$ matrix A. Since A has at most $n^{0.1}$ non-zero entries, $\text{rank}(A) \leq n^{0.1}$. It can be written as a product $A = BC$ where $B, C^T \in \mathbb{F}_2^{\sqrt{n} \times n^{0.1}}$. Therefore, there is a degree-2 function h that takes as input B, C of size $2n^{0.6}$ and outputs the sparse matrix A.

The presented encoding of s-sparse vectors in $\mathbb{F}_2^n$ is only non-trivial for $s < \sqrt{n}$ (as otherwise the number of inputs of h exceeds the number of outputs). As a result, we cannot encode the entire matrix S as an $n^{1.1}$-sparse vector in $\mathbb{F}_2^{n^2}$. However, for Valiant's approach of proving circuit lower bounds, we can assume that S is $n^{0.1}$-*row* sparse.[5] Thus, we can separately encode each row of S using only $O(n^{0.6})$ inputs to obtain an encoding of S in $O(n^{1.6})$ bits. In Lemma 8, we will demonstrate how to accommodate slightly better sparsity parameter when we are allowed higher (but still constant) degree d for the encoding function.

1.2.2 Simple algorithm for NC^0_3-Avoid

We start with a short description of a simple deterministic polynomial-time algorithm for NC^0_3-Avoid for stretch $m \geq \binom{n}{2}/3 + 2n$ (presented in Appendix A). This algorithm already improves on the best known algorithm for NC^0_3-Avoid, as our algorithm does not use an NP oracle.

If we had a #SAT oracle, then we could solve NC^0_3-Avoid even for stretch $m = n + 1$. Our algorithm would iteratively find constant assignments to each of the first n outputs to minimize the number of inputs that map to the current (partial) output assignments. Throughout our exposition, we say such inputs are *consistent with the (partial) output assignments*. Before we describe our algorithm, it is important to note that we always fix circuit outputs to constant assignments. At each iteration, we would use the #SAT oracle to find the output assignment that reduces the size of the input set by at least half. After fixing the first n outputs, we still have at least $m - n \geq 1$ unassigned outputs, and only one input point $x \in \mathbb{F}_2^n$ that is consistent with the previously assigned output bits. This allows us to find an assignment of the $(n+1)$-th output bit such that the string specified by the output bits lies outside the range of the circuit.

Unfortunately, solving #SAT (even approximately) is hard for this class of multi-output circuits. In the absence of an efficient #SAT algorithm, our algorithm maintains an affine subspace $\mathcal{S}$ that contains all inputs from $\mathbb{F}_2^n$ that are consistent with the current partial assignment ($\mathcal{S}$ may also contain inputs that are not consistent with the current partial assignment). We carefully set output values so that at each iteration, we reduce the dimension of $\mathcal{S}$ by at least one. This way, after $n + 1$ steps we will find a solution to the NC^0_3-Avoid problem. However, our algorithm can only work when the stretch is $m \geq \Omega(n^2)$.

Without loss of generality, we assume that each output reads exactly three input bits. At each iteration the number of currently unassigned outputs is $> \binom{n}{2}/3$. This allows us to find a pair of outputs y_1 and y_2 that share a pair of input variables.[6] Say, $y_1 = f_1(x_1, x_2, x_3)$ and $y_2 = f_2(x_2, x_3, x_4)$. We will find a constant assignment to y_1 and y_2 that reduces the dimension of the affine subspace $\mathcal{S}$.

Note that there are 16 assignments to (x_1, x_2, x_3, x_4) and four different values of (y_1, y_2). Therefore, there is a way to assign $y_1 = c_1, y_2 = c_2$ such that at most four points (x_1, x_2, x_3, x_4) map to these values of the outputs. Note that there always exists a hyperplane H containing

[5] Alternatively, one can argue by Markov's inequality that if a matrix M cannot be written as a sum of rank-r and $n^{0.1}$-*row* sparse matrices, then M also cannot be written as a sum of rank-$2r$ and $rn^{0.1}$-*globally* sparse matrices.

[6] Each output sees 3 pairs of input bits, giving a total of $3m > \binom{n}{2}$ pairs. By the pigeonhole principle, at least one pair of inputs appears in two outputs.

any four points in $\mathbb{F}_2^4$. Let $\mathcal{H}$ be the affine subspace obtained by extending H to all n inputs. Then, $\mathcal{S} \cap \mathcal{H}$ gives us an affine subspace containing all inputs consistent with the assignment $y_1 = c_1, y_2 = c_2$.

If $\mathcal{S} \not\subseteq \mathcal{H}$, then we have an assignment of two outputs that reduces the dimension of our affine subspace as $\dim(\mathcal{S} \cap \mathcal{H}) < \dim(\mathcal{S})$. Otherwise, if $\mathcal{S} \subseteq \mathcal{H}$, then instead of considering all 16 assignments to the inputs (x_1, x_2, x_3, x_4), we can restrict our attention to at most 8 such assignments that belong to H, which makes the problem only easier (as we show in Theorem 16).

1.2.3 Better algorithm for NC^0_k-Avoid

The main bottleneck of this simple algorithm is that it maintains an affine subspace that must contain all consistent inputs. While affine subspaces are easy to work with, they are not expressive enough to accurately describe all inputs that are consistent with an arbitrary partial assignment. In order to improve the previous algorithm, we will maintain a more expressive structure than an affine subspace – a union of affine subspaces. Below we sketch our approach to solving NC^0_3-Avoid with stretch $m \geq \Omega(n^2/\log(n))$. Theorem 4 generalizes this to solving NC^0_k-Avoid problems with stretch $m \geq \Omega(n^{k-1}/\log(n))$ for all values of k.

Again, without loss of generality, we assume that each output depends on exactly three inputs. Consider a bipartite graph, where the left vertices correspond to n inputs, the right vertices correspond to m outputs, and an input-output pair (x_i, y_j) is connected by an edge if the output y_j depends on the input x_i. First we select $t = 3n^2/m$ highest-degree inputs $I = \{x_1, \ldots, x_t\}$. Their neighborhood must contain at least $3n$ distinct outputs $O = \{y_1, \ldots, y_{3n}\}$.[7] Let C be the sub-circuit defined on outputs from O and their corresponding inputs. Now, we will find a $y \in \mathbb{F}_2^{3n}$ outside Range(C).

First, consider all 2^t assignments to the inputs in $I = \{x_1, \ldots, x_t\}$, resulting in circuits $\mathsf{C}_1, \ldots, \mathsf{C}_{2^t}$. Since every output in O is connected to at least one input from I, fixing an assignment to the inputs I reduces each C_i to an NC^0_2 circuit. In a way, we have reduced NC^0_3-Avoid to an OR of 2^t instances of NC^0_2-Avoid: we need to find a $y \in \{0,1\}^{3n}$ outside the ranges of all the C_i's. For each circuit C_i, we will maintain an affine subspace $\mathcal{S}_i$ containing all inputs consistent with the current partial assignment of the outputs.

Our algorithm works by iteratively fixing the output bits from $\{y_1, \ldots, y_{3n}\}$ such that at each step the total number of points in the (disjoint) union of the affine subspaces $\mathcal{S}_i$ is reduced by a constant factor, eventually making all the subspaces empty. We observe (in Lemma 13) that for any affine subspace $\mathcal{S}_i$, one of the assignments $y_i = 0$ or $y_i = 1$ always reduces the dimension of $\mathcal{S}_i$ by one. Therefore, by picking the "best" assignment $y_i = c$ across all the subspaces $\mathcal{S}_i$, we can reduce the size of the union of such affine subspaces by a constant factor of $4/3$. Repeating this procedure for $\log_{4/3}(2^n) + 1 < 3n$ steps finishes the proof.

1.3 Open Problems

Our work motivates several natural questions about the complexity of NC^0_3-Avoid and degree-2-Avoid. We reduce explicit constructions of rigid matrices to solving degree-2-Avoid, and then NC^0_3-Avoid, with appropriate stretch.

[7] Since the number of outputs is m, and each output has degree 3, the number of edges in the graph is $3m$, and the average degree of an input is $3m/n$. The t highest-degree inputs then have total degree at least $3mt/n$, and must be connected to at least $mt/n = 3n$ distinct outputs.

▶ **Open Problem 3.** *Can other explicit construction questions be reduced to* NC^0_3*-*Avoid *or degree-*2*-*Avoid*?*

Particularly, we suspect that the construction of linear and list-decodable codes with optimal parameters [9] might be good candidates for these reductions.

Using the encoding of [2] in the reduction from degree-2-Avoid to NC^0_3-Avoid almost always decreases the required stretch to $m = n + o(n)$ (as highlighted in Section 3). It would also be interesting to find a more efficient encoding or reduction from degree-2-Avoid to NC^0_3-Avoid. This could potentially increase the stretch for NC^0_3-Avoid required to obtain explicit constructions thereby making the problem easier.

▶ **Open Problem 4.** *Can we construct a more efficient reduction from degree-*2*-*Avoid *to* NC^3_0*-*Avoid*?*

We believe degree-2-Avoid might be of independent interest since it allows for a larger stretch. For example, for improved constructions of rigid matrices, it suffices to solve degree-2-Avoid for super-linear stretch $m \geq n^{12/11-\varepsilon}$ for a constant $\varepsilon > 0$. In fact, degree-2-Avoid is easy to solve when the stretch is $m \geq n^2$. Note that there are at most $\binom{n}{2}$ unique degree-2 monomials on n variables. If $m \geq n^2$, then we can replace each unique monomial with a new variable. As a result, we will have m linear functions in $< m$ variables. We can solve Avoid on this linear function instance by a dimension reduction strategy similar to the one outlined in the previous section.

▶ **Open Problem 5.** *Are there algorithmic techniques to solve degree-*2*-*Avoid *that do not use a reduction to* NC^0_3*-*Avoid*?*

For the NC^0_3-Avoid problem, our algorithm runs in deterministic time $2^{O(n^2/m)}$ for any stretch $m \geq n + 1$. In particular, this recovers the exponential-time brute force algorithm for the hardest case of $m = n + 1$. It would be interesting to obtain matching conditional lower bounds for deterministic algorithms for NC^0_3-Avoid.

▶ **Open Problem 6.** *Is there a conditional lower bound of* $2^{\Omega(n^2/m)}$ *on the complexity of deterministic algorithms without an* NP *oracle for* NC^0_3*-*Avoid*?*

Finally, it is natural to ask if algorithms with NP oracles can solve NC^0_3-Avoid more efficiently.

▶ **Open Problem 7.** *Do there exist polynomial-time algorithms with* NP *oracles that solve* NC^0_3*-*Avoid *for stretch* $m = o(n^2/\log(n))$*?*

1.4 Structure

The rest of the paper is organized as follows. In Section 2, we give all necessary background material, including a reduction from degree-d-Avoid to NC^0_{d+1}-Avoid in Section 2.3. In Section 3, we reduce the problem of constructing explicit rigid matrices to NC^0_3-Avoid. In Section 4, we give deterministic algorithms solving NC^0_k-Avoid in polynomial time for stretch $m \geq n^{k-1}/\log(n)$. Finally, Appendix A contains an alternative deterministic polynomial-time algorithm for NC^0_3 for the case where stretch $m \geq \Omega(n^2)$.

2 Preliminaries

For every $a \in \mathbb{F}_2^n$ and subspace L of $\mathbb{F}_2^n$, we can define an affine subspace $\mathcal{A} \subseteq \mathbb{F}_2^n$ where $\mathcal{A} = \{a + v \mid v \in L\}$. The dimension of the affine subspace $\dim(\mathcal{A})$ is the same as the dimension of the linear subspace L that defines it. Equivalently, the set of points that lie on a specified set of hyperplanes over $\mathbb{F}_2^n$ also characterize an affine subspace of $\mathbb{F}_2^n$. The hyperplanes can be written as a system of linear equations $Ax = b$, and the dimension of the corresponding affine subspace $\mathcal{A}$ can be calculated as $\dim(\mathcal{A}) = n - \text{rank}(A)$.

The circuits and algorithms in this paper generally work over the boolean hypercube $\{0,1\}^n$. We work with multi-output circuits $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ where $m > n$ and m is called the stretch of the circuit. A partial output assignment, $y \in \{0,1,*\}^m$, is a fixing of a subset of the output bits of the circuit to constants. For an input $x \in \{0,1\}^n$ to the circuit, we say *x is consistent with a partial output assignment $y \in \{0,1,*\}^m$*, if $\mathsf{C}(x)$ agrees with y on the fixed bits. When specified, the input (resp. output) space of a circuit might instead be viewed as the vector space $\mathbb{F}_2^n$ (resp. $\mathbb{F}_2^m$) over the finite field $\mathbb{F}_2$.

The complexity classes $\mathsf{FP}, \mathsf{FP}^{\mathsf{NP}}, \mathsf{FE}$, and $\mathsf{FE}^{\mathsf{NP}}$ are classes of search problems analogous to the classes of decision problems $\mathsf{P}, \mathsf{P}^{\mathsf{NP}}, \mathsf{E}$, and E^{NP}. For example, the class FP contains all functions that can be computed by deterministic polynomial-time Turing machines.

2.1 Circuits and Matrix Rigidity

In this paper, we work with circuit classes NC^0_k and NC^1, which we define below.

▶ **Definition 1** (NC Circuits). *The circuit class NC^i contains multi-output Boolean circuits on n inputs of depth $O(\log^i(n))$ where each gate has fan-in 2. We are particularly concerned with the following classes of circuits:*

- *For every constant $k \geq 1$, NC^0_k is the class of circuits where each output depends on at most k inputs.*
- *NC^1 is the class of circuits of depth $O(\log(n))$ where all gates have fan-in 2.*
- *Linear NC^1 circuits are circuits of depth $O(\log(n))$ where every gate has fan-in 2 and computes an affine function, i.e., the XOR of its two inputs or its negation.*

It is a long-standing open problem in circuit complexity to prove super-linear lower bounds on the size of (linear) NC^1 circuits computing an n-output function from FP or even $\mathsf{FE}^{\mathsf{NP}}$ [22, 3, Frontier 3]. Valiant [22] suggested an approach for proving super-linear lower bounds for linear NC^1 circuits using the notion of matrix rigidity.

▶ **Definition 2** (Matrix Rigidity). *For $r, s \in \mathbb{Z}^+$, a matrix $M \in \mathbb{F}_2^{n \times n}$ is (r,s)-*rigid *if M cannot be written as a sum*

$$M = L + S\,,$$

where $L, S \in \mathbb{F}_2^{n \times n}$, L is low rank, i.e., $\text{rank}(L) \leq r$, and S is row sparse, i.e., every row of S has at most s non-zero entries.

Valiant [22] proved that a linear operator given by a sufficiently rigid matrix requires linear NC^1 circuits of size at least $\Omega(n \log\log(n))$, but there are still no known constructions of such rigid matrices even in $\mathsf{FE}^{\mathsf{NP}}$.

▶ **Theorem 3** ([22]). *If a family of matrices $(M_n)_{n \geq 1}$, $M_n \in \mathbb{F}^{n \times n}$, is $(\varepsilon n, n^\delta)$-rigid for constant $\varepsilon, \delta > 0$, then the linear map $x \mapsto Mx$ requires linear NC^1 circuits of size $\Omega(n \log\log(n))$.*

2.2 Range Avoidance for Circuits

In the range avoidance problem, given a circuit C with n inputs and m outputs, $m > n$, the goal is to find an m-bit string outside the range of C.

▶ **Definition 3** (Avoid). *In the* Avoid *problem, given a description of a circuit* $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ *for* $m > n$*, the task is to find a* $y \in \{0,1\}^m$ *such that* $\forall x \in \{0,1\}^n \colon \mathsf{C}(x) \neq y$.

The function $m = m(n)$ is called the *stretch* of the multi-output circuit C. Note that Avoid is a total search problem, i.e., there always exists such a $y \in \{0,1\}^m$ since $m > n$. We focus on a more restricted problem where there is an additional promise that the input circuit C is from a fixed circuit class $\mathcal{C}$.

▶ **Definition 4** ($\mathcal{C}$-Avoid). *In the* $\mathcal{C}$-Avoid *problem, given a description of a circuit* $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ *for* $m > n$*, where* $\mathsf{C} \in \mathcal{C}$*, the task is to find a* $y \in \{0,1\}^m$ *such that* $\forall x \in \{0,1\}^n \colon \mathsf{C}(x) \neq y$.

In particular, we are concerned with NC^1-Avoid and NC^0_k-Avoid for constant $k \geq 1$. We will also consider the class of functions where each output is a multivariate polynomial of the inputs of degree at most d over $\mathbb{F}_2$.

▶ **Definition 5** (degree-d-Avoid). *In the* degree-d-Avoid *problem, given a description of a function* $\mathsf{C} \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ *for* $m > n$*, where each output can be computed by a polynomial of degree* $\leq d$ *in the* n *inputs, the task is to find a* $y \in \mathbb{F}_2^m$ *such that* $\forall x \in \mathbb{F}_2^n \colon \mathsf{C}(x) \neq y$.

2.3 Low Degree and Low Locality

Perfect randomized encodings were introduced by [2] for various cryptographic applications. We are interested in the following property of perfect encodings: For a Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ and its encoding $\widehat{f} \colon \{0,1\}^{n+\ell} \to \{0,1\}^{m+\ell}$, there exists a polynomial-time decoding algorithm, $\mathrm{Dec} \colon \{0,1\}^{m+\ell} \to \{0,1\}^m$, such that for all $y \in \{0,1\}^{m+\ell}$ and $x \in \{0,1\}^n$ satisfying $\mathrm{Dec}(y) = f(x)$, there exists $r \in \{0,1\}^\ell$ such that $y = \widehat{f}(x,r)$. This property can be used in Avoid reductions as follows. Given a solution to the Avoid problem for the function $\widehat{f}$, i.e., $y \notin \mathrm{Range}(\widehat{f})$, one can find a solution to the Avoid problem for the function f in polynomial time by simply computing $\mathrm{Dec}(y) \notin \mathrm{Range}(f)$.

[2] first encode NC^1 functions as degree-3 functions. Then, they encode every degree-d function as an NC^0_{d+1} function. Composing these two encodings provides an encoding of NC^1 functions in NC^0_4. Using this encoding, [19] provides a polynomial time reduction from NC^1-Avoid to NC^0_4-Avoid. We use only one part of the result from [2]: there is a polynomial time reduction from degree-d-Avoid to NC^0_{d+1}-Avoid. For completeness, we include the proof here.

▶ **Lemma 6.** *Let* $d \geq 2$ *be a constant, and* $f \colon \mathbb{F}_2^n \to \mathbb{F}_2^m$ *be a multi-output function where every output computes a sum of* k *monomials of degree* $\leq d$*. Then there exists a function* $\widehat{f} \colon \{0,1\}^{n+(2k-1)m} \to \{0,1\}^{2km}$ *computed by an* NC^0_{d+1} *circuit and a polynomial time algorithm* $\mathrm{Dec} \colon \{0,1\}^{2km} \to \{0,1\}^m$ *such that for all* x, y*, if* $\mathrm{Dec}(y) = f(x)$*, there exists an* $r \in \{0,1\}^{(2k-1)m}$ *such that* $\widehat{f}(x,r) = y$.

Proof. We follow the encoding constructed in [2]. First, we construct an encoding $\widehat{g}$ for each single output function g of f. Let $g(x) = T_1(x) + T_2(x) + \cdots + T_k(x)$ be a single output degree-d function where each $T_i(x)$ is a monomial of degree at most d. Consider the encoding of g, $\widehat{g} : \{0,1\}^n \times \{0,1\}^k \times \{0,1\}^{k-1} \to \{0,1\}^{2k}$ defined as follows

$$\begin{aligned}\widehat{g}(x,r,s) = (&T_1(x) - r_1, \quad T_2(x) - r_2, \quad \dots \quad T_{k-1}(x) - r_{k-1}(x), \quad T_k(x) - r_k,\\ &r_1 - s_1, \quad s_1 + r_2 - s_2, \quad \dots \quad s_{k-2} + r_{k-1} - s_{k-1}, \quad s_{k-1} + r_k)\,.\end{aligned}$$

Clearly, each output bit of $\widehat{g}$ can be computed by an NC^0_{d+1} circuit. We define a polynomial-time algorithm $\mathrm{Dec}_{\widehat{g}}\colon \{0,1\}^{2k} \to \{0,1\}$ such that if $\mathrm{Dec}_{\widehat{g}}(y) = g(x)$ then there exist r and s satisfying $\widehat{g}(x,r,s) = y$. Given $y \in \{0,1\}^{2k}$, $\mathrm{Dec}_{\widehat{g}}(y)$ sums up the bits of y modulo 2.

Suppose $\mathrm{Dec}_{\widehat{g}}(y) = g(x)$ for some $x \in \{0,1\}^n$, i.e.,

$$\mathrm{Dec}_{\widehat{g}}(y) = \sum_{j=1}^{2k} y_j = g(x) = \sum_{j=1}^{k} T_j(x)\,. \tag{1}$$

We will now show that there exist r and s such that $\widehat{g}(x,r,s) = y$. For each $j \in [k]$, we set $r_j = T_j(x) - y_j$. We also set $s_1 = r_1 - y_{k+1}$ and sequentially set $s_j = s_{j-1} + r_j - y_{k+j}$ for each $j \in \{2,\dots,k-1\}$. By definition, the first $2k-1$ bits of $\widehat{g}(x,r,s)$ equal the first $2k-1$ bits of y. For the last bit, note that:

$$\begin{aligned} s_{k-1} + r_k &= \sum_{i=1}^{k} r_i - \sum_{i=k+1}^{2k-1} y_i && \text{(by the definition of } s\text{)} \\ &= \sum_{i=1}^{k} T_i(x) - \sum_{i=1}^{2k-1} y_i && \text{(by the definition of } r\text{)} \\ &= y_{2k}\,. && \text{(by Equation (1))} \end{aligned}$$

Therefore, for the constructed r and s, $\widehat{g}(x,r,s) = y$, as required.

Suppose $f(x) = (f_1(x), f_2(x), \dots, f_m(x))$, where each $f_i(x)$ is a sum of at most k monomials of degree $\le d$. Let $\widehat{f_i}$ be the encoding of f_i as defined above. Then our encoding of f is simply a concatenation of the encodings of its individual outputs, $\widehat{f}\colon \{0,1\}^{n+(2k-1)m} \to \{0,1\}^{2km}$, where

$$\widehat{f}(x, r^{(1)}, r^{(2)}, \dots, r^{(m)}, s^{(1)}, s^{(2)}, \dots, s^{(m)}) = (\widehat{f_1}(x, r^{(1)}, s^{(1)}), \dots, \widehat{f_m}(x, r^{(m)}, s^{(m)}))\,. \tag{2}$$

On input $y = (y_1, y_2, \dots, y_m) \in \{0,1\}^{2km}$, the decoding algorithm returns

$$\mathrm{Dec}(y) = (\mathrm{Dec}_{\widehat{f_1}}(y_1), \dots, \mathrm{Dec}_{\widehat{f_m}}(y_m))\,. \tag{3}$$

Suppose $\mathrm{Dec}(y) = f(x)$ for some $x \in \{0,1\}^n$. Then $\mathrm{Dec}_{\widehat{f_i}}(y_i) = f_i(x)$ for all $i \in [m]$. By our proof above, there exists $r^{(i)}$ and $s^{(i)}$ such that $y_i = \widehat{f_i}(x, r^{(i)}, s^{(i)})$ and, thereby,

$$y = (y_1, \dots, y_m) = (\widehat{f_1}(x, r^{(1)}, s^{(1)}), \dots, \widehat{f_m}(x, r^{(m)}, s^{(m)})) = \widehat{f}(x, r^{(1)}, s^{(1)}, \dots, r^{(m)}, s^{(m)})\,.$$

Finally, Dec runs in time $O(mk)$ since it runs m iterations of $\mathrm{Dec}_{\widehat{f_i}}$ for each i, each of which simply computes a sum of $2k$ bits. Since each $\widehat{f_i}$ is in NC^0_{d+1}, so is $\widehat{f}$. ◀

Now, following [19], we conclude that there is a polynomial-time reduction from degree-d-Avoid to NC^0_{d+1}-Avoid.

▶ **Corollary 7.** *For every $d \ge 1$, if there exists an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for NC^0_{d+1}-Avoid, then there exists an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for degree-$d$-Avoid.*

Proof. Let f be an input to a degree-d-Avoid problem with m output bits. Then, each output bit of f is a sum of at most $k = O(n^d)$ monomials of degree d. Let $\widehat{f}$ be the encoding of f in NC^0_{d+1} guaranteed by Lemma 6. Note that $\widehat{f}\colon \{0,1\}^{n+(2k-1)m} \to \{0,1\}^{2km}$. By the assumption of the Corollary, there is an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm that returns a $y \notin \mathrm{Range}(\widehat{f})$. Then, by Lemma 6, $\mathrm{Dec}(y) \notin \mathrm{Range}(f)$ and Dec runs in polynomial time. Therefore, there is an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for degree-$d$-Avoid. ◀

3 Hardness of NC^0_3-Avoid

In this section, we reduce the problem of constructing explicit rigid matrices to the algorithmic task of solving NC^0_3-Avoid. First, in Lemma 8 we give an explicit degree-2 function $f\colon \{0,1\}^k \to \{0,1\}^n$, $k \ll n$, whose range contains all sparse vectors of length n. Note that such a function f must have degree at least 2. Indeed, if f was affine and its range contained all vectors of sparsity at most 1, then its range must have dimension n, and the number of inputs of f would be $k \geq n$.

Next, in Theorem 9, we apply this lemma, together with the reduction from degree-2-Avoid to NC^0_3-Avoid from Lemma 6, to conclude that an efficient algorithm for NC^0_3-Avoid would provide an explicit construction of rigid matrices.

▶ **Lemma 8.** *For every $d \geq 1$ and every polynomial-time computable $s := s(n) < \frac{n^{1-1/d}}{d}$, there exists a polynomial-time computable function $f\colon \mathbb{F}_2^{dsn^{1/d}} \to \mathbb{F}_2^n$ whose range contains all vectors of sparsity at most s, and each output of f is a degree-d polynomial.*

Proof. Let G be an arbitrary d-uniform hypergraph on $\ell = dn^{1/d}$ vertices and n hyperedges (such a graph exists because $\binom{\ell}{d} \geq (\ell/d)^d = n$). Fix an ordering $\{1,\dots,\ell\}$ of the vertices and $\{1,\dots,n\}$ of the edges. Each vertex of G will be labeled by a vector from $\mathbb{F}_2^s$. Our function $f\colon \mathbb{F}_2^{s\ell} \to \mathbb{F}_2^n$ will take as input the labels of the vertices of G and output n elements corresponding to the n hyperedges of G: the ith output is the generalized inner product of the labels of the d vertices in the ith hyperedge. We interpret the input as a matrix $X \in \mathbb{F}_2^{s\times\ell}$, where the jth column $X_j \in \mathbb{F}_2^s$ is the label corresponding to the jth vertex. Suppose the hyperedge i contains the vertices $\{j_1,\dots,j_d\}$ then the ith output is $f_i(X) = \sum_{k=1}^{s} X_{k,j_1}\cdots X_{k,j_d}$. Clearly, f is a degree-d function, it only remains to show that its output contains all vectors of sparsity $\leq s$. For this, we show that for every vector $y \in \mathbb{F}_2^n$ of sparsity $\leq s$, there is an input, i.e., a labeling of the vertices of G, such that f outputs y. Let the s non-zero elements of y correspond to the distinct edges $i_1,\dots,i_s$ in G. For each vertex j in G we set its label $X_j \in \mathbb{F}_2^s$ to be such that $(X_j)_k = 1$ if $j \in i_k$ and $(X_j)_k = 0$ otherwise.

Consider any edge $i = \{j_1,\dots,j_d\}$ and the submatrix $X_{j_1,\dots,j_d}$ of X containing the labels of these vertices connected by i.

- If $i = i_k$ for some $k \in [s]$, then $y_i = 1$. The i_kth row of $X_{j_1,\dots,j_d}$ contains all 1 entries. Furthermore, every other row contains at least one zero. Therefore, $f_i(X) = 1$.
- If $i \neq i_k$ for all $k \in [s]$, then $y_i = 0$ and each row of $X_{j_1,\dots,j_d}$ contains at least one zero. Therefore, $f_i(X) = 0$. ◀

Equipped with Lemma 8, we are ready to show that an efficient algorithm for degree-2-Avoid or NC^0_3-Avoid would imply an explicit construction of rigid matrices.

▶ **Theorem 9.** *For every constant $1/2 \leq \delta \leq 1$, an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for degree-2-Avoid with stretch $m = 2n^{2/(1+\delta)}$ will provide an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for finding an $(n^\delta/10, n^{\delta-1/2}/10)$-rigid matrix.*

Furthermore, for every $1/2 \leq \delta \leq 1$, an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for NC^0_3-Avoid with stretch $m = n + O(n^{2/(2+\delta)})$ will provide an FP (resp. $\mathsf{FP}^{\mathsf{NP}}$) algorithm for finding an $(n^\delta/10, n^{\delta-1/2}/10)$-rigid matrix.

Proof. Let $r = n^\delta/10$ and $s = n^{\delta-1/2}/10$. First, we reduce (in deterministic polynomial time) the problem of finding an (r,s)-rigid matrix to solving degree-2-Avoid for a function $g\colon \mathbb{F}_2^{4rn} \to \mathbb{F}_2^{n^2}$.

Suppose $M \in \mathbb{F}_2^{n\times n}$ is not (r,s)-rigid. Then, M can be written as a sum $M = Q + S$, where $\mathrm{rank}(Q) \leq r$ and S is s-row sparse. Furthermore, $Q = L \cdot R$ for some matrices $L, R^T \in \mathbb{F}_2^{n\times r}$.

We view the input of g as $2rn$ entries of the matrices L and R, and $2sn^{3/2}$ inputs of n copies of the degree-2 function f from Lemma 8 needed to encode the entries of n sparse rows of S. Then the function g simply outputs all n^2 entries of $M = L \cdot R + S$. Note that each output of g computes a dot-product of a row of L and a column of R, and adds a degree-2 output of f. Therefore, we constructed a degree-2 function g whose range contains all non-rigid matrices. A solution to degree-2-Avoid on input g would therefore give an (r,s)-rigid matrix. The number of inputs of g is $n' = 2rn + 2sn^{3/2} = 4rn = 2n^{1+\delta}/5$, and the stretch of the function g is at least $m'(n') \geq 2(n')^{2/(1+\delta)}$. This concludes the proof of the first part of the theorem.

For the second part, we use the polynomial-time reduction from degree-2-Avoid to NC^0_3-Avoid from Lemma 6. By the construction above, we have a degree-2 function $g\colon \mathbb{F}_2^{4rn} \to \mathbb{F}_2^{n^2}$ where each output bit is the sum of at most $t = r + s \leq 2r$ degree-2 monomials. We apply Lemma 6 to reduce Avoid for g to Avoid for an NC^0_3 function $\widehat{g}\colon \{0,1\}^{\widehat{n}} \to \{0,1\}^{\widehat{m}}$, where $\widehat{n} = n' + (2t-1)n^2$ and $\widehat{m} = 2tn^2$. This yields a stretch of $\widehat{m}(\widehat{n}) = \widehat{n} + O(\widehat{n}^{2/(2+\delta)})$ for the function $\widehat{g}$. Therefore, an algorithm for NC^0_3-Avoid for stretch $\widehat{m}(n)$ yields an (r,s)-rigid matrix. ◀

We remark that in the regime $\delta > 1/2$, Theorem 9 would give matrices that for rank n^δ have higher rigidity than all known constructions of rigid matrices in $\mathsf{FP}, \mathsf{FP}^{\mathsf{NP}}$ and $\mathsf{FE}^{\mathsf{NP}}$. Therefore, for every $\varepsilon > 0$, an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for degree-2-Avoid with stretch $n^{12/11-\varepsilon}$ or an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for NC^0_3-Avoid with stretch $n + n^{12/17+\varepsilon}$ would lead to new rigidity lower bounds. Since the regime of $\delta = 1$ in Theorem 9 is sufficient for Valiant's program of proving super-linear lower bounds on the size of linear NC^1 circuits (see Theorem 3), we have the following corollary.

▶ **Corollary 10.** *An* FP *(resp.* $\mathsf{FP}^{\mathsf{NP}}$*) algorithm for degree-2-Avoid with stretch* $m = 2n$ *or for* NC^0_3*-Avoid with stretch* $m = n + O(n^{2/3})$ *will provide a linear function in* FP *(resp.* $\mathsf{FP}^{\mathsf{NP}}$*) that cannot be computed by linear* NC^1 *circuits of size* $o(n \log\log(n))$.

4 Algorithms for NC^0_k-Avoid

In this section, we describe polynomial-time algorithms for solving NC^0_k-Avoid with non-trivial stretch. More specifically, we provide an algorithm that runs in time $2^{O(n^{k-1}/m)} \cdot \mathrm{poly}(n)$ when the stretch of the input circuit is at least $m \geq \Omega(n^{k-2})$. First, we describe a useful structural property of NC^0_k circuits, which follows from the following simple graph-theoretic result.

▶ **Lemma 11.** *For any constants* $c \geq 1$ *and* $k \geq 3$*, every* k*-uniform hypergraph* $G = (V, E)$ *with* n *vertices and* $m \geq cn^{k-2}$ *hyperedges contains a subset of vertices* $V' \subseteq V, |V'| \leq cn^{k-1}/m$ *and a subset of hyperedges* $E' \subseteq E, |E'| \geq cn$ *such that each hyperedge in* E' *contains at least* $k-2$ *vertices from* V'*. Furthermore, there is a polynomial-time algorithm that finds such a* V' *and* E'*.*

Proof. Consider the following bipartite graph H with vertex set $A \sqcup B$ where $A = \{u_S \mid S \subseteq V, |S| = k-2\}$ is the set of vertices indexed by the $k-2$ sized subsets of V and $B = \{v_e \mid e \in E\}$ is indexed by the edges of G. Furthermore there is an edge (u_S, v_e) in H if $S \subseteq e$, i.e., if all the vertices in S are contained in the hyperedge $e \in E$. Since each hyperedge

e contains k vertices, the degree of each vertex in B is exactly $\binom{k}{k-2}$. Then, the average degree of the vertices in A is $\frac{|B|\binom{k}{k-2}}{|A|} = \frac{m\binom{k}{k-2}}{\binom{n}{k-2}}$. Let $A' \subseteq A$ be the subset of t vertices with highest degree in A and let $N(A')$ be their neighbors in B. Then total degree of vertices in A' is at least $\frac{tm\binom{k}{k-2}}{\binom{n}{k-2}}$. Since each vertex in B has degree $\binom{k}{k-2}$, $|N(A')| \geq \frac{tm\binom{k}{k-2}}{\binom{n}{k-2}\binom{k}{k-2}} = \frac{tm}{\binom{n}{k-2}}$. Therefore, setting $t = \frac{cn\binom{n}{k-2}}{m}$, V' to be the set of $t(k-2) = \frac{cn(k-2)\binom{n}{k-2}}{m} \leq cn^{k-1}/m$ vertices of G contained in the union of the vertex subsets in A', and $E' = N(A')$ completes our proof.

To find V' and E', we first construct the graph H which has polynomial size, and then find the vertices in A' by finding the t vertices in A with maximum degree. It is now straightforward to construct V' from A' and to find $E' = N(A')$. ◀

▶ **Corollary 12.** *For any constants $c \geq 1$ and $k \geq 3$, given an NC^0_k circuit C with n inputs and $m \geq cn^{k-2}$ outputs, there exists a subset of outputs O of size $|O| \geq cn$, and a subset of inputs I of size $|I| \leq cn^{k-1}/m$, such that for every output bit $\mathsf{C}_i \in O$, at least $k-2$ of the input bits feeding into C_i are from I. Furthermore, there is a polynomial-time algorithm that finds such sets I and O.*

Proof. Without loss of generality we assume that each output of C reads exactly k inputs (as if it reads $\ell < k$ inputs, we let it additionally read arbitrary $k - \ell$ inputs and ignore them). Consider the hypergraph where each vertex corresponds to one of the n inputs $\{x_1, \ldots, x_n\}$ of C. Each edge of the hypergraph corresponds to an output C_i, $e_i = \{j \mid \mathsf{C}_i \text{ reads } x_j\}$. Now, we apply Lemma 11 on this hypergraph and set $I = V'$ and $O = E'$. Note that $|I| = |V'| \leq cn^{k-1}/m$ and $|O| = |E'| \geq cn$. ◀

This corollary finds a linear number of output bits O of the circuit that mostly depend on a small number of common input bits I. Our algorithm for NC^0_k-AVOID will "branch" on all possible assignments to the inputs from I. Each such assignment will correspond to an affine subspace $\mathcal{S} \subseteq \mathbb{F}_2^n$ of the input space. Then, our algorithm works by fixing the output bits from O such that the sum of the dimensions of these affine subspaces is significantly reduced at each step, eventually making all subspaces empty. Note that by the guarantee of Corollary 12, after fixing the inputs I, each output from O depends on at most two inputs. Thus, we need an efficient way to reduce the dimension of the affine subspace containing the consistent inputs for the case where output functions depend on at most two inputs. In Lemma 13, we provide such a subroutine AFFINEREDUCE (Algorithm 1).

▶ **Lemma 13.** *Let $\mathcal{S} \subseteq \mathbb{F}_2^n$ be an affine subspace, and $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a function that depends on at most two inputs. The algorithm AFFINEREDUCE in deterministic polynomial time finds two affine subspaces (or empty sets) $\mathcal{S}_0, \mathcal{S}_1 \subseteq \mathcal{S}$ such that*

(1) *$\forall x \in \mathcal{S}, b \in \mathbb{F}_2$, if $f(x) = b$, then $x \in \mathcal{S}_b$;*

(2) *$|\mathcal{S}_0| + |\mathcal{S}_1| \leq 3|\mathcal{S}|/2$.*

Proof. Without loss of generality we assume that $f(x)$ depends on (a subset of) x_1 and x_2. We will consider three cases depending on the degree of f, and in each case we will find affine subspaces (or empty sets) $\mathcal{S}_0, \mathcal{S}_1 \subseteq \mathcal{S}$ such that at least one of them has dimension strictly smaller than the dimension of $\mathcal{S}$ (or at least one of them is an empty set). This will ensure that $|\mathcal{S}_0| + |\mathcal{S}_1| \leq 3|\mathcal{S}|/2$.

- If $f(x) = c$ for some $c \in \mathbb{F}_2$ is a constant function, then we set $\mathcal{S}_c = \mathcal{S}$ and $\mathcal{S}_{1-c} = \emptyset$. Clearly, $\mathcal{S}_c = \mathcal{S}$ and $\mathcal{S}_{1-c}$ contain all points $x \in \mathcal{S}$ that are consistent with $f(x) = c$ and $f(x) = 1 - c$, respectively.

- If $f(x) = a_1x_1 + a_2x_2 + c$ for some constants $a_1, a_2, c \in \mathbb{F}_2$ is an affine function, then for each $b \in \mathbb{F}_2$ let H_b be the hyperplane defined by $a_1x_1 + a_2x_2 + c = b$, and let $\mathcal{S}_b = \mathcal{S} \cap H_b$. Again, $\mathcal{S} \cap H_b$ contains all the inputs in $\mathcal{S}$ that are consistent with $f(x) = b$. Furthermore, if $\dim(\mathcal{S}_0) = \dim(\mathcal{S})$, then $\mathcal{S} \subseteq H_0$ and $\mathcal{S}_1 = \mathcal{S} \cap H_1 = \emptyset$. Therefore, either $\dim(\mathcal{S}_0) < \dim(\mathcal{S})$ or $\mathcal{S}_1 = \emptyset$.
- If $f(x) = (x_1 + a_1)(x_2 + a_2) + c$ for some constants $a_1, a_2, c \in \mathbb{F}_2$ is a quadratic function, then let $\mathcal{H}$ be the affine subspace defined by $\mathcal{H} = \{x \in \mathbb{F}_2^n \mid x_1 = 1 + a_1, x_2 = 1 + a_2\}$. Consider the affine subspace $\mathcal{S}_{1-c} = \mathcal{S} \cap \mathcal{H}$ which contains all points $x \in \mathcal{S}$ satisfying $f(x) = 1 - c$.
  - If $\dim(\mathcal{S}_{1-c}) < \dim(\mathcal{S})$, then we are done as we can take $\mathcal{S}_c = S$
  - If $\dim(\mathcal{S}_{1-c}) = \dim(\mathcal{S})$, then $\mathcal{S} \subseteq \mathcal{H}$. Then, every point in $\mathcal{S}$ satisfies $f(x) = 1 - c$, thus, setting $\mathcal{S}_{1-c} = \mathcal{S}$ and $\mathcal{S}_c = \emptyset$ completes our construction.

In each case, either $|\mathcal{S}_0| \leq \frac{|\mathcal{S}|}{2}$ or $|\mathcal{S}_1| \leq \frac{|\mathcal{S}|}{2}$. Therefore, $|\mathcal{S}_0| + |\mathcal{S}_1| \leq 3|\mathcal{S}|/2$.

The only computation made by AffineReduce is to compute the dimensions of explicitly given affine subspaces, which can be performed in polynomial time. ◀

Algorithm 1 AffineReduce$(\mathcal{S}, f)$.

Input: Affine subspace $\mathcal{S} \subseteq \mathbb{F}_2^n$, $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ that may depend only on x_1 and x_2
Output: $\mathcal{S}_0, \mathcal{S}_1 \subseteq \mathcal{S}$

```
if f(x) = c then
    return S_c = S and S_{1-c} = ∅
if f(x) = a_1x_1 + a_2x_2 + c then
    For b ∈ F_2, let H_b = {x ∈ F_2^n : a_1x_1 + a_2x_2 + c = b}
    return S_0 = S ∩ H_0 and S_1 = S ∩ H_1
if f(x) = (x_1 + a_1)(x_2 + a_2) + c then
    Let H = {x ∈ F_2^n : x_1 = 1 + a_1, x_2 = 1 + a_2}
    Let S_{1-c} = S ∩ H
    if dim(S_{1-c}) < dim(S) then
        return S_{1-c} and S_c = S
    else
        return S_{1-c} and S_c = ∅
```

A simple application of AffineReduce recovers a polynomial-time algorithm for NC^0_2-Avoid from [9].

▶ **Corollary 14.** *There is a deterministic polynomial-time algorithm that, given an* NC^0_2 *circuit* $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^m$, $m \geq n + 1$, *finds an element* $y \in \{0,1\}^m, y \notin \mathrm{Range}(\mathsf{C})$.

Proof. At iteration $1 \leq i \leq n + 1$, our algorithm will fix the value of the ith output bit y_i. The algorithm also maintains an affine subspace $\mathcal{S} \subseteq \mathbb{F}_2^n$ that contains all inputs $x \in \mathbb{F}_2^n$ consistent with the partial output assignments of $y_1, \ldots, y_i$. By Lemma 13, there exists an assignment $y_i = b$, such that either none of the inputs in $\mathcal{S}_b$ are consistent with y or the dimension of $\mathcal{S} = \mathcal{S}_b$ reduces at least by one. In the former case, we already find our desired output y (we can just set the unassigned bits of y to arbitrary values). Otherwise, after fixing the first n outputs, we have $\dim(\mathcal{S}) = 0$, i.e., $S = \{x\}$ for some $x \in \mathbb{F}_2^n$. Let $b \in \{0,1\}$ be the value of the $(n+1)$th output bit of $\mathsf{C}(x)$. Then setting $y_{n+1} = 1 - b$ produces our desired output y. This algorithm runs in polynomial time since it makes at most n calls to AffineReduce and one call to $\mathsf{C}(x)$. ◀

Finally, equipped with Lemma 13, we are ready to present our main algorithm for NC^0_k-Avoid.

▶ **Theorem 4.** *Given an NC^0_k circuit $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^m$, where $m \geq 3n^{k-2}$, the algorithm SubspaceUnion finds an element $y \in \{0,1\}^m, y \notin \mathrm{Range}(\mathsf{C})$ in deterministic time $2^{O(n^{k-1}/m)} \cdot \mathrm{poly}(n)$.*

Proof. First we apply Corollary 12 with $c = 3$ to the circuit C, and select in polynomial time a subset of inputs $I = \{x_1, \ldots, x_t\}$ and a set of outputs $O = \{y_1, y_2, ..., y_{3n}\}$ for $t \leq 3n^{k-1}/m$. This ensures that each y_i has at most two inputs outside of I. For each of the 2^t assignments of the inputs from I, we consider a circuit where the values of these t inputs are fixed. Namely, for $j \in \{0, \ldots, 2^t - 1\}$, we fix the inputs in I to the bits in the binary representation of j. Then we restrict the circuit C to the outputs $y_1, \ldots, y_{3n}$ and all the inputs that feed them, and obtain a circuit C_j, where each output depends on at most 2 inputs. We'll find a value $y \in \{0,1\}^{3n}$ that no C_j outputs, and this will give us a solution to the original NC^0_k-Avoid instance.

Our algorithm will maintain the following invariant. At the ith iteration of the algorithm after we fix the values of the outputs $y_1, \ldots, y_i$, we maintain $\mathcal{U} = \bigcup_{j=0}^{2^t-1} \mathcal{U}_j$, a disjoint union of 2^t affine subspaces, such that all inputs $x \in \mathbb{F}_2^n$ that are consistent with $y_1, \ldots, y_i$ belong to $\mathcal{U}$ (and $\mathcal{U}$ may contain points that are inconsistent with $y_1, \ldots, y_i$, too).

In the beginning of the algorithm, for every $0 \leq j < 2^t$, we let $\mathcal{U}_j$ be the affine subspace where the inputs in I are fixed to the bits in the binary representation of j. Then $\mathcal{U} = \bigcup_j \mathcal{U}_j = \mathbb{F}_2^n$ is the set of all inputs consistent with our initial empty partial assignment.

At every step i, we will show how to find a constant $b \in \{0,1\}$ such that after fixing $y_i = b$, the size of our disjoint union $|\mathcal{U}|$ reduces by a factor of $4/3$. Therefore, after repeating this procedure for the $3n$ outputs from O, we will have an empty $\mathcal{U}$, and the constructed partial assignment will give us a solution to the NC^0_k-Avoid problem.

At the ith iteration of the algorithm, we have values of outputs $y_1, \ldots y_{i-1}$ fixed, and are to fix the value of y_i. We have two choices: either set $y_i = 0$ or set $y_i = 1$. By Lemma 13, we have two affine subspaces (or empty sets) $\mathcal{U}_{j,0}, \mathcal{U}_{j,1} \subseteq \mathcal{U}_j$ containing all inputs $x \in \mathcal{U}_j$ mapping to $y_i = 0$ and $y_i = 1$, respectively. Moreover, Lemma 13 guarantees that $|\mathcal{U}_{j,0}| + |\mathcal{U}_{j,1}| \leq 3|\mathcal{U}_j|/2$. Summing over all $0 \leq j < 2^t$, we get

$$\sum_j |\mathcal{U}_{j,0}| + \sum_j |\mathcal{U}_{j,1}| \leq \sum_j 3|\mathcal{U}_j|/2 = 3|\mathcal{U}|/2\,.$$

Let $b \in \{0,1\}$ be the value minimizing $\sum_j |\mathcal{U}_{j,b}|$. In particular, we have that $\sum_j |\mathcal{U}_{j,b}| \leq 3|\mathcal{U}|/4$. Therefore, setting $y_i = b$ reduces the size of $\mathcal{U}$ at least by a factor of $4/3$. Repeating this procedure $\log_{4/3}(2^n) + 1 \leq 3n$ times will result in a partial assignment to the output bits O with no inputs that map to it.

The algorithm SubspaceUnion maintains 2^t affine subspaces of $\mathbb{F}_2^n$, computes their dimensions and calls the deterministic polynomial-time AffineReduce procedure polynomial number of times. Therefore, this algorithm runs in time $2^t \cdot \mathrm{poly}(n) = 2^{O(n^{k-1}/m)} \cdot \mathrm{poly}(n)$. ◀

We conclude this section with a corollary stating that SubspaceUnion solves NC^0_k-Avoid efficiently for certain non-trivial values of stretch m.

▶ **Corollary 15.** *For any constants $k \geq 3$ and $\varepsilon > 0$, the algorithm SubspaceUnion solves NC^0_k-Avoid on n inputs and m outputs in deterministic polynomial and deterministic sub-exponential $2^{O(n^{1-\varepsilon})}$ time for $m \geq n^{k-1}/\log(n)$ and $m \geq n^{k-2+\varepsilon}$, respectively.*

Algorithm 2 SubspaceUnion(C).

Input: NC^0_k circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$, where $m \geq 3n^{k-2}$
Output: $y \in \{0,1\}^m$, $y \notin \text{Range}(\mathsf{C})$

```
Find x_1, ..., x_t and y_1, ..., y_{3n} via Corollary 12 for t ≤ 3n^{k-1}/m
For 0 ≤ j < 2^t, set U_j = {x ∈ {0,1}^n : Σ_{i=1}^t x_i 2^{i-1} = j}
for i=1 to 3n do
    Find function f at y_i
    For 0 ≤ j < 2^t, set U_{j,0}, U_{j,1} ← AffineReduce(U_j, f)
    Find b ∈ {0,1} minimizing Σ_j |U_{j,b}|
    Set y_i = b
    For 0 ≤ j < 2^t, set U_j = U_{j,b}
Set all remaining y_k = 0
return y
```

References

1 Josh Alman and Lijie Chen. Efficient construction of rigid matrices using an NP oracle. In *FOCS*, 2019. 3

2 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. *SIAM Journal on Computing*, 36(4):845–888, 2006. 2, 4, 7, 9

3 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. 3, 8

4 Amey Bhangale, Prahladh Harsha, Orr Paradise, and Avishay Tal. Rigid matrices from rectangular PCPs. In *FOCS*, 2020. 3

5 Aleksandr V. Chashkin. On the complexity of Boolean matrices, graphs and their corresponding Boolean functions. *Discrete Mathematics and Applications*, 4(3):229–257, 1994. 3

6 Yeyuan Chen, Yizhi Huang, Jiatu Li, and Hanlin Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *STOC*, 2023. 2

7 Joel Friedman. A note on matrix rigidity. *Combinatorica*, 13(2):235–239, 1993. 3

8 Oded Goldreich and Avishay Tal. Matrix rigidity of random Toeplitz matrices. In *STOC*, 2016. 3

9 Venkatesan Guruswami, Xin Lyu, and Xiuhan Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. In *RANDOM*, 2022. 2, 3, 4, 7, 14

10 Rahul Ilango, Jiatu Li, and Ryan Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *STOC*, 2023. 2

11 Russell Impagliazzo and Ramamohan Paturi. The complexity of k-SAT. In *CCC*, 1999. 2

12 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *FOCS*, 1998. 2

13 Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, 2000. 2, 4

14 Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, 2002. 2, 4

15 Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou. Total functions in the polynomial hierarchy. In *ITCS*, 2021. 1, 2

16 Oliver Korten. The hardest explicit construction. In *FOCS*, 2021. 2

17 Jiatu Li and Tianqi Yang. $3.1n - o(n)$ circuit lower bounds for explicit functions. In *STOC*, 2022. 3

18 Pavel Pudlák and Vojtech Rödl. Some combinatorial-algebraic problems from complexity theory. *Discrete Mathematics*, 1(136):253–279, 1994. 3

19 Hanlin Ren, Rahul Santhanam, and Zhikun Wang. On the range avoidance problem for circuits. In *FOCS*, 2022. 2, 3, 4, 9, 10

20 Claude E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28:59–98, 1949. 3

21 Mohammad A. Shokrollahi, Daniel A. Spielman, and Volker Stemann. A remark on matrix rigidity. *Information Processing Letters*, 64(6):283–285, 1997. 3

22 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, 1977. 3, 4, 8

A An Alternative Algorithm for NC^0_3-Avoid

▶ **Theorem 16.** *Given an* NC^0_3 *circuit* $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$, *where* $m \geq \frac{1}{3}\binom{n}{2} + 2n$, *the algorithm* OneSubspace *finds an element* $y \in \{0,1\}^m, y \notin \mathrm{Range}(\mathsf{C})$ *in deterministic polynomial time.*

Proof. The algorithm maintains an affine subspace $\mathcal{S} \subseteq \mathbb{F}_2^n$ over the inputs, and a partial output assignment $y \in \{0,1,*\}^m$ such that $\mathcal{S}$ contains all inputs $x \in \mathbb{F}_2^n$ consistent with y. Initially, $y = (*,\dots,*)$ and $\mathcal{S} = \mathbb{F}_2^n$. At each iteration, OneSubspace assigns at most two outputs and reduces the dimension of $\mathcal{S}$ by at least 1. After n steps, $\mathcal{S}$ must have dimension 0. Then the algorithm assigns one more output bit, and terminates with an element $y \notin \mathrm{Range}(\mathsf{C})$.

Now, we only need to argue that the algorithm can reduce the dimension of $\mathcal{S}$ in each iteration and that we can perform each step in polynomial time.

First, if there is an output y_1 that depends on at most 2 inputs x_1, x_2, let f be the function computed at that output: $y_1 = f(x_1, x_2)$. By Lemma 13, AffineReduce$(\mathcal{S}, f)$ outputs an affine subspace $\mathcal{S}_b$ of lower dimension $\dim(\mathcal{S}_b) < \dim(\mathcal{S})$, containing all inputs consistent with $y_1 = b$. Thus, in the following we assume that each output depends on exactly 3 inputs.

Since we fix at most 2 bits of the output at each iteration, the number of unassigned outputs m is always greater than $\frac{1}{3}\binom{n}{2}$. Then, there exists a pair of outputs y_1, y_2 that both depend on the same pair of inputs x_2, x_3. Since each output depends on three pairs of inputs, and the number of such pairs is $\binom{n}{2} < 3m$, there must be a pair of inputs that feeds into two outputs. Let x_1, x_4 be the remaining inputs that feed into the outputs y_1, y_2, respectively: $y_1 = f_1(x_1, x_2, x_3), y_2 = f_2(x_2, x_3, x_4)$.

There exist 4 possible values of (y_1, y_2) and at most 16 possible values of the input bits (x_1, x_2, x_3, x_4) appearing in $\mathcal{S}$. Then, there exists a pair of constants $(b_1, b_2) \in \{0,1\}^2$ such that at most 4 different assignments $A \subseteq \{0,1\}^4$ to (x_1, x_2, x_3, x_4) are consistent with the partial assignment $(y_1, y_2) = (b_1, b_2)$. Since $|A| \leq 4$, there is a 3-dimensional affine subspace in $\mathbb{F}_2^4$ that contains all points from A. Therefore, there is a hyperplane $\mathcal{H} \subseteq \mathbb{F}_2^4$ defining this 3-dimensional affine subspace. Extending $\mathcal{H}$ to all n inputs, gives us an affine subspace $\mathcal{H}' = \{x \in \mathbb{F}_2^n : (x_1, x_2, x_3, x_4) \in \mathcal{H}\} \subseteq \mathbb{F}_2^n$ that contains all inputs in $\mathbb{F}_2^n$ consistent with the partial assignment $(y_1, y_2) = (b_1, b_2)$.

If $\mathcal{S} \not\subseteq \mathcal{H}'$, then setting $(y_1, y_2) = (b_1, b_2), \mathcal{S} = \mathcal{S} \cap \mathcal{H}'$ reduces the dimension of the affine subspace $\mathcal{S}$. In the following we assume that $\mathcal{S} \subseteq \mathcal{H}'$.

- Suppose there exists $(c_1, c_2) \in \{0,1\}^2$ such that for all points (x_1, x_2, x_3, x_4) in $\mathcal{H}'$, $(f_1(x_1, x_2, x_3), f_2(x_2, x_3, x_4)) \neq (c_1, c_2)$. Then we can set $(y_1, y_2) = (c_1, c_2)$ and $\mathcal{S} = \emptyset$ as no points in $\mathcal{S} \subseteq \mathcal{H}'$ can output (c_1, c_2). In this case we found a $y \notin \mathrm{Range}(\mathsf{C})$.
- If there are no such assignments, then since $|\mathcal{H}| = 8$, there must exist an assignment $(c_1, c_2) \in \{0,1\}^2$ such that at most two points from $\mathcal{H}$ are consistent with $(y_1, y_2) = (c_1, c_2)$. These (at most) two points form a 0- or 1-dimensional affine subspace $\mathcal{U} \subseteq \mathbb{F}_2^4$, which we extend to all n inputs $\mathcal{U}' = \{x \in \mathbb{F}_2^n : (x_1, x_2, x_3, x_4) \in \mathcal{U}\} \subseteq \mathbb{F}_2^n$.

- If $\mathcal{S} \not\subseteq \mathcal{U}'$, we can set $(y_1, y_2) = (c_1, c_2)$ and $\mathcal{S} = \mathcal{S} \cap \mathcal{U}'$, reducing the dimension of $\mathcal{S}$.
- Otherwise, all inputs in $\mathcal{S} \subseteq \mathcal{U}'$ have $(y_1, y_2) = (c_1, c_2)$, and we can set $(y_1, y_2) = (1 - c_1, c_2)$ to obtain $\mathcal{S} = \emptyset$.

This algorithm performs n iterations, each of which computes dimensions of a constant number of explicitly given affine subspaces in polynomial time. ◀

Algorithm 3 OneSubspace(C).

Input: NC^0_3 circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$, where $m \geq \frac{1}{3}\binom{n}{2} + 2n$
Output: $y \in \{0,1\}^m$, $y \notin \text{Range}(\mathsf{C})$

```
Let S = F_2^n
for i=1 to n do
    if S = ∅ then
        Set all remaining y_k = 0
        return y
    if ∃y_1, x_1, x_2 s.t. y_1 = f(x_1, x_2) then
        S_0, S_1 = AffineReduce(S, f)
        Find b ∈ {0,1} that minimizes |S_b|, Set y_1 = b, S = S_b
    else
        Find y_1, y_2, x_1, x_2, x_3, x_4 s.t. y_1 = f_1(x_1, x_2, x_3), y_2 = f_2(x_2, x_3, x_4)
        Find b_1, b_2 ∈ {0,1}, s.t.
              A = {(x_1, x_2, x_3, x_4) ∈ F_2^4 : (f_1(x_1, x_2, x_3), f_2(x_2, x_3, x_4)) = (b_1, b_2)} and |A| ≤ 4
        Let H ⊆ F_2^4 be the hyperplane defined by points in A
        Let H' = {x ∈ F_2^n : (x_1, x_2, x_3, x_4) ∈ H}
        if S ⊄ H then
            Set (y_1, y_2) = (b_1, b_2), S = S ∩ H'
        else
            if ∃(c_1, c_2) s.t. ∀(x_1, x_2, x_3, x_4) ∈ H (f_1(x_1, x_2, x_3), f_2(x_2, x_3, x_4)) ≠ (c_1, c_2) then
                Set S = ∅, (y_1, y_2) = (c_1, c_2)
            else
                Find (c_1, c_2) s.t.
                      U = {(x_1, x_2, x_3, x_4) ∈ H : (f_1(x_1, x_2, x_3), f_2(x_1, x_2, x_3)) = (c_1, c_2)}, |U| ≤ 2
                Let U' = {x ∈ F_2^n | (x_1, x_2, x_3, x_4) ∈ U}
                if S ⊄ U' then
                    Set S = S ∩ U', (y_1, y_2) = (c_1, c_2)
                else
                    S = ∅, (y_1, y_2) = (1 − c_1, c_2)
return y
```