Electronic Colloquium on Computational Complexity, Report No. 140 (2024)

Almost-catalytic Computation

Sagar Bisoyi* Krishnamoothy Dinesh† Bhabya Deep Rai‡ Jayalal Sarma‡.

September 11, 2024

Abstract

Designing algorithms for space bounded models with restoration requirements on (most of) the space used by the algorithm is an important challenge posed about the catalytic computation model introduced by Buhrman *et al.* (2014). Motivated by the scenarios where we do not need to restore unless w is *useful*, we relax the restoration requirement: only when the content of the catalytic tape is $w \in A \subseteq \Sigma^*$, the catalytic Turing machine needs to restore w at the end of the computation. We define, $\mathsf{ACL}(A)$ to be the class of languages that can be accepted by almost-catalytic Turing machines with respect to A (which we call the catalytic set), that uses at most $c \log n$ work space and n^c catalytic space. We prove the following for the almost-catalytic model.

- We show that if there are almost-catalytic algorithms for a problem with catalytic set as $A \subseteq \Sigma^*$ and its complement respectively, then the problem can be solved by a zero-error randomized algorithm that runs in expected polynomial time. More formally, for any language $A \subseteq \Sigma^*$, $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) \subseteq \mathsf{ZPP}$. In particular, when $A \in \mathsf{L}$, $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) = \mathsf{CL}$. This leads to newer algorithmic approaches for designing catalytic algorithms.
- Using the above, we derive that to design catalytic algorithms for a language, it suffices to design almost-catalytic algorithms where the catalytic set is the set of strings of odd weight (PARITY). Towards this, we consider two complexity measures of the set A which are maximized for PARITY. One is the random projection complexity (denoted by $\mathcal{R}(A)$) and the other is the subcube partition complexity (denoted by $\mathcal{P}(A)$). We show that, for all $k \geq 1$, there exists a language $A_k \subseteq \Sigma^*$ such that $\mathsf{DSPACE}(n^k) \subseteq \mathsf{ACL}(A_k)$ where for every $m \geq 1$, $\mathcal{R}(A_k \cap \{0,1\}^m) \geq \frac{m}{4}$ and $\mathcal{P}(A_k \cap \{0,1\}^m) = 2^{m/4}$. This is in contrast to the catalytic machine model where it is unclear if it can accept all languages in $\mathsf{DSPACE}(\log^{1+\epsilon} n)$ for any $\epsilon > 0$.
- Improving the partition complexity of the restoration set A further, we show that for all $k \geq 1$, there exists $A_k \subseteq \{0,1\}^*$ such that $\mathsf{DSPACE}(\log^k n) \subseteq \mathsf{ACL}(A_k)$ where for every $m \geq 1$, $\mathcal{R}(A_k \cap \{0,1\}^m) \geq \frac{m}{4}$ and $\mathcal{P}(A_k \cap \{0,1\}^m) = 2^{m/4+\Omega(\log m)}$. Our main new technique for the last two items is the use of error correcting codes to design almost-catalytic algorithms.
- We also show that, even when there are more than two alphabet symbols, if the catalytic set A does not use one of the alphabet symbols, then efficient almost-catalytic algorithms with A as the catalytic set can be designed for any language in PSPACE.

*Work was done while the the author was a masters student at IIT Madras. Email: sagarbisoyi@gmail.com

†Indian Institute of Technolgy, Palakkad, India. Email: kdinesh@iitpkd.ac.in

‡Indian Institute of Technology Madras, Chennai, India. Email: {cs21d200|jayalal}@cse.iitm.ac.in, The fourth author's work is also supported by SERB-CRG Grant No : CRG/2020/003553 by Govt of India.

ISSN 1433-8092

Contents

1 Introduction

The catalytic Turing machine model (originally proposed by [BCK+14]) involves a Turing machine that is equipped with an input tape, a work tape and a special tape called the catalytic tape. Let $s, c : \mathbb{N} \longrightarrow \mathbb{N}$ be non-decreasing functions. A language L is said to be decided by a *catalytic* Turing machine M in space $s(n)$ and using catalytic space $c(n)$ if on every input x of length n and arbitrary string $w \in \{0,1\}^{c(n)}$ of length $c(n)$ written on the catalytic tape, the machine halts with w on its catalytic tape. During the computation, M uses at most $s(n)$ tape cells on the work tape and $c(n)$ cells on its catalytic tape, and M correctly outputs whether $x \in L$. CL is the class of languages that can be accepted by catalytic Turing machines that use at most $O(\log n)$ work space, and $O(n^c)$ catalytic space.

In addition to its theoretical appeal, the motivation for this model (c.f. [BCK+14, BKLS18]) also comes from practically relevant contexts - where the memory that algorithms need is all used up to store otherwise useful data. In such situations, catalytic algorithms (and more formally, catalytic Turing machines) that guarantee restoration of the content of their catalytic tape to the original content, are arguably useful.

A natural question is whether this extra space (which needs to be restored to its original content at the end of the computation) helps at all. Quite surprisingly, [BCK+14] showed that L-uniform $\mathsf{TC}^1 \subseteq \mathsf{CL}$. The fact that $\mathsf{NL} \subseteq \mathsf{TC}^1$ makes this immediately surprising for a space complexity theorist, because it implies that the directed graph reachability problem has a deterministic algorithm in the above model that uses $O(\log n)$ space in the worktape and at most $\mathsf{poly}(n)$ space in its catalytic tape.

[BCK+14] also showed that CL is contained in ZPP. The main observation that leads to this upper bound is that two computations starting with different initial catalytic tape contents, say w and

w' cannot reach the same configuration at any point in their computations on the same input. In a subsequent work, [BKLS18] explore the power of non-determinism in catalytic space. CNL is the class of problems solvable by non-deterministic logspace catalytic Turing machines. Using similar ideas from [BCK+14], it was shown [BKLS18, DGJ+20] that the ZPP upper-bound holds even for non-deterministic and randomized variants of catalytic logspace classes. [BKLS18] showed that under a plausible hardness assumption, CNL = coCNL. In a work by [DGJ+20], it is shown that under the same hardness assumption and using very similar techniques, CBPL = CSL = CSC^1 = CL. Recently, [CLMP24] completely removed the need for any hardness assumption and showed that CBPL = CL unconditionally. Here CBPL and CSL are sets of languages solvable by logspace randomized and symmetric catalytic Turing machines, respectively. CSC^1 denotes the set of languages solved by catalytic log-space machines that run in polynomial time. [GJST19] showed that under the same hardness assumption CNL = CUL. For more details, the reader is referred to the following surveys: [Kou16, Mer23]. Algorithmic techniques that were used to design catalytic algorithms has also been proven helpful in designing non-trivial space efficient algorithms for the Tree evaluation problem (proposed in [CMW+12]). For more details, see [CM24] and the references therein.

Our Results: Motivated by the scenarios where we do not need to restore unless w is *useful*, we relax the restoration requirement: only when the content of the catalytic tape is $w \in A \subseteq \Sigma^*$, the catalytic Turing machine needs to restore w at the end of the computation. Indeed, $A \subseteq \Sigma^*$ represents the set of "useful" w's. We call such Turing machines as *almost-catalytic Turing machines* and the languages accepted by such machines, using logarithmic work space and polynomial catalytic space as $\mathsf{ACL}(A)$ (See Section 2 for a formal definition). We call the set A to be the *catalytic set*.

Thus, the major challenge in this context is to design algorithms for useful catalytic sets. We first consider two ways of exploring the almost-catalytic in terms of the catalytic set A. Firstly in terms of the cardinality of A and secondly in terms of the complexity of A. To start with, observe that $\forall A \subseteq \Sigma^*$, $\mathsf{ACL}(A) \subseteq$ PSPACE. In addition, it is easy to observe that $\mathsf{ACL}(\Sigma^*) =$ CL and $\mathsf{ACL}(\emptyset) =$ PSPACE. Given this, one natural way to work towards catalytic logspace algorithms for PSPACE from almost-catalytic algorithms is to parameterize based on the size of A. Defining $f(n) = |A \cap \{0,1\}^n|$ to be a measure of sparsity of A, we are interested to see how close can the function $f(n)$ be to 2^n, such that we have almost-catalytic algorithms for every language in PSPACE.

In this direction, it is easy to see that if A is a tally set ($A \subseteq \{1\}^*$), then PSPACE = $\mathsf{ACL}(A)$. Such a consequence is unclear if A is only known to be polynomially sparse. However, if A is polynomially sparse with low space complexity, then, we can simulate the whole of PSPACE using almost-catalytic Turing machines. That is, for any sparse set $A \in$ L, $\mathsf{ACL}(A) =$ PSPACE (see Proposition 3.4). Indeed, it is more challenging to design $\mathsf{ACL}(A)$ algorithms for every language in PSPACE when A is large in size.

However, we note that there is a set A with exponential density for which we can design almost-catalytic algorithms to accept any language in $\mathsf{DSPACE}(n^k)$ (see Proposition 3.3). This implies that $|A \cap \{0,1\}^n|$ is not a good parameter to measure our progress towards designing catalytic algorithms by this approach.

To make further progress, we turn to the structural front. We show a limitation of the almost-

catalytic Turing machines with respect to A by showing the following upper bound.

Theorem 1.1. *For any $A \subseteq \Sigma^*$, $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) \subseteq \mathsf{ZPP}$. If $A \in \mathsf{L}$ then $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) = \mathsf{CL}$.*

The first part of the above theorem and the argument is a generalization of the idea in [BCK+14] which shows $\mathsf{CL} \subseteq \mathsf{ZPP}$. In particular, when $A = \Sigma^*$ or $A = \emptyset$, we recover their result. We remark that this generalization is different from the compress-or-random method that appear in [CLMP24, Pyn24]. The second part of the above theorem can also be viewed as a method of obtaining catalytic algorithms by designing almost-catalytic algorithms with respect to an appropriate set A. We also remark that, unlike the arguments in [BCK+14], the Theorem 1.1 or the proof of it, does not imply for any almost-catalytic Turing machine runs in expected polynomial time.

A notable example of such a set is the language PARITY consisting of strings over $\{0,1\}^*$ with an odd number of ones. Indeed, PARITY $\in \mathsf{L}$. However, if we have an almost-catalytic algorithm for a language L with the catalytic set being PARITY, then there is an almost-catalytic algorithm with respect to $\overline{\text{PARITY}}$ as well (See Proposition 3.5). Hence, $\mathsf{ACL}(\text{PARITY}) = \mathsf{CL}$. Thus, it suffices to design almost-catalytic algorithms with respect to PARITY and we set this as the target.

To measure our progress towards the set PARITY, we define two measures for the set A, defined below, which are maximised for parity.

Random Projection Complexity: For an $A \subseteq \{0,1\}^m$, we define, for an $\epsilon \geq 0$, the random projection complexity, $\mathcal{R}_\epsilon(A)$ as the largest $\ell \geq 0$ such that: $\Pr_{\substack{T \subseteq [m] \\ |T|=\ell}} \left[|A_T| \geq 2^{\ell-1} \right] \geq 1 - \epsilon$ where A_T denotes the set of strings in A projected to the indices in T. Observe that $\mathcal{R}_0(\text{PARITY}_m) = m - 1$. Thus, in order to approach $A = \text{PARITY}$, we will design almost-catalytic computation with respect to set A, where $\mathcal{R}_\epsilon(A)$ is as large as possible where ϵ is close to 0, say $2^{-\alpha m}$ for some small constant $0 \leq \alpha < 1$. In this case, we use $\mathcal{R}(A)$ to denote $\mathcal{R}_{2^{-\alpha m}}(A)$.

Subcube Partition Complexity: A subcube C of the cube $\{0,1\}^m$ is given by a mapping (partial assignment) $\alpha : [n] \to \{0,1,*\}$ and is defined to be the set of all vectors in the Boolean hypercube on n bits, B_n, that agree with α on coordinates that are assigned a non-$*$ value by α. More precisely the subcube C_α is the set $\{x \in \{0,1\}^m : \alpha(i) \neq * \implies x_i = \alpha(i)\}$. For a set A, a partition $C = \{C_1, \ldots, C_t\}$ of A into subcubes C_i such that $C_i \subseteq A$ is called a subcube partition of A. We denote by $\mathcal{P}(A)$ the minimum number of subcubes in a subcube partition of A. Observe that $\mathcal{P}(\text{PARITY}_m) = 2^{m-1}$. Thus, in order to approach $A = \text{PARITY}$, we propose to design almost-catalytic algorithms for sets with high partition complexity.

We remark about the choice of the above two measures in our journey towards achieving PARITY as our catalytic set. As noted earlier, there are specific catalytic sets (see Proposition 3.3) $A \subseteq \{0,1\}^*$ which are of exponential density for which $\mathsf{PSPACE} \subseteq \mathsf{ACL}(A)$. However, it can be shown (see Proposition 2.1) that this catalytic set A has a subcube partition complexity $\mathcal{P}(A)$ of 1 and small random projection complexity $\mathcal{R}(A)$. Hence, it is natural to look for other catalytic sets A such that $\mathsf{ACL}(A)$ is powerful enough to simulate polynomial space bounded computation, and which possess larger values for one or both of these measures.

As our next result, we show the following simulation of $\mathsf{DSPACE}(n^k)$ almost-catalytically for set A with a large $\mathcal{P}(A)$ and $\mathcal{R}(A)$.

Theorem 1.2. *For all $k \geq 1$, there exists a language $A_k \subseteq \Sigma^*$ such that* $\mathsf{DSPACE}(n^k) \subseteq \mathsf{ACL}(A_k)$ *where for every $m \geq 1$, $\mathcal{R}(A_k \cap \{0,1\}^m) \geq \frac{m}{4}$ and $\mathcal{P}(A_k \cap \{0,1\}^m) = 2^{m/4}$.*

When we need to simulate only polylogarithmic space, the partition complexity of the catalytic set A for which we restore can be improved. We prove the following theorem in this direction:

Theorem 1.3. *For all $k \geq 1$, there exists $A_k \subseteq \{0,1\}^*$ such that* $\mathsf{DSPACE}(\log^k n) \subseteq \mathsf{ACL}(A_k)$ *where for every $m \geq 1$, $\mathcal{R}(A_k \cap \{0,1\}^m) \geq \frac{m}{4}$ and $\mathcal{P}(A_k \cap \{0,1\}^m) = 2^{m/4+\Omega(\log m)}$.*

We remark that this is in contrast to the catalytic machine model where it is unclear if it can accept all languages in $\mathsf{DSPACE}(\log^{1+\epsilon} n)$ for any $\epsilon > 0$. In addition, note that, if Theorem 1.3 holds when $A_k \cap \{0,1\}^m$ covers the whole of $\{0,1\}^m$, then it would imply that $\mathsf{DSPACE}(\log^k n) \subseteq \mathsf{CL}$. Since $\mathsf{CL} \subseteq \mathsf{ZPP}$ [BCK+14], this would show that $\mathsf{DSPACE}(\log^k n) \subseteq \mathsf{ZPP}$, which in-turn would separate L and ZPP.

Exploring the power of additional alphabets, we show that even if the catalytic tape alphabet has even a single symbol that is not included in the alphabet for the catalytic set (irrespective of the size), the almost-catalytic machine can simulate the whole of PSPACE (See Proposition 3.6).

Our Techniques: Our technique starts with a novel approach towards designing almost-catalytic algorithms using codes that can be decoded space efficiently. At a high level, the idea is as follows: let us say we want to design a catalytic Turing machine accepting a language L which has a Turing machine that runs in space $c(n)$. For the catalytic Turing machine, the given content of the catalytic tape can be treated as the codeword (for a fixed code), the Turing machine proceeds to modify the content of the tape according to its computational needs. The modification of the work tape during computation can be seen as introducing "errors" to the codeword. Finally we use the decoding algorithm to correct the "errors" and finally obtain the original codeword we started with, thus achieving the restoration condition.

Indeed, there are a number of challenges in implementing the above plan. The first limitation is that the number of bits modified to the initial string on the catalytic tape must be such that the modified string (after computation) is still within a decodable distance from the original word. Thus, if $c(n)$ is the catalytic space available, we can hope to allow the catalytic TM to use only strictly $o(c(n))$ bits in the catalytic tape during the computation. Thus, an interesting target is to simulate normal Turing machines that use an asymptotically smaller amount of space.

A second challenge is that the code must be decodable in deterministic logarithmic space, as we have only so much work space. Fortunately, there are codes that have constant rate and constant relative distance, for which logspace decoding algorithms are known (See Theorem 19 in [Spi97] and Theorem 14 in [GK06]). Using additional decodability properties of Spielman codes, we show that the set A can be expanded to achieve larger random projection complexity and larger subcube partition complexity, thus progressing towards $A = \textsc{Parity}$. In order to establish the progress in terms of measures of the catalytic sets, we also employ techniques from basic combinatorics of codes, and Fourier analysis of Boolean functions to estimate the partition complexity of the catalytic set in our algorithms.

2 Preliminaries

We begin by defining catalytic computation as described by [BCK+14]. We refer the reader to standard references [Gol08, AB09] for definitions of the complexity classes not defined in this paper.

A catalytic Turing machine is a Turing machine with a read-only input tape, a work tape of size $s(n)$, and a catalytic tape of size $c(n)$ initially containing some $w \in \{0,1\}^{c(n)}$, where n is the size of the input. The machine M is said to decide a language L if (1) $x \in L$ if and only if M accepts on input x for all possible initial catalytic content w and (2) For each input $x \in \{0,1\}^n$ and any initial catalytic tape content w, M halts with w on its catalytic tape. We shall use the term *catalytic space* to denote the space in the catalytic tape. The class $\mathsf{CSPACE}(s(n), c(n))$ is the set of all languages decided by a catalytic Turing machine with work space $s(n)$ and catalytic space $c(n)$. The class CL denotes $\mathsf{CSPACE}(O(\log n), \mathsf{poly}(n))$.

2.1 Bounds on the Complexity Measures

Recall, from the introduction, that for a set $A \subseteq \{0,1\}^m$, we use $\mathcal{R}(A)$ to denote its Random projection complexity and $\mathcal{P}(A)$ to denote its Subcube partition complexity. For an integer b dividing m, let $A_b = \{w \mid w \text{ is of the form } 0^{m/b}(0+1)^{m-m/b}\} \subseteq \{0,1\}^m$.

Proposition 2.1. *For the set A_b defined above, $\mathcal{P}(A_b) = 1$ and for a constant b, $\mathcal{R}(A_b) = O(1)$.*

Proof. The subcube partition complexity $\mathcal{P}(A_b)$ is 1 as the entire set is contained in the subcube C with the first m/b coordinates set to 0 and it cannot be any smaller.

We need an upper bound on the random projection complexity $\mathcal{R}(A_b)$. Towards this, let ℓ be the smallest value for which $\Pr_{\substack{T \subseteq [m] \\ |T|=\ell}}\left[|(A_b)_T| \geq 2^{\ell-1}\right] < 1 - \epsilon$, where ϵ is a small constant. It can seen that this probability is lower bounded by α^ℓ for a constant α that depends on b. Hence for a constant b, ℓ is bounded by a constant. □

We now establish a lower bound for the measure for another set A which we use as a catalytic set later. We quickly recall linear codes, and related parameters below.

A linear code over a q-ary alphabet of length m and dimension k is a linear subspace C with dimension k of the vector space $\mathbb{F}_q^m$. The distance d of a linear code C is the minimum Hamming distance between any two codewords in C, where Hamming distance between two codewords is the number of locations where they differ. Furthermore, C is said to be an $[m,k,d]_q$ code if it has length m, dimension k, distance d and alphabet size q. The relative distance of a $[m,k,d]_q$ code, δ is defined as $\delta = \frac{d}{m}$. The covering radius of a code C is the minimum D such that for all $w \in \mathbb{F}_q^m$ there exists a codeword $c \in C$ such that $d(c,w) \leq D$. We will have the following bounds on the measure.

Proposition 2.2. *If A is a set of codewords for an $[m,k,\delta m]_2$ code with δ being a constant, then $\mathcal{R}_\epsilon(A) \geq k$ for $\epsilon = 2^{-2k}$.*

The proof of the above Lemma is a standard application of codes. We reproduce the argument in Appendix A.1 for completeness.

2.2 Lower Bounds on the Measures for Union of Hypercubes

We identify $\mathbb{F}_2^m$ with $\{0,1\}^m$. For two strings, $x, y \in \{0,1\}^m$, the Hamming distance, is denoted by $\Delta(x,y)$. The same definition can be extended to subsets as follows: for any $A, B \subseteq \{0,1\}^m$, $\Delta(A,B) = \min\{\Delta(a,b) \mid a \in A, b \in B\}$.

A set $H \subseteq \{0,1\}^m$ is said to be a Hamming ball if and only if there exists a $k \geq 0$ and a $z \in \{0,1\}^n$ such that for every $h \in H$, $\Delta(h,z) \leq k$. We call k as the *radius* of the Hamming ball H and z to be its *center*.

Proposition 2.3. *Let $A \subseteq \Sigma^*$ be such that for any $m \geq 1$, $A_m := A \cap \{0,1\}^m$ can be expressed as a union of Hamming balls $H_1, H_2, \ldots, H_t$ over $\{0,1\}^m$ such that for any $i \neq j$, $\Delta(H_i, H_j) > 1$. Then, $\mathcal{P}(A_m) = \sum_{i=1}^{t} \mathcal{P}(H_i)$.*

Proof. Consider any partition of A_m into t subcubes given by $C_1, C_2, \ldots, C_t$. Suppose that there exists a subcube C_k, such that it contains points from H_i and H_j for some $i \neq j$. As it is a partition, every point in the subcube must belong to some Hamming ball and hence there exists two strings $x, y \in C_k$ that differ in one bit with $x \in H_i$ and $y \in H_j$. But this contradicts $\Delta(H_i, H_j) > 1$. Hence, each C_i can contain at most one Hamming ball. With no sub cube partition of A_m intersecting two Hamming balls, we conclude $\mathcal{P}(A_m) \geq \sum_{i=1}^{t} \mathcal{P}(H_i)$.

On the other hand, since a sub cube partition of the Hamming balls gives a sub cube partition of A_m, $\mathcal{P}(A_m) \leq \sum_{i=1}^{t} \mathcal{P}(H_i)$. □

We now claim that any Hamming ball over $\{0,1\}^m$ with radius strictly less than $m/2$ centered at 0^m, must have a subcube partition complexity of $\Omega(\sqrt{m})$.

Proposition 2.4. *Let H be a Hamming ball over $\{0,1\}^m$ of radius $k < m/2$ centered at 0^m. Then $\mathcal{P}(H) = \Omega(\sqrt{m})$.*

Proof. Define the Boolean function $Th_{m,k} \colon \{0,1\}^n \to \{-1,1\}$ as for any $x \in \{0,1\}^m, Th_{m,k}(x) = -1$ if and only if $|x| \leq k$. We start with the observation that a Hamming ball of radius k centered at 0^m are precisely the set of inputs on which the threshold Boolean function $Th_{m,k}$ evaluates to -1.

It is known that the partition complexity of a Boolean function f on m bits is lower bounded by $\sum_{S \subseteq [m]} |\widehat{f}(S)|$ where $\widehat{f}(S)$ is the Fourier coefficient of f (cf. Lemma 3.8 of [CKLS16]). We show that this quantity grows about linear in $\sqrt{m}$.

Towards arguing this, we use a known result due to Gotsman and Linial [GL94] on the spectral properties of threshold functions.

Proposition 2.5 ([GL94]). *For any threshold function f on m bits, $\widehat{f}(\emptyset)^2 + \sum_{i \in [m]} \widehat{f}(\{i\})^2 \geq \frac{1}{2}$.*

Since $f = Th_{m,k}$ is a symmetric function, the value of $\widehat{f}(S)$ depends only on $|S|$. Moreover, for the choice of k, $|\widehat{f}(\emptyset)| < 1/2$. Hence, by Proposition 2.5, for any $i \in [m]$, $m \cdot \widehat{f}(\{i\})^2 \geq 1/4$. This implies that

$$\mathcal{P}(H) \geq \sum_{S} |\widehat{f}(S)| \geq \sum_{i \in [m]} |\widehat{f}(\{i\})| = \Omega(\sqrt{m})$$

□

The main lemma that we will need in later sections is the following:

Lemma 2.6. *Let $A \subseteq \Sigma^*$ such that for every $m \geq 1$, A_m is a disjoint union of Hamming balls $H_1, \ldots, H_t$ of radius $k < m/2$ over $\{0,1\}^m$ such that for every $i, j \in [t]$, $\Delta(H_i, H_j) > 1$. Then for every $m \geq 1$, $\mathcal{P}(A_m) = \Omega(t\sqrt{m})$.*

Proof. For a contradiction, suppose that $\mathcal{P}(A_m) = o(t\sqrt{m})$. By Proposition 2.3, which says $\mathcal{P}(A_m) = \sum_{i=1}^{t} \mathcal{P}(H_i)$, there exists an H_i centered at some $z \in \{0,1\}^m$ such that $\mathcal{P}(H_i) = o(\sqrt{m})$. Consider the set $H' := H_i \oplus z = \{h \oplus z \mid h \in H_i\}$ obtained by taking the bitwise XOR of each string in H_i by z. Observe that H' is a Hamming ball centered at 0^m of same radius as that of H_i since the operation performed does not alter the relative Hamming distance between the points. With the subcubes shifted by z also forming a partition of H', we have $\mathcal{P}(H') = o(\sqrt{m})$ which contradicts Proposition 2.4. This completes the proof. □

3 Almost-catalytic Turing Machines

In this section, we present the definition and our results on Almost-catalytic Turing machines. We begin with the following definition.

Definition 3.1 (Almost-catalytic Computation with respect to A : $\mathsf{ACSPACE}_A$ and $\mathsf{ACL}(A)$). Let $A \subseteq \Sigma^*$, a language L is said to be in the class $\mathsf{ACSPACE}_A(s(n), c(n))$ if there is a Turing machine M which on inputs of length n uses a work tape of size $s(n)$ and catalytic tape of size $c(n)$ (over an alphabet set of size 2) such that, (1) for all $x \in \Sigma^*$, $x \in L$ if and only if the Turing machine M accepts x. (2) for all $w \in A$, if the machine M starts the computation with content of the catalytic tape as w, then at the end of the computation w will be restored back in the tape. Furthermore we define $\mathsf{ACL}(A)$ to denote the class $\mathsf{ACSPACE}_A(O(\log n), O(n^c))$ for some constant c.

We make some preliminary observations about almost-catalytic computation. Indeed, by definition, $\mathsf{CL} = \mathsf{ACL}(\Sigma^*)$, and $\mathsf{PSPACE} = \mathsf{ACL}(\emptyset)$. In general, for any $A \subseteq \Sigma^*$, $\mathsf{CL} \subseteq \mathsf{ACL}(A) \subseteq \mathsf{PSPACE}$. Moreover, there are languages $A \subsetneq \Sigma^*$ for which the $\mathsf{ACL}(A)$ can simulate the whole of PSPACE. The following proposition is also easy to see.

Proposition 3.2. *If $A = \{1^n \mid n \geq 0\}$, then $\mathsf{PSPACE} = \mathsf{ACL}(A)$*

The above proposition is true since the catalytic tape can be filled with 1^n at the end of the computation irrespective of the original content. However, it is a challenge to show the above for an arbitrary singleton set A.

A natural question is about the density of the catalytic set. We establish (see Appendix A.2 for a proof) that for every k, there are sets with high density with respect to which every language in $\mathsf{DSPACE}(n^k)$ admits almost-catalytic algorithms.

Proposition 3.3. *For any $k \geq 1$, there exists a language $A \subseteq \{0,1\}^*$ with $\mathsf{DSPACE}(n^k) \subseteq \mathsf{ACL}(A)$ via an almost-catalytic logspace machine using $m = bn^k$ catalytic space for some constant $b \geq 1$, such that for any $m \geq 1$, $|A \cap \{0,1\}^m| \geq 2^{m - m/b}$.*

At the other extreme, if $|A \cap \{0,1\}^n| = \mathsf{poly}(n)$ i.e. A is sparse, we ask the question if it is true that for all sparse A, $\mathsf{ACL}(A) = \mathsf{PSPACE}$? We observe that the answer is affirmative when the sparse set A under consideration is in L.

Proposition 3.4. *Let $A \subseteq \Sigma^*$ be a language in* L*. Then if A is sparse then* $\mathsf{ACL}(A) = \mathsf{PSPACE}$.

The proof for the above theorem can be found in Appendix A.3. We now show the following proposition for PARITY which is logspace decidable but is not sparse.

Proposition 3.5. $\mathsf{ACL}(\text{PARITY}) = \mathsf{ACL}(\overline{\text{PARITY}})$.

Along with Theorem 1.1, this shows that it suffices to design almost-catalytic logspace algorithms for $A = \text{PARITY}$ to show membership in CL.

Proof. It suffices to show that if $L \in \mathsf{ACL}(\text{PARITY})$, then $L \in \mathsf{ACL}(\overline{\text{PARITY}})$. Let $L \in \mathsf{ACL}(\text{PARITY})$ via an almost-catalytic machine M. Consider an almost-catalytic machine M' which works by first checking if the catalytic content w belongs to $\overline{\text{PARITY}}$. If yes, it flips the first bit of the catalytic content (which makes the catalytic content to be in PARITY), runs M, flips the first bit of catalytic tape and accepts iff M accepts x. The simulation of M will correctly decide L and restore the catalytic content which is the same as w except for the first bit. The final step of M' will restore the first bit. Hence M' restores all strings in $\overline{\text{PARITY}}$ and accepts L. □

It is important that the definition of almost-catalytic space (Definition 3.1) use a catalytic tape alphabet set of size 2. A larger alphabet set can dramatically increase the power of almost-catalytic space. Suppose we let the almost-catalytic machine with catalytic alphabet over a larger Γ with $\{0,1\} \subsetneq \Gamma$ and make the machine restore any set $A \subseteq \{0,1\}^*$. More precisely, let $\mathsf{ACL}^{\Gamma}(A)$ denote the languages decidable by almost-catalytic logspace machines working over the catalytic tape alphabet Γ with $A \subseteq \Gamma^*$ as the catalytic set. Observe that for any $A \subseteq \Gamma^*$, $\mathsf{ACL}^{\Gamma}(A) \subseteq \mathsf{PSPACE}$.

We now show that even if the catalytic tape alphabet has even a single symbol that is not included in the alphabet for the catalytic set, the almost-catalytic machine can simulate the whole of PSPACE.

Proposition 3.6. *Let Σ be an input alphabet set and Γ be a catalytic tape alphabet with $|\Gamma| > |\Sigma|$. Then for any $A \subseteq \Sigma^*$,* $\mathsf{PSPACE} = \mathsf{ACL}^{\Gamma}(A)$*. In particular, for $\Sigma = \{0,1\}$ and any Γ with $|\Gamma| \geq 3$,* $\mathsf{PSPACE} = \mathsf{ACL}^{\Gamma}(\Sigma^*)$

Proof. Without loss of generality, assume $\Sigma = \{0,1\}$ by suitably fixing a binary encoding for the input alphabets. Let $A \subseteq \Sigma^*$. It suffices to show that $\mathsf{PSPACE} \subseteq \mathsf{ACL}^{\Gamma}(A)$.

Consider a language L in PSPACE via a $p(n)$ space bounded deterministic Turing machine M where $p(n)$ is a fixed polynomial in n. Also, without loss of generality, let the work tape of M use the alphabet set $\{0, 1, \sqcup\}$.

An almost-catalytic machine M' using catalytic tape alphabet Γ having $\{0, 1, \widehat{0}\} \subseteq \Gamma$ accepting L with a catalytic tape of length $4p(n)$ is described as follows: Scan across the catalytic tape and check if the initial catalytic content w contains a $\widehat{0}$ symbol. If there is no occurrence of $\widehat{0}$, then w can be a member of A and in particular consists of 1s and 0s alone. Using the work tape, M' counts the number of 0s in w denoted by m.

We now describe how M' simulates M. Suppose that the number of 0s is more than the number of 1s. Then $m \geq \frac{1}{2} \times 4p(n) = 2p(n)$. The machine M' uses the first $2p(n)$ cells out of the m cells containing 0 of the catalytic tape to simulate the workspace of M. Note that M is over alphabet set $\{0, 1, \sqcup\}$ while the catalytic tape of M' is over the alphabet set Γ. To handle the work tape symbols

of M correctly during the simulation, M' uses the following encoding $E : \{0, 1, \sqcup\} \to \Gamma$ defined as $E(0) = 00$, $E(1) = 0\widehat{0}$ and $E(\sqcup) = \widehat{0}0$. More precisely, if M reads (or writes) a symbol $\alpha \in \{0, 1, \sqcup\}$ at position i of its tape, M' proceeds to read (or write) $E(\alpha)$ at the $2i$ and $2i + 1$th cells having 0 or $\widehat{0}$ counted from the left end on the catalytic tape. Once the computation ends, restoration of w is achieved (irrespective of whether $w \in A$ or not) by replacing all the $\widehat{0}$ with 0 at the end of the simulation. Now, if the number of 1's are more than the number of 0s, then M' uses an encoding $E(0) = 11$, $E(1) = 1\widehat{0}$ and $E(\sqcup) = \widehat{0}1$ and repeat the above simulation of M with 0 replaced by 1 in the above text.

If w contains a $\widehat{0}$, then $w \notin A$ and therefore M' is not required to restore the catalytic tape. In such a case, M' erases the catalytic tape and simulates M on it. Clearly, if M uses $\mathsf{poly}(n)$ workspace, M' can simulate M using $O(\log n)$ work space and $4 \cdot \mathsf{poly}(n)$ catalytic space. Hence $L \in \mathsf{ACL}^{\Gamma}(A)$ which completes the proof. □

4 An Upper Bound on Almost-catalytic Computation

In this section, we show that for any language $A \subseteq \Sigma^*$, languages computable by almost-catalytic Turing machines with respect to A which are also computable by catalytic Turing machines with respect to $\overline{A}$, both using are contained in ZPP.

Lemma 4.1. *Define for any almost-catalytic Turing machine M restricted to A, $C_t(x, w)$ to be the configuration with input x and catalytic tape content w at time t. For all x, for all $w, w' \in A$ such that $w \neq w'$ and for all t, t' $C_t(x, w) \neq C_{t'}(x, w')$.*

Proof. Assume there exists an x and there exists $w, w' \in A$ such that $C_t(x, w) = C_{t'}(x, w')$. Now, from that point onward the computation would be the same and the restoration part would be incorrect for one of w or w', a contradiction. This justifies our lemma. □

Theorem 1.1. *For any $A \subseteq \Sigma^*$, $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) \subseteq \mathsf{ZPP}$. If $A \in \mathsf{L}$ then $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) = \mathsf{CL}$.*

Proof. First, we argue that for any $A \subseteq \Sigma^*$, $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) \subseteq \mathsf{ZPP}$. Let $L \in \mathsf{ACL}(A)$ via Turing machine M_1 and $L \in \mathsf{ACL}(\overline{A})$ via Turing machine M_2. Algorithm 1 describes the ZPP machine M' for L.

Algorithm 1 Description for Machine M' on input x and initial catalytic tape content w

1: Choose a $w \in \{0, 1\}^{\mathsf{poly}(n)}$ u.a.r.
2: Perform steps (3) and (4) in a time shared fashion till one of them halts
3: Run M_1 on x with w on a catalytic tape
4: Run M_2 on x with w on a separate catalytic tape
5: Accept if and only if the machine that halted accepted.

Correctness follows since M' either simulates M_1 or M_2 both of which correctly accepts L.

We now analyze the run time of M' and show that it runs in expected polynomial time (w.r.t. w). Let $t(x, w)$ denote the total number of steps M' makes on x and w. Let $t_1(x, w)$ denote the running time of machine M_1 on input w in Step 4. Let $t_2(x, w)$ denote the number of steps taken in Step 6. Observe that $t(x, w) = O(\min\{t_1(x, w), t_2(x, w)\})$. For a fixed x, the expected running time

(over the random choices of w) of M' can be obtained as $\mathbb{E}[t(x,w)] = \mathbb{E}[\min\{t_1(x,w), t_2(x,w)]$. We now bound the expectation.

$$\begin{aligned}
\mathbb{E}[t(x,w)] &= \mathbb{E}[t(x,w)|w \in A] \times \Pr[w \in A] + \mathbb{E}[t(x,w)|w \in \overline{A}] \times \Pr[w \in \overline{A}] \\
&\leq \mathbb{E}[t_1(x,w)|w \in A] \times \Pr[w \in A] + \mathbb{E}[t_2(x,w)|w \in \overline{A}] \times \Pr[w \in \overline{A}] \qquad (1) \\
&\leq \frac{\sum_{w \in A} t_1(x,w)}{|A|} \times \frac{|A|}{2^{|w|}} + \frac{\sum_{w \in \overline{A}} t_2(x,w)}{|\overline{A}|} \times \frac{|\overline{A}|}{2^{|w|}} \\
&\leq \frac{2^{|w|} \times n^c}{|A|} \times \frac{|A|}{2^{|w|}} + \frac{2^{|w|} \times n^c}{|\overline{A}|} \times \frac{|\overline{A}|}{2^{|w|}} = O(n^c) \qquad (2)
\end{aligned}$$

Note that Eq. 1 follows as $t(x,w)$ is the minimum among $t_1(x,w)$ and $t_2(x,w)$ and Eq. 2 follows from Lemma 4.1 where c is some absolute constant. Thus it follows that $\mathbb{E}[t(x,w)] \leq \mathsf{poly}(n)$. Hence, the overall running time of M' will be polynomial on expectation.

We now argue that for any $A \in \mathsf{L}$, $\mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A}) = \mathsf{CL}$.

For any $L \in \mathsf{CL}$, $L \in \mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A})$ as any catalytic machine always restores the catalytic content (irrespective of the choice of A). On the other hand, suppose that $L \in \mathsf{ACL}(A) \cap \mathsf{ACL}(\overline{A})$ via an almost-catalytic machine M_1 with restoration for A and via an almost logspace catalytic machine M_2 with restoration for $\overline{A}$. Since A can be decided in logspace, the catalytic algorithm first checks if the catalytic content belongs to A and runs M_1 and runs M_2 otherwise. The resulting machine is indeed catalytic as it restores irrespective of the catalytic content and uses only logarithmic work space. Hence $L \in \mathsf{CL}$. □

5 Almost-catalytic Computation via Error Correcting Codes

We observed for any $A \subseteq \Sigma^*$, $\mathsf{ACL}(A) \subseteq \mathsf{PSPACE}$. We now show that there exists $A \subseteq \Sigma^*$ such that $\mathsf{PSPACE} \subseteq \mathsf{ACL}(A)$. We prove this by showing that there exists an $A \subseteq \Sigma^*$ such that for any k and for any $L \in \mathsf{DSPACE}(n^k)$, $L \in \mathsf{ACL}(A)$. This suffices since $\mathsf{PSPACE} = \cup_{k \geq 0} \mathsf{DSPACE}(n^k) \subseteq \mathsf{ACL}(A)$.

Our intuition is the following : any computation can be seen as "corrupting" the catalytic tape content making the restoration difficult. With this view, it is natural to set A to be codewords from an error correcting code of good distance. In addition, the code should be decodable in $O(\log n)$ space. In the following Theorem, we choose A to be one such code.

Theorem 1.2. *For all $k \geq 1$, there exists a language $A_k \subseteq \Sigma^*$ such that $\mathsf{DSPACE}(n^k) \subseteq \mathsf{ACL}(A_k)$ where for every $m \geq 1$, $\mathcal{R}(A_k \cap \{0,1\}^m) \geq \frac{m}{4}$ and $\mathcal{P}(A_k \cap \{0,1\}^m) = 2^{m/4}$.*

Proof. Fix any $k \geq 0$. Let $L \in \mathsf{DSPACE}(n^k)$ via a Turing machine M using a work space of cn^k for some constant $c > 0$. The goal is to construct a catalytic logspace Turing machine M' such that $L(M') = L$ and it always restores the catalytic content w if $w \in A$. We choose our A such that $A_n := A \cap \{0,1\}^n$ consists of codewords of an explicit $[n, \frac{n}{4}, \alpha n]_2$ linear code constructed by Spielman [Spi97]. Here, α (a constant, independent of n) is the relative distance of the code (as described in Theorem 19 of [Spi97]). In their work, it was shown that these codes can be decoded in deterministic logspace. Let D be such a logspace decoding machine.

We now describe a machine M' (shown in Algorithm 2) accepting L. With $x \in \Sigma^*$ of length n, let the length of the catalytic tape be bn^k where b is at least $\frac{2c}{\alpha}$. We work with the set A_{bn^k} (where

A_n is as defined above). Let D be the logspace decoder, given access to a string of length bn^k, can correct it using $O(\log n)$ space provided the string is within the decoding limit of some codeword in A_{bn^k}.

Algorithm 2 Description of M' on input x and initial catalytic tape content w

1: On input x, run M on x using the first cn^k cells of the catalytic tape as the work tape for M.
2: Using the work tape as the work space for D, decode the content of the catalytic tape.
3: Accept if M accepted x
4: Else reject

Since M' simulates M, $L(M') = L(M) = L$. Let w be the initial content of the catalytic tape with $|w| = bn^k$. Let w' be its content at the end of the computation in Step 1 of M'. Observe that w and w' can differ in at most cn^k bits as M uses only the first cn^k bits of the catalytic tape. If $w \in A_{bn^k}$, then w is a codeword and w' must fall in the Hamming ball of radius $\frac{\alpha}{2}bn^k$ since $\Delta(w', w) \leq cn^k \leq \frac{\alpha}{2}bn^k$ by the choice of b. Hence upon running D on w' in Step 2 of M', will restore the catalytic tape content to w. Observe that this step uses $O(\log n)$ work tape cells. Thus, the above arguments imply that $L \in \mathsf{ACL}(A)$ via the machine M'.

The lower bound on $\mathcal{R}(A_k)$ follows Proposition 2.2 since the set A_k is exactly the set of codewords of Spielman codes that have $\delta = O(1)$ [Spi97]. The lower bound on $\mathcal{P}(A_k)$ follows from the minimum distance of the set A_k is $d > 1$, and hence no two elements of A_k can be covered by the same subcube. Hence $\mathcal{P}(A_k) \geq |A_k|$ which is at least $2^{m/4}$. □

We remark that it also suffices if the code used has a rate that is a polynomial in n (message length), a good distance that is logspace constructible and decodable. In addition to the Speilman codes [Spi97], logspace decodable codes from [GK06] also suffice for the above theorem.

6 An Improvement on the Subcube Partition Complexity of the Catalytic Set

In Theorem 1.2, we showed that any PSPACE algorithm can be simulated in almost-catalytic logspace by restoring catalytic content w that are codewords of a carefully defined code as the set A. The ideal case would be to cover every such w that appears as catalytic content. With this motivation, in the main result of this section (Theorem 1.3), we attempt to cover strings that are not codewords as well at the expense of using less space. This allows the set A to be larger than the one in Theorem 1.2 and also has a better subcube partition complexity.

Theorem 1.3. *For all $k \geq 1$, there exists $A_k \subseteq \{0,1\}^*$ such that $\mathsf{DSPACE}(\log^k n) \subseteq \mathsf{ACL}(A_k)$ where for every $m \geq 1$, $\mathcal{R}(A_k \cap \{0,1\}^m) \geq \frac{m}{4}$ and $\mathcal{P}(A_k \cap \{0,1\}^m) = 2^{m/4+\Omega(\log m)}$.*

Proof. Let $L \in \mathsf{DSPACE}(\log^k n)$ via a Turing machine M. Let $m = n^k$ and C be an $[m, m/4, \alpha m]_2$ logspace decodable Spielman code where $\alpha > 0$ is a constant [Spi97]. We crucially use the existence of the family of functions (f_m) (Theorem 19 of [Spi97]) in our algorithm defined as: $f_m \colon \{0,1\}^m \times \{0,1\}^{\log m} \to \{0,1\}$ is a Boolean function that takes in word $w \in \{0,1\}^m$ such that $\exists y \in \{0,1\}^{m/4}$ with $d(C(y), w) \leq \frac{\alpha m - 1}{2}$ and an index $j \in [m]$ and outputs 1 if and only if

$w_j \neq C(y)_j$. If $w_j \neq C(y)_j$, then we shall denote them as corrupted bits/indices of w. Here d denotes the Hamming distance between two binary strings.

In addition, these functions can be computed by a log-space uniform family of bounded fan-in log-depth polynomial size circuits. Note that these circuits can be evaluated in $O(\log n)$ space. Hence for any given $w \in \{0,1\}^m$ and $j \in [m]$, $f(w,j)$ can be computed in $O(\log n)$ space.

We define T_m to be the set of all strings that are uniquely decodable to some codeword in C. More precisely, $T_m = \left\{z \in \{0,1\}^m \mid \exists y \in \{0,1\}^{m/4}, d(z, C(y)) \leq \frac{\alpha m - 1}{2}\right\}$. Define $A_k = \bigcup_{m \geq 1} T_m$. Since ℓ is asymptotically smaller than $\alpha \cdot m$, the set T_m must also contain words that are at a distance of at most ℓ from some codeword in C. This gives the desired lower bound on the $|A_k \cap \{0,1\}^m|$.

The description of an ACL machine M' simulating M is given in Algorithm 3.

Algorithm 3 Description of M' on input x and catalytic tape content w with $|w| = m$.

1: Partition $[m]$ into disjoint contiguous blocks $B_1, B_2, \ldots B_\ell$ each of size $b = \log^k n$.
2: Using the function f_m, find $i \in [\ell]$ in w such that $|\{j \in B_i \mid f_m(w,j) = 1\}| \leq \frac{\log n}{\log(\log^k n)}$
3: If such an i does not exist, set $i = \ell$, $E = \emptyset$, and go to line 6. //in this case $w \notin A_k$
4: Store the start and end indices of the block B_i. Call them p and q respectively.
5: Let $E = \{i \in [b] \mid f_m(w, p+i) = 1\}$ be the corrupted bits of B_i.
6: Run M on x using catalytic tape cells indexed by B_i as work space for M and accept if and only if M accepts.
7: **Restoration:** Let w' be the content of the catalytic tape at the end of the computation. For each $j \in B_i$, if $[f_m(w', j) = 1)] \oplus [j \in E]$, flip w'_j.

We argue correctness first. Irrespective of whether $w \in A_k$ or not, there will be an index B_i that is identified in step 3 of the algorithm and the cells of the catalytic tape indexed by B_i will be used to correctly simulate M on x which requires only $O(\log^k n)$ space. Hence correctness follows.

We argue now that the above algorithm restores w at the end of the computation if $w \in A_k$. $w \in A$ implies that there is a codeword $\gamma \in \{0,1\}^m$ that is at a Hamming distance of at most $\frac{m}{\log^k n} \times \frac{\log n}{\log(\log^k n)}$ from w. By averaging, there must be a block B_i with at most $\frac{\log n}{\log(\log^k n)}$ errors.

Since $|B_i| \leq O(\log^k n)$, we still have that $d(w', \gamma) \leq d(w, \gamma) + O(\log^k n) \leq \frac{m}{\log^k n} \times \frac{\log n}{\log(\log^k n)} + O(\log^k n) \leq \frac{\alpha m - 1}{2}$ for large enough n. Hence the word w' is still within the decoding radius of the code that we started with.

Let B_i be the block chosen by the algorithm in step 2. If $j \notin B_i$, step 6 does not change j and hence $w'_j = w_j$. If $j \in B_i$, then w_j may get changed during the simulation of the machine M in step 6. We argue now that these bits also get restored. We do this in two cases.

Case 1: $w'_j = \gamma_j$. This implies that j-th bit in w'_j is not corrupted. If in addition, $j \in E$, that implies $w_j \neq \gamma_j$. Hence $w_j \neq w'_j$. Hence when the algorithm flips w'_j in line 7, it makes it equal to w_j. In case, $j \notin E$, we have that $w_j = \gamma_j = w'_j$, and hence no flipping is required in line 7 of the algorithm.

Case 2: $w'_j \neq \gamma_j$. This implies that j-th bit in w'_j is corrupted. If in addition, $j \notin E$, that implies $w_j = \gamma_j$. Hence $w_j \neq w'_j$. Hence when the algorithm flips w'_j in line 7, it makes it equal to w_j. In case, $j \in E$, we have that $w_j \neq \gamma_j$ and hence $w_j = w'_j$, and hence no flipping is required in line 7 of the algorithm.

We now argue the space bound for M'. The indices $p, q \in [m]$ and $i, j \in [\ell]$ can all be stored in $O(\log n)$ space. Note that we store $E \subseteq [b]$, which can be done using $|E| \log b$ many bits. Since the number of corrupted bits $|E|$, is at most $\frac{\log n}{\log(\log^k n)}$, we can store E in the work tape using $O(\log n)$ bits. As mentioned above, the function f_m can also be computed in $O(\log n)$ space as needed in lines 2, 5 and 7 of the algorithm. This establishes the space bound.

The lower bound on $\mathcal{R}(A_k)$ follows Proposition 2.2 since the set A_k is exactly the set of codewords of Spielman codes that have $\delta = O(1)$ [Spi97] and that $R_\epsilon(A)$ is a monotone property with respect to A. The lower bound on $\mathcal{P}(A_k)$ follows from Lemma 2.6 noting that the set A_k is defined to the union of Hamming balls of radius $\frac{m}{(\log^{k-1} n)\log(\log^k n)}$. □

Two Limitations of the Approach towards $\mathsf{DSPACE}(\log^k n) \subseteq \mathsf{ACL}(\Sigma^*)$ We first note a limitation of the approach due to the fact that there is a direct simulation of the $\mathsf{DSPACE}(\log^k n)$ machine in the argument. [BCK+14] observed that there cannot be a step-by-step simulation of Turing machines that use $\omega(s(n))$ space by using catalytic Turing machines that uses $s(n)$ work space and even $2^{s(n)}$ catalytic space. We note that this also implies a limitation of our approach towards almost-catalytic Turing machines as well.

Consider the following family of algorithms that attempts to show $\mathsf{DSPACE}(\log^k(n)) \subseteq \mathsf{CL}$ via error correcting codes as follows: Let L belongs to $\mathsf{DSPACE}(\log^k(n))$ via machine M. Then we construct a catalytic machine as follows: (1) Apply logspace computable transformations to the initial catalytic tape content to make it "recoverable" from $O(\log^k n)$ errors. (2) Run the machine M on input x on the catalytic tape. (3) Correct the $O(\log^k n)$ errors on the catalytic tape and restore w. (4) Accept if M accepts and reject otherwise.

Proposition 6.1. *There is no simulation of determinisitic polylogarithmic space in catalytic logspace via direct simulation and using logspace decodable error correcting codes.*

Proof. We argue that the direct simulation cannot work as it is. Indeed, it implies that the machine M must necessarily run in expected polynomial time (with respect to choice of initial catalytic content). In other words, every $O(\log^k n)$ machine must run in polynomial time - a statement which can be proved to be false. We argue the same below.

Consider the case when machine M is constructed as follows: M visits all its configurations before halting. There are $O(\exp(\log^k n))$ many configurations to the machine and thus it takes time at least $O(\exp(\log^k n))$ to run. In step 2 of our algorithm, we run the machine M directly (i.e. step-by-step). So our algorithm too runs in time at least $O(\exp(\log^k n))$. The initial content w does not affect step 2, so the average running time (over the choice of w) is still super-polynomial which is a contradiction to the fact that any CL machine takes polynomial running time on average over the choice of w. □

We observe a second limitation of the approach due to the fact that we cannot expect linear codes to have a covering radius as low as required for the algorithm. More precisely, for the approach, we need the covering radius of the code $C \subseteq \{0,1\}^m$ to be at most $\frac{m}{(\log m)^{k-1} \log\log n}$. However, for every code with rate r, the covering radius is known [CKMS85] to be at least $m\left(\frac{1}{2} - \frac{\sqrt{r}}{2^{3/2}}\right)$ which is at least $\Omega(n)$ even for constant rate codes.

Acknowledgments We would like to thank the anonymous reviewers for pointing out that Theorem 1.1 works for any $A \subseteq \Sigma^*$ (previous versions stated restrictions on A) and for pointing out Proposition 3.3.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009. pages 6

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: Catalytic space. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 857–866, New York, NY, USA, 2014. Association for Computing Machinery. pages 2, 3, 4, 5, 6, 14

[BKLS18] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space: Non-determinism and hierarchy. *Theory of Computing Systems*, 62(1):116–135, jan 2018. pages 2, 3

[CKLS16] Sourav Chakraborty, Raghav Kulkarni, Satyanarayana V. Lokam, and Nitin Saurabh. Upper bounds on fourier entropy. *Theoretical Computer Science*, 654:92–112, 2016. Computing and Combinatorics. pages 7

[CKMS85] G. Cohen, M. Karpovsky, H. Mattson, and J. Schatz. Covering radius—survey and recent results. *IEEE Transactions on Information Theory*, 31(3):328–343, 1985. pages 14

[CLMP24] James Cook, Jiatu Li, Ian Mertz, and Edward Pyne. The structure of catalytic space: Capturing randomness and time via compression. *Electron. Colloquium Comput. Complex.*, TR24-106, 2024. pages 3, 4

[CM24] James Cook and Ian Mertz. Tree evaluation is in space $o(\log n \cdot \log\log n)$. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1268–1278, New York, NY, USA, 2024. Association for Computing Machinery. pages 3

[CMW+12] Stephen Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2), jan 2012. pages 3

[DGJ+20] Samir Datta, Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Randomized and symmetric catalytic computation. In Henning Fernau, editor, *Computer Science – Theory and Applications*, pages 211–223, Cham, 2020. Springer International Publishing. pages 3

[GJST19] Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Unambiguous catalytic computation. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS 2019)*, volume 150 of *LIPIcs*, pages 16:1–16:13, 2019. pages 3

[GK06] Venkatesan Guruswami and Valentine Kabanets. Hardness amplification via space-efficient direct products. In *Proceedings of the 7th Latin American Conference on Theoretical Informatics*, LATIN'06, page 556–568, Berlin, Heidelberg, 2006. Springer-Verlag. pages 5, 12

[GL94] Craig Gotsman and Nathan Linial. Spectral properties of threshold functions. *Combinatorica*, 14(1):35–50, March 1994. pages 7

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. pages 6

[Kou16] Michal Koucký. Catalytic computation. *Bull. EATCS*, 118, 2016. pages 3

[Mer23] Ian Mertz. Reusing space: Techniques and open problems. *Bull. EATCS*, 141, 2023. pages 3

[Pyn24] Edward Pyne. Derandomizing logspace with a small shared hard drive. *Electron. Colloquium Comput. Complex.*, TR23-168, 2024. pages 4

[Spi97] Daniel A. Spielman. The complexity of error-correcting codes. In Bogdan S. Chlebus and Ludwik Czaja, editors, *Fundamentals of Computation Theory*, pages 67–84, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. pages 5, 11, 12, 14

A Appendix

A.1 Proof of Lemma 2.2

Proof. Let $E : \{0,1\}^k \to \{0,1\}^m$ be the encoding associated with the given $C = (m, k, \delta m)_2$ code. Let ℓ be the random projection complexity of the set of codewords. We show that when $\epsilon = 1/2^{2k}$, $\mathcal{R}_\epsilon(C) \geq k$.

Consider any set $S \subseteq \{0,1\}^k$ such that $|S| \geq 2^{\ell-1}$. We will fix S later. Firstly notice that,

$$\Pr_{\substack{T\subseteq[m]\\|T|=\ell}}\left[\forall x \neq y \in S \text{ such that } E(x)_T \neq E(y)_T\right] \leq \Pr_{\substack{T\subseteq[m]\\|T|=\ell}}\left[\left|A|_T \cap \{0,1\}^\ell\right| \geq 2^{\ell-1}\right]$$

Our goal is to show a lower bound of $1-\epsilon$ for the term in the left-hand size. Instead, we start by analysing the complementary event and show that its probability is upper bounded by ϵ. For any distinct pair $x, y \in S$,

$$\Pr_{\substack{T\subseteq[m]\\|T|=\ell}}\left[E(x)_T = E(y)_T\right] \leq (1-\delta)^\ell$$

The above follows from the fact that since A is a code of distance δm, the probability for $E(x)$ and $E(y)$ to be same at a random index in T is at most $1-\delta$. Thus, we have,

$$\Pr_{\substack{T\subseteq[m]\\|T|=\ell}}\left[\exists x \neq y \in S \text{ such that } E(x)_T = E(y)_T\right] \leq (1-\delta)^\ell \binom{|S|}{2}$$

Choosing S to be any subset of $\{0,1\}^k$ of size 2^{k-1}, we want $(1-\delta)^\ell \binom{2^{k-1}}{2} \leq \epsilon$. This means that

$$\ell \geq \frac{2k - \log(1/\epsilon)}{\log(1/(1-\delta))}$$

In addition, since $|S| \geq 2^{\ell-1}$, we have $k \geq \ell$. For our choice of ϵ all values of ℓ up to k are feasible. Since we need the maximum possible ℓ, we choose $\ell = k$. This completes the proof. □

A.2 Proof of Proposition 3.3

Proof. Consider a language L that can be decided by a machine M in n^k space. Now, we shall construct an almost-catalytic Turing machine M' deciding L using $m = bn^k$ catalytic space for some $b \geq 1$. We shall define the catalytic set A used by M' as $\{w \mid w \text{ is of the form } 0^{n^k}(0+1)^{(b-1)n^k} \mid n \geq 1\}$.

The machine M' works as follows: Simulate M on input x using the first n^k many bits of the catalytic tape. Now, for restoration, we set the first m many bits back to 0, which belongs to the set A. Finally, we observe that $|A \cap \{0,1\}^m| = 2^{(b-1)n^k} = 2^{m-m/b}$. □

A.3 Proof of Proposition 3.4

Proof. Let $L \in \mathsf{PSPACE}$ via a Turing machine M_L. We want to show that we can construct a $\mathsf{ACL}(A)$ machine M' such that it decides L. Say $A \in \mathsf{L}$ via the machine M_A. Following is the description of M' (Algorithm 4) with catalytic tape initialized with $w \in \{0,1\}^{\mathsf{poly}(n)}$:

Algorithm 4 Machine M' on $x \in \{0,1\}^n$ and $w \in \{0,1\}^{\mathsf{poly}(n)}$

1: Check if $w \in A$ using the work space to run the machine M_A. If not, run the machine M_L on the catalytic space. Accept if M_L accepts, Rejects if M_L rejects.
2: Initialize $count = 0$
3: **repeat**
4: Run machine M_A on w:
5: **if** $w \in A$ **then**
6: Increment $count$.
7: Update $w = w - 1$.
8: **end if**
9: **until** w becomes all 0's
10: Run M_L on catalytic tape. If M_L accepts, set $flag = true$, else $flag = false$
11: **repeat**
12: Run machine M_A on w:
13: **if** $w \in A$ **then**
14: Decrement $count$.
15: Update $w = w + 1$
16: **end if**
17: **until** $count = 0$
18: If $flag = true$ then Accept, otherwise Reject and halt.

Because A is in L, we can compute the membership of w in A using only the work space, which

is logarithmic in size. If $w \notin A$, it is not essential to restore the catalytic tape hence we simply run the machine M_L on the catalytic tape without restoring w. Next, if A is sparse there are only $\mathsf{poly}(n)$ many strings that the machine M_A would accept. A counter that remembers the position of such a string in a lexicographically ordered A, would need only $O(\log n)$ many bits for its storage. So if $w \in A$, we start "decrementing" the string while incrementing the counter, until when w becomes all 0's, $count$ stores exactly the position of the string in a lexicographically ordered A. Finally, we can simply run the machine M_L on the catalytic tape, and knowing the position of w (in the lexicographically ordered A) stored by $count$ helps us restore w at the end. □

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il