



Time Hierarchies for Sampling Distributions

Thomas Watson*

March 25, 2012

Abstract

We prove that for every constant $k \geq 2$, every polynomial time bound t , and every polynomially small ϵ , there exists a family of distributions on k elements that can be sampled exactly in polynomial time but cannot be sampled within statistical distance $1 - 1/k - \epsilon$ in time t . Our proof involves reducing the problem to a communication problem over a certain type of noisy channel. We solve the latter problem by giving a construction of a new type of list-decodable code, for a setting where there is no bound on the number of errors but each error gives more information than an erasure.

1 Introduction

The most commonly studied computational problems in theoretical computer science are *search problems*, where there is a relation specifying which outputs are acceptable, and the goal is to find any acceptable output. Another important type of computational problem is *sampling problems*, where the goal is for the output to be distributed according to (or at least statistically close to) a specified probability distribution.

Sampling problems have received much attention in the algorithms community. For example, there has been substantial work on algorithms for sampling graph colorings [FV07], independent sets [LV99, Vig01], matchings [JS89, JSV04], lattice tilings [LRS01, Wil04], knapsack solutions [MS04], linear extensions of partial orders [BD99, Wil04], factored numbers [Bac88, Kal03], DNF solutions [KLM89], eulerian tours [CCM10], stable marriages [BGR08], contingency tables [KTV99, CDR10, and references within], and spanning trees [PW98, Wil96, and references within]. In the complexity community, historically most research has focused on search problems (and the special case of decision problems). However, there has been a surge of interest in complexity-theoretic results that accord sampling problems a status as first-class computational problems [GGN10, Vio12, Aar11, LV11, DW12, Vio11].

In the context of sampling problems, we revisit the genesis of complexity theory. In their seminal paper, Hartmanis and Stearns [HS65] proved a *time hierarchy theorem* for decision problems, showing that there are decision problems that are solvable by deterministic algorithms running in time t but not by deterministic algorithms running in time a little less than t . This is often considered the first result in complexity theory. We study the corresponding question for sampling

*Computer Science Division, University of California, Berkeley. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0946797 and by the National Science Foundation under Grant No. CCF-1017403.

problems. First, observe that there is a trivial time hierarchy for exact sampling: In time t , an algorithm can produce a certain output with probability $1/2^t$, which clearly cannot be done in time less than t . The interesting question is whether a *robust* time hierarchy can be proved, showing that there exists a distribution family that can be sampled in time t , but that algorithms running in time a little less than t cannot even come close to sampling (in statistical distance). We succeed in proving such a hierarchy theorem for distributions on constant-size domains, showing that algorithms running in a sufficiently smaller amount of time cannot sample the distribution much better than the trivial statistical distance achieved by the uniform distribution.

There are several proofs of time hierarchy theorems for nondeterministic algorithms and other models of computation [Coo73, SFM78, Žák83, FS11], but these proofs do not directly carry over to our setting. On the surface, our setting may seem more closely related to the long-standing open problem of proving a hierarchy for polynomial-time randomized algorithms solving decision problems. The chief difficulty in the latter setting is that an algorithm must satisfy the “promise” of having bounded error on every input, and it is not known how to guarantee this while diagonalizing against a randomized algorithm that may not have bounded error. There is a beautiful line of research that circumvents this obstacle by working in slightly-nonuniform or average-case settings [Bar02, FS04, GST11, FST05, Per05, GHP05, vMP07, Per07, KvM10, Its10]. Our setting is intrinsically different because there is no “promise” that could be violated: Whatever algorithm we consider, it is guaranteed to sample some family of distributions. We have fundamentally different issues to address.

1.1 Result

We start with our definitions. We let $[k] = \{1, \dots, k\}$ and let \mathbb{N} denote the set of positive integers. For distributions D, D' on $[k]$, recall that the statistical distance is defined as $|D - D'| = \max_{S \subseteq [k]} |\Pr_D(S) - \Pr_{D'}(S)|$.

For a function $k : \mathbb{N} \rightarrow \mathbb{N}$, we define a k -family to be a sequence $D = (D_1, D_2, D_3, \dots)$ where D_n is a distribution on $[k(n)]$. For a function $\delta : \mathbb{N} \rightarrow [0, 1]$, we say a randomized algorithm A δ -samples a k -family D if when given n as input, A outputs an element $A(n) \in [k(n)]$ such that the output distribution satisfies $|A(n) - D_n| \leq \delta(n)$. For a function $t : \mathbb{N} \rightarrow \mathbb{N}$, we say that A runs in time t if for all $n \in \mathbb{N}$, A always halts in at most $t(n)$ steps when given n as input.¹ Finally, we define

$$\text{SAMP}_{k,\delta}(t)$$

to be the class of k -families δ -sampled by algorithms running in time t .²

Theorem 1. *For every constant $k \geq 2$ and every constant $c \geq 1$,*

$$\text{SAMP}_{k,0}(\text{poly}(n)) \not\subseteq \text{SAMP}_{k,1-1/k-\epsilon}(t)$$

where $\epsilon(n) = 1/n^c$ and $t(n) = n^c$.

We mention that our definition of k -families is “unary”, since there is one distribution for each n . We could alternatively define a k -family to be a function mapping bit strings of length

¹We measure the running time here as a function of the value of the input, not the bit length of the input.

²If we write something such as $\text{SAMP}_{O(\log n), 1/2-1/\text{poly}(n)}(\text{poly}(n))$, we formally mean the union of $\text{SAMP}_{k,\delta}(t)$ over all functions of the form $k = O(\log n)$, $\delta = 1/2 - 1/\text{poly}(n)$, and $t = \text{poly}(n)$.

n to distributions on $[k(n)]$ (for all n). This would more realistically model algorithmic sampling problems, but Theorem 1 is stronger with the unary definition. We also mention that our proof of Theorem 1 generalizes straightforwardly to show that for every constant $k \geq 2$ and all sufficiently constructible monotone functions t and ϵ such that $2^{t(n)} \leq t(2^{\text{poly}(n)})$, we have $\text{SAMP TIME}_{k,0}(\text{poly}(t(\text{poly}(n))/\epsilon(\text{poly}(n)))) \not\subseteq \text{SAMP TIME}_{k,1-1/k-\epsilon}(t)$. Finally, we mention that our proof of Theorem 1 relativizes, and that it carries through without change for quantum algorithms instead of classical randomized algorithms.

We give the intuition for Theorem 1 in Section 2. We give the formal proof of Theorem 1 in Section 3. One key ingredient in the proof is a certain type of code, which we construct in Section 4. We conclude with open problems in Section 5.

2 Intuition

Recall that the original deterministic time hierarchy of [HS65] is proved by diagonalization: A separate input length n_i is reserved for each algorithm A_i running in the smaller time bound, and an algorithm running in the larger time bound is designed which, when given an input of length n_i , simulates A_i and outputs the opposite answer. This “direct complementation” strategy cannot possibly work in our setting, because of Brouwer’s fixed point theorem: Suppose we design an algorithm running in the larger time bound which takes n_i and simulates $A_i(n_i)$ any number of times (drawing samples from the distribution $A_i(n_i)$ as a black box) and then performs some computation and produces an output. This algorithm would implement a continuous function from distributions on $[k]$ to distributions on $[k]$,³ where the input to the function represents the distribution $A_i(n_i)$. This function would have a fixed point, so there would be some distribution, which $A_i(n_i)$ might sample, that would cause the diagonalizing algorithm to produce exactly the same distribution. The trivial time hierarchy for exact sampling mentioned in the introduction gets around this by exploiting the fact that $A_i(n_i)$ cannot be an arbitrary distribution; it must be “discretized”. However, the latter observation cannot be used to get a robust time hierarchy with a nonnegligible statistical distance gap.

Since direct complementation cannot work, we take as a starting point the *delayed diagonalization* technique introduced by Žák [Žák83]. This technique can be used to prove a time hierarchy for solving decision problems with almost any uniform model of computation that is syntactic (meaning there is no promise to be satisfied). The idea is to space out the n_i ’s so that n_{i+1} is exponentially larger than n_i , and use all the input lengths from n_i to $n_{i+1} - 1$ to diagonalize against the i^{th} algorithm A_i . On inputs of length $n \in \{n_i, \dots, n_{i+1} - 2\}$ the diagonalizing algorithm copies the behavior of A_i on inputs of length $n + 1$, and on inputs of length $n = n_{i+1} - 1$ the diagonalizing algorithm “does the opposite” of A_i on inputs of length n_i (by brute force). Thus A_i cannot agree with the diagonalizing algorithm for all $n \in \{n_i, \dots, n_{i+1} - 1\}$ or we would obtain a contradiction.

The delayed diagonalization technique leads to a straightforward proof of the $k = 2$ case of Theorem 1, as follows. Let us use $D = (D_1, D_2, \dots)$ to denote the k -family 0-sampled by A_i (the algorithm we are diagonalizing against) and $D^* = (D_1^*, D_2^*, \dots)$ to denote the k -family 0-sampled by our diagonalizing algorithm. We can let $D_{n_{i+1}-1}^*$ be concentrated entirely on the least likely outcome of D_{n_i} , say $M \in [2]$.⁴ Now for $n = (n_{i+1} - 2), \dots, n_i$, by induction we may assume that $\Pr_{D_{n+1}^*}(M) \geq 1 - \epsilon(n + 1)/2$ and thus $\Pr_{D_{n+1}}(M) \geq 1/2 + \epsilon(n + 1)/2$ (assuming

³In general we would talk about $[k(n_i)]$, but recall that k is a constant in Theorem 1.

⁴ M might seem like unusual notation here, but it is convenient in the formal proof, and it stands for “message”.

$|D_{n+1} - D_{n+1}^*| \leq 1/2 - \epsilon(n+1)$). By sampling from D_{n+1} many times and taking the majority outcome, we can ensure that $\Pr_{D_n^*}(M) \geq 1 - \epsilon(n)/2$. In the end we have $\Pr_{D_{n_i}^*}(M) \geq 1 - \epsilon(n_i)/2$ while $\Pr_{D_{n_i}}(M) \leq 1/2$, which gives a contradiction if $|D_{n_i} - D_{n_i}^*| \leq 1/2 - \epsilon(n_i)$.

This simple argument breaks down when $k \geq 3$. Suppose we let $D_{n_{i+1}-1}^*$ be concentrated on $M \in [k]$, the least likely outcome of D_{n_i} . If $D_{n_{i+1}-1}$ is uniform on $\{1, \dots, k-1\}$ then this would be consistent with any $M \in \{1, \dots, k-1\}$, since $D_{n_{i+1}-1}$ would simultaneously have statistical distance $1 - 1/(k-1) \ll 1 - 1/k$ from the distributions concentrated on such M 's. Note that it is impossible to have statistical distance $< 1 - 1/k$ from the distributions concentrated on *all* possible M 's, so $D_{n_{i+1}-1}$ would be forced to reveal *some* information about the correct M , namely it must rule out at least one value.

Here is the first idea we use to fix this problem. Instead of using a single input n_i to “close the cycle” and obtain a contradiction, suppose we reserve m inputs $n_i, n_i + 1, \dots, n_i + m - 1$ and let M_α be the least likely outcome of $D_{n_i+\alpha}$ for $\alpha \in \{0, 1, \dots, m-1\}$. Suppose that on these inputs, our diagonalizing algorithm could somehow obtain (with high probability) a list of m candidates for the sequence M_0, M_1, \dots, M_{m-1} , where at least one candidate is correct. Then we could have $D_{n_i+\alpha}^*$ put most of its probability mass on the α^{th} value from the α^{th} candidate sequence. If the α^{th} candidate sequence is the correct one, then we get $\Pr_{D_{n_i+\alpha}^*}(M_\alpha) \geq 1 - \epsilon(n_i + \alpha)/2$ while $\Pr_{D_{n_i+\alpha}}(M_\alpha) \leq 1/k$, which gives a contradiction if $|D_{n_i+\alpha} - D_{n_i+\alpha}^*| \leq 1 - 1/k - \epsilon(n_i + \alpha)$.

How do we get a small list of candidates? For some input n_i^* exponentially larger than n_i , suppose we encode the message M_0, M_1, \dots, M_{m-1} in some way as $\gamma \in [k]^\ell$ and use a block of ℓ inputs $n_i^*, n_i^* + 1, \dots, n_i^* + \ell - 1$ to “declare” the codeword γ , by having $D_{n_i^*+j-1}^*$ be concentrated entirely on γ_j for $j \in [\ell]$.⁵ Then we are faced with the following communication problem over a noisy channel: For some smaller inputs $n < n_i^*$, we would like to recover the original message so we can “retransmit” it to even smaller inputs (until it eventually reaches the inputs $n_i, n_i+1, \dots, n_i+m-1$). Our only way to get information about the message is by sampling from the distributions $D_{n_i^*+j-1}$ (for $j \in [\ell]$), which only weakly reflect the transmitted codeword (under the assumption that A_i ($1 - 1/k - \epsilon$)-samples D^*). Thus the algorithm A_i being diagonalized against serves as a noisy channel for transmitting the message from larger inputs to smaller inputs.

As noted above, it is information-theoretically impossible to uniquely recover the original message when $k \geq 3$, but provided we use a suitable encoding we may be able to recover a small list of candidates. Then for each candidate in the list we could use a disjoint block of ℓ inputs to retransmit the encoding of that candidate message. More precisely, suppose there exists a small set S of messages containing the correct one, such that by sampling from $D_{n_i^*+j-1}$ (for $j \in [\ell]$) we can discover S with high probability. Then for each message in S we could have a block of ℓ inputs (that are polynomially smaller than n_i^*) “declare” the codeword corresponding to that message. Then on even smaller inputs, the diagonalizing algorithm could sample from D on the inputs in a particular block to recover a small list of candidates for the message encoded by that block. This leads to a tree structure, illustrated in Figure 1.⁶ Each node in the tree attempts to transmit a codeword to its children, after attempting to receive a codeword from its parent by simulating A_i to get samples from D , and running some sort of list-decoder. Each node can see “which child it is” and interpret this as advice specifying which message on the list it is responsible for encoding and transmitting (the h^{th} child is responsible for the lexicographically h^{th} smallest message). The

⁵For notational reasons, it turns out to be more convenient for us to use 0-based indexing for the sequence of M_α 's and 1-based indexing for the coordinates of the codeword γ .

⁶The paper [vMP07] uses a similar tree of input lengths but for a different reason.

inputs $n_i, n_i + 1, \dots, n_i + m - 1$ are the leaves of the tree. The final overall list corresponds to these leaves; input $n_i + \alpha$ would get the α^{th} message of the overall list. So α specifies a path down the tree, and there must be some path along which the original message is faithfully transmitted. Provided the tree has height logarithmic in n_i and the list at each node has constant size, the overall list would have size polynomial in n_i , and for every input the diagonalizing algorithm would only need to get polynomially many samples from D on polynomially larger inputs, and would thus run in polynomial time.

However, there are complications with implementing the above idea. It is too much to hope that when a codeword is transmitted over the channel, we can recover a unique set S of candidate messages with high probability. To cut to the chase, what we will be able to guarantee is that there exists a fixed set S of $k - 1$ messages (where S depends on the distributions of D on the block we are trying to receive from) such that we can get a random set of messages T which, with high probability, contains the correct message and is contained in S . We have no further control over the distribution of T . When $k = 3$ this is not a problem: Suppose we use the advice to specify whether the correct message is lexicographically first or last in S . The child corresponding to the correct advice will get $T = S$ with some probability, and with the remaining probability T will contain only the correct message, and in either case the child knows what the correct message is. The child corresponding to the wrong advice may output garbage, but it does not matter.

The above argument does not generalize to $k \geq 4$. For example, when $k = 4$ and the correct message is the middle message in S , if we get $|T| = 2$ then we do not know whether the correct message is the first or second message in T . We now describe the key idea to solve this problem. For each message in S , consider the probability it is in T . By the pigeonhole principle, using a constant amount of advice we can identify a significant “gap” in these probabilities, so that every message in S has probability either above the gap or below the gap. By taking a certain number of samples of T and intersecting these sets, the probabilities go down exponentially in the number of samples, so the probabilities below the gap become vanishingly small while the probabilities above the gap remain very close to 1. Then by a union bound over the messages in S , we find that with high probability the intersection of our sampled sets T equals T^* , the set of messages in S with probabilities above the gap (which includes the correct message). As described above, since we get the unique set T^* with high probability, we can retransmit the correct codeword provided we know which message of T^* is the correct one. The branching factor of the tree becomes k^2 because the advice needs to specify which of k possible “gaps” to use (and thus how many samples of T to take) as well as the lexicographic index of the correct message within T^* .⁷

We now explain the decoding process in more detail. Suppose we are trying to receive the codeword $\gamma \in [k]^\ell$ transmitted by some block of inputs $n, \dots, n + \ell - 1$. Then for $j \in [\ell]$, $\Pr_{D_{n+j-1}}^*(\gamma_j)$ is close to 1 (assuming we are on the “good” path down the tree) and thus $\Pr_{D_{n+j-1}}(\gamma_j)$ is somewhat larger than $1/k$ (since we are assuming for contradiction that A_i $(1 - 1/k - \epsilon)$ -samples D^*). There is some other value $\kappa_j \in [k]$ such that $\Pr_{D_{n+j-1}}(\kappa_j) < 1/k$. Hence if we repeatedly sample from D_{n+j-1} and let $\rho_j \subseteq [k]$ be the set of values that occur with frequency at least slightly greater than $1/k$ in the empirical distribution, then with high probability we get $\gamma_j \in \rho_j \subseteq [k] \setminus \kappa_j$. In general we will not get a unique ρ_j with high probability, since under D_{n+j-1} some symbols might occur with probability very close to the threshold used in defining ρ_j . We view $\rho = \rho_1 \cdots \rho_\ell$ as the received word. There is no bound on the number of “errors” here, but each error is more informative than an erasure ($\rho_j = [k]$ would correspond to an erasure). We need to construct a list-decodable error-

⁷We actually only need a branching factor of $(k - 1)^2$, but for simplicity we round it up to k^2 in the proof.

correcting code for this (non-traditional) setting. The list-decoding is deterministic, but since ρ is random, the list of messages T is also random. With high probability, $T \subseteq S$ where S is the list of messages for the received word $[k] \setminus \kappa_1 \cdots [k] \setminus \kappa_\ell$.

We now discuss our code construction. The codeword for a message is defined by interpreting the message as a bit string⁸ and evaluating all possible surjections $f : \{0, 1\}^{k-1} \rightarrow [k]$ on all possible sets of $k-1$ coordinates of the bit string. It can be shown that this code is list-decodable in principle (with list size $k-1$) by using the following lemma: For every set of k distinct bit strings of the same length, there exist $k-1$ coordinates on which they remain distinct. Our polynomial-time list-decoder uses this lemma in an iterative way, building and pruning a set of candidate strings of increasing lengths until it has arrived at the correct set of messages.

3 Proof of Theorem 1

In Section 3.1 we state a theorem giving an explicit construction of a certain type of code used in the proof of Theorem 1. Then in Section 3.2 we give the proof of Theorem 1.

3.1 Code Construction

We need a construction of a code for a new model of error-correction. Codewords are length- ℓ strings over the alphabet $[k]$, and each coordinate of a codeword can be corrupted to a subset of $[k]$ containing the correct symbol. More formally, we say a codeword $\gamma \in [k]^\ell$ is *consistent* with a received word⁹ $\rho \in (\mathcal{P}([k]))^\ell$ if $\gamma_j \in \rho_j$ for all $j \in [\ell]$. A traditional erasure corresponds to the case $\rho_j = [k]$, but in our model of error-correction that is forbidden: ρ_j must be a strict subset of $[k]$, so each coordinate of the received word is more informative than an erasure. The tradeoff is that, unlike in traditional error-correction settings, we do not assume any upper bound on the number of “errors”.

Theorem 2. *For every constant $k \geq 3$ there exists a polynomial-time encodable code $C : \{0, 1\}^m \rightarrow [k]^\ell$ where $\ell = \Theta(m^{k-1})$ such that the following holds. For every received word $\rho \in (\mathcal{P}([k]))^\ell$ with $\rho_j \neq [k]$ for all $j \in [\ell]$, there are at most $k-1$ messages $\mu \in \{0, 1\}^m$ whose codeword $C(\mu)$ is consistent with ρ ; moreover, the list of all such μ can be found in polynomial time given ρ .*

We prove Theorem 2 in Section 4. A simple application of the probabilistic method shows that such codes exist with $\ell = \Theta(m)$ if we drop the requirement that the encoding and list-decoding can be done in polynomial time.¹⁰ We are not aware of explicit constructions of such codes with $\ell = \Theta(m)$, but the polynomial length in Theorem 2 is good enough for our purpose. Also, our construction for Theorem 2 is elementary.

We remark that in the error-correction model of Theorem 2, it can be assumed without loss of generality that $|\rho_j| = k-1$ for all $j \in [\ell]$ (since we can always enlarge each coordinate of the received word to a set of size $k-1$, then find all the relevant messages, and then output only those messages whose codeword is consistent with the original received word). However, the way we have described the code is more convenient for our application.

⁸In the formal proof we actually use m to denote the bit length of the message, rather than the length of the sequence over $[k]$.

⁹Recall that $\mathcal{P}([k])$ denotes the power set of $[k]$.

¹⁰Specifically, there exist codes with $\ell = \Theta(m)$ such that for every set of k codewords there exists a coordinate on which each element of $[k]$ appears exactly once among the k codewords.

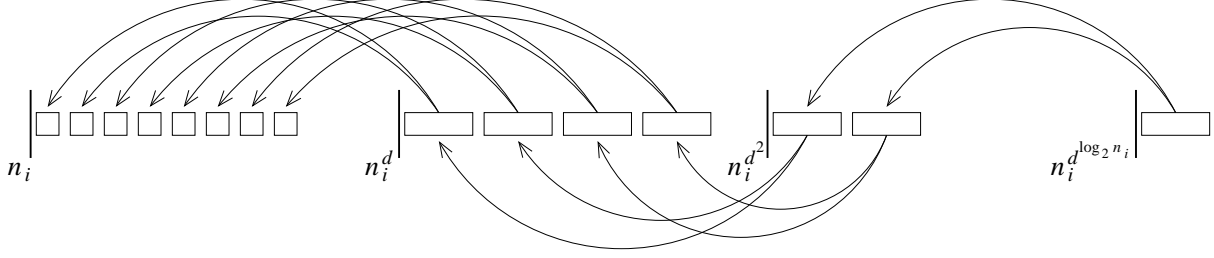


Figure 1: Tree of input blocks

3.2 The Diagonalization

We now prove Theorem 1. As sketched in Section 2, the $k = 2$ case of Theorem 1 is a simple application of delayed diagonalization and estimation by repeated sampling. Henceforth we assume $k \geq 3$. Let A_1, A_2, A_3, \dots be an enumeration of all randomized algorithms that run in time t and always output an element of $[k]$. We use a procedure $\text{Estimate}(A_i, n, \zeta, \eta)$ which returns a vector $(\pi_1, \pi_2, \dots, \pi_k) \in [0, 1]^k$ such that

- (i) with probability at least $1 - \eta$, $|\pi_\kappa - \Pr(A_i(n) = \kappa)| \leq \zeta$ for all $\kappa \in [k]$, and
- (ii) with probability 1, $\pi_1 + \pi_2 + \dots + \pi_k = 1$.

In other words, it returns a distribution that probably approximates the distribution of $A_i(n)$. If $\zeta, \eta > 0$ then by a standard Chernoff bound, $\text{Estimate}(A_i, n, \zeta, \eta)$ can be implemented in time $O(t(n) \cdot \frac{1}{\zeta^2} \log \frac{1}{\eta})$ by simulating $A_i(n)$ $O(\frac{1}{\zeta^2} \log \frac{1}{\eta})$ times and taking the empirical distribution.¹¹ Also, $\text{Estimate}(A_i, n, 0, 0)$ can be implemented in time $O(t(n) \cdot 2^{t(n)})$.

Algorithm 1 0-samples a k -family $D^* = (D_1^*, D_2^*, \dots)$, and we argue below that it runs in time $\text{poly}(n)$. Thus $D^* \in \text{SAMPTIME}_{k,0}(\text{poly}(n))$. We claim that $D^* \notin \text{SAMPTIME}_{k,1-1/k-\epsilon}(t)$. Suppose for contradiction there exists an i such that A_i $(1 - 1/k - \epsilon)$ -samples D^* . Let $D = (D_1, D_2, \dots)$ be the k -family that is 0-sampled by A_i . We have $|D_n - D_n^*| \leq 1 - 1/k - \epsilon(n)$ for all n .

The parameters used in Algorithm 1 are defined in Figure 2. We use the inputs from n_i through $n_{i+1} - 1$ to diagonalize against A_i . The parameters create a tree structure out of the inputs, illustrated in Figure 1. The tree is a full tree with branching factor k^2 and depth $\log_2 n_i$, with the leaves at level $b = 0$ and the root at level $b = \log_2 n_i$. Thus the number of leaves is $(k^2)^{\log_2 n_i}$. Each node of the tree has a contiguous block of inputs associated to it. Each leaf's block only consists of a single input, but each internal node's block has ℓ_i inputs, which represent the coordinates of codewords under the code C_i . Level b of the tree starts at input $n_{i,b} = n_i^{d^b}$. There are $(k^2)^{(\log_2 n_i) - b}$ nodes across level b , indexed by $\alpha \in \{0, 1, \dots, (k^2)^{(\log_2 n_i) - b} - 1\}$, and their blocks of inputs $N_{i,b,\alpha}$ are consecutive from left to right across the level. Writing α in base k^2 allows us to interpret α as specifying a path down the tree from the root to the current node. The input n_1 is an unspecified constant power of 2, which just needs to be large enough so the blocks $N_{i,b,\alpha}$ are all disjoint and $\log_2 n_1 > 1$. There exists such an n_1 since $d \geq 3k \log_2 k$. Hence line 1 of Algorithm 1 will find unique values i, b, α (if they exist).

The reason we use message length $m_i = \lceil \log_2 k^{(k^2)^{\log_2 n_i}} \rceil$ is because our messages represent sequences of length $(k^2)^{\log_2 n_i}$ over the alphabet $[k]$ (one symbol for each leaf of the tree). We

¹¹We are ignoring the logarithmic factor time overhead usually associated with simulating an algorithm using a universal algorithm.

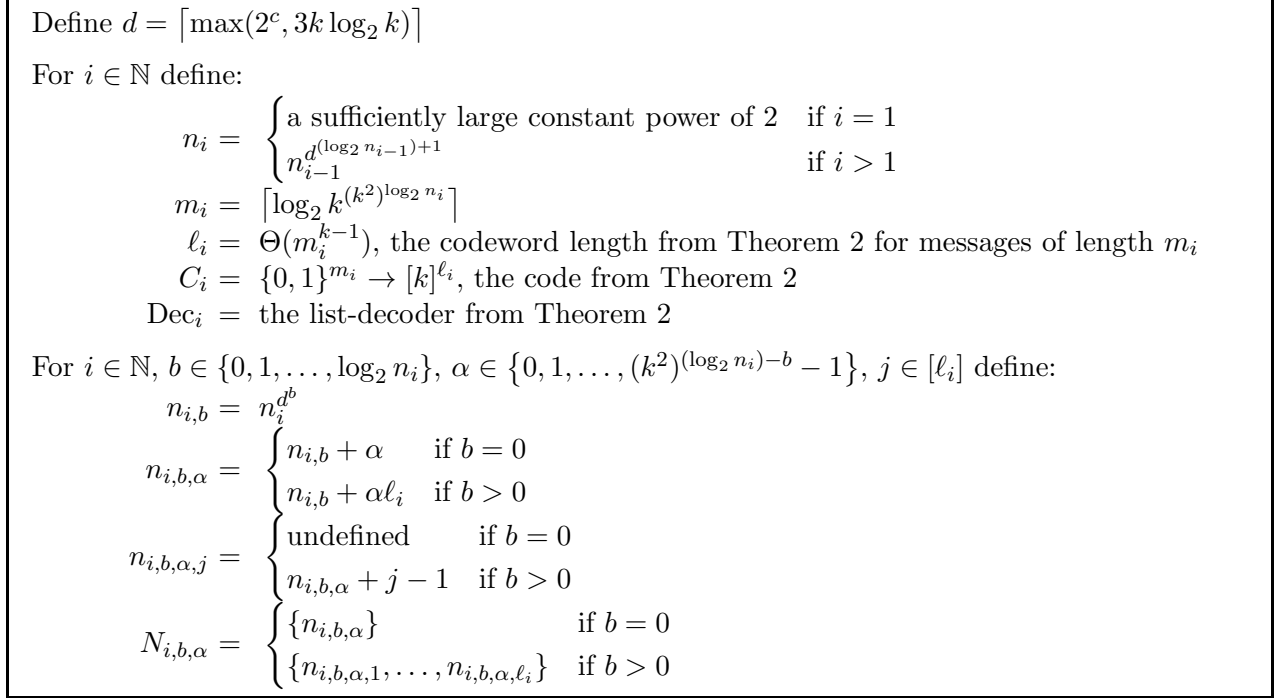


Figure 2: Notation for Algorithm 1

assume there is a canonical way of interconverting between sequences of length $(k^2)^{\log_2 n_i}$ over $[k]$ and messages in $\{0, 1\}^{m_i}$. It is most convenient for us to use 0-based indexing for the sequences $M_0, M_1, \dots, M_{(k^2)^{\log_2 n_{i-1}}}$ and 1-based indexing for the messages $\mu = \mu_1 \cdots \mu_{m_i}$, codewords $C(\mu) = C(\mu)_1 \cdots C(\mu)_{\ell_i}$, and received words $\rho = \rho_1 \cdots \rho_{\ell_i}$.

In general, each block of inputs $N_{i,b,\alpha}$ attempts to “receive” an encoded message via a noisy channel from its parent block and “send” the re-encoded message to its children blocks. Lines 3–24 are the *receiving phase*, and lines 25–32 are the *sending phase*. The receiving is different at the root ($b = \log_2 n_i$) because the algorithm generates the message directly without receiving it over a noisy channel. The sending is different at the leaves ($b = 0$) because instead of sending, the algorithm uses the message to attempt to deliver the coup de grâce and ensure that A_i fails to $(1 - 1/k - \epsilon)$ -sample D^* if it has not already failed somewhere along the chain of “transmissions”. The following claim is the heart of the analysis. It shows that there exists a path down the tree along which the original message μ^* (generated by the root) is faithfully transmitted.

Claim 1. *For every $b \in \{0, 1, \dots, \log_2 n_i\}$ there exists an $\alpha \in \{0, 1, \dots, (k^2)^{(\log_2 n_i)-b} - 1\}$ such that for every $n \in N_{i,b,\alpha}$, with probability $\geq 1 - \epsilon(n)/2$ Algorithm 1 reaches the sending phase (lines 25–32) and the μ computed in the receiving phase (lines 3–24) equals μ^* (the message generated by the root of the tree on line 8).*

Claim 2. *Algorithm 1 runs in time $\text{poly}(n)$.*

We now show how to finish the proof of Theorem 1 given these claims. By Claim 2, D^* is indeed in $\text{SAMP}_{k,0}(\text{poly}(n))$. Consider the good α from Claim 1 for $b = 0$. On input $n = n_{i,0,\alpha}$, with probability $\geq 1 - \epsilon(n)/2$ Algorithm 1 reaches the sending phase and the μ computed in the receiving

Algorithm 1: Diagonalizing algorithm for Theorem 1**Input:** $n \in \mathbb{N}$ **Output:** an element of $[k]$

```

1 find  $i, b, \alpha$  such that  $n \in N_{i,b,\alpha}$ 
2 if such values do not exist then halt and output an arbitrary element of  $[k]$ 

3 if  $b = \log_2 n_i$  then
4   foreach  $\alpha' \in \{0, 1, \dots, (k^2)^{\log_2 n_i} - 1\}$  do
5     let  $(\pi_1^{\alpha'}, \dots, \pi_k^{\alpha'}) = \text{Estimate}(A_i, n_i, 0, \alpha', 0, 0)$ 
6     let  $M_{\alpha'} = \arg \min_{\kappa \in [k]} (\pi_{\kappa}^{\alpha'})$  (breaking ties arbitrarily)
7   end
8   convert the sequence  $M_0, M_1, \dots, M_{(k^2)^{\log_2 n_i} - 1}$  to a bit string  $\mu \in \{0, 1\}^{m_i}$ 
9 else
10  write  $\alpha$  in base  $k^2$ :  $\alpha = \sum_{\tau=0}^{(\log_2 n_i) - b - 1} \alpha_{\tau} (k^2)^{\tau}$  where  $\alpha_{\tau} \in \{0, 1, \dots, k^2 - 1\}$ 
11  write  $\alpha_0$  in base  $k$ :  $\alpha_0 = (q - 1)k + (h - 1)$  where  $q, h \in [k]$ 
12  let  $\alpha' = \sum_{\tau=0}^{(\log_2 n_i) - b - 2} \alpha_{\tau+1} (k^2)^{\tau}$ 
13  let  $Q = (1/\epsilon(n_{i,b+1}))^{4q+2}$ 
14  foreach  $r \in [Q]$  do
15    foreach  $j' \in [\ell_i]$  do
16      let  $n' = n_{i,b+1,\alpha',j'}$ 
17      let  $(\pi_1^{j'}, \dots, \pi_k^{j'}) = \text{Estimate}(A_i, n', \epsilon(n')/4, \eta)$  where  $\eta = \epsilon(n_{i,b+1})/4\ell_i Q$ 
18      let  $\rho_{j'} = \{\kappa \in [k] : \pi_{\kappa}^{j'} \geq 1/k + \epsilon(n')/4\}$ 
19    end
20    let  $T_r = \text{Dec}_i(\rho) \subseteq \{0, 1\}^{m_i}$  where  $\rho = \rho_1 \cdots \rho_{\ell_i} \in (\mathcal{P}([k]))^{\ell_i}$ 
21  end
22  if  $|T_1 \cap \cdots \cap T_Q| < h$  then halt and output an arbitrary element of  $[k]$ 
23  let  $\mu$  be the lexicographically  $h^{\text{th}}$  smallest element of  $T_1 \cap \cdots \cap T_Q$ 
24 end

25 if  $b = 0$  then
26   convert  $\mu$  to a sequence  $M_0, M_1, \dots, M_{(k^2)^{\log_2 n_i} - 1}$  over  $[k]$ 
27   halt and output  $M_{\alpha}$ 
28 else
29   compute  $C_i(\mu)$ 
30   find  $j$  such that  $n = n_{i,b,\alpha,j}$ 
31   halt and output  $C_i(\mu)_j$ 
32 end

```

phase equals μ^* . Thus the sequence $M_0, M_1, \dots, M_{(k^2)^{\log_2 n_i - 1}}$ found on line 26 is the same as the sequence generated by the root of the tree on lines 4–7. Hence $M_\alpha = \arg \min_{\kappa \in [k]} (\Pr_{D_n}(\kappa))$ and in particular $\Pr_{D_n}(M_\alpha) \leq 1/k$. Since $\Pr_{D_n^*}(M_\alpha) \geq 1 - \epsilon(n)/2$, this contradicts the fact that $|D_n - D_n^*| \leq 1 - 1/k - \epsilon(n)$ (which follows from our contradiction assumption). This finishes the proof of Theorem 1. All that remains is to prove Claim 1 and Claim 2.

Proof of Claim 1. By induction on $b = \log_2 n_i, \dots, 0$. The base case $b = \log_2 n_i$ is trivial by the definition of μ^* (with $\alpha = 0$ and with probability 1, in fact). Now assume $b < \log_2 n_i$ and the claim holds for $b+1$. Let $\alpha' \in \{0, 1, \dots, (k^2)^{(\log_2 n_i) - b - 1} - 1\}$ be the good α from the induction hypothesis. For each $n' \in N_{i, b+1, \alpha'}$, say $n' = n_{i, b+1, \alpha', j'}$, the induction hypothesis says that on input n' , with probability $\geq 1 - \epsilon(n')/2$ Algorithm 1 reaches the sending phase and the μ computed in the receiving phase equals μ^* . Since $b+1 > 0$, by lines 28–32 this implies that $D_{n'}^*$ puts $\geq 1 - \epsilon(n')/2$ probability mass on $C_i(\mu^*)_{j'}$. Since $|D_{n'} - D_{n'}^*| \leq 1 - 1/k - \epsilon(n')$, we find that $D_{n'}$ puts $\geq 1/k + \epsilon(n')/2$ probability mass on $C_i(\mu^*)_{j'}$.

We show that there exist $g, h \in [k]$ such that $\alpha = (k^2)\alpha' + \alpha_0$ satisfies the desired properties, where $\alpha_0 = (g-1)k + (h-1)$. For any such α , suppose $n \in N_{i, b, \alpha}$ and consider Algorithm 1 on input n . Note that α' computed on line 12 is indeed the α' from the induction hypothesis, and the block $N_{i, b+1, \alpha'}$ is the parent of the block $N_{i, b, \alpha}$ in the tree (see Figure 1).

Now consider lines 15–19. For any $j' \in [\ell_i]$, let us denote $n' = n_{i, b+1, \alpha', j'}$, and let us define $\kappa_{j'}$ to be the least likely outcome of $D_{n'}$ (breaking ties arbitrarily). Then $D_{n'}$ puts $\geq 1/k + \epsilon(n')/2$ probability mass on $C_i(\mu^*)_{j'}$ and $< 1/k$ probability mass on $\kappa_{j'}$. Hence with probability $\geq 1 - \eta$ over the estimation on line 17,

$$\pi_{C_i(\mu^*)_{j'}}^{j'} \geq (1/k + \epsilon(n')/2) - \epsilon(n')/4 = 1/k + \epsilon(n')/4$$

and $\pi_{\kappa_{j'}}^{j'} < 1/k + \epsilon(n')/4$ and thus

$$C_i(\mu^*)_{j'} \in \rho_{j'} \subseteq [k] \setminus \kappa_{j'}. \quad (1)$$

Note that with probability 1 we have $\rho_{j'} \neq [k]$ for all j' and thus ρ is a valid received word. For any $r \in [Q]$, let E_r be the event (depending on the randomness of lines 15–19) that Equation (1) holds for all j' (in the r^{th} iteration of the loop on line 14). We have

$$\Pr(E_r) \geq (1 - \eta)^{\ell_i} \geq 1 - \eta \ell_i. \quad (2)$$

Now define

$$S = \text{Dec}_i([k] \setminus \kappa_1 \cdots [k] \setminus \kappa_{\ell_i}) \subseteq \{0, 1\}^{m_i}$$

and note that $|S| \leq k - 1$. Conditioned on E_r , we have $\mu^* \in T_r \subseteq S$ (since $C_i(\mu^*)$ is consistent with ρ , and all codewords consistent with ρ are also consistent with $[k] \setminus \kappa_1 \cdots [k] \setminus \kappa_{\ell_i}$). Note that for different r 's, T_r conditioned on E_r are independent and identically distributed. For each $\sigma \in S$ let us define p_σ to be the probability that $\sigma \in T_r$ conditioned on E_r . Note that $p_{\mu^*} = 1$ and since $|S| \leq k - 1$, by the pigeonhole principle there exists a $q \in [k]$ such that for every $\sigma \in S$, either $p_\sigma \geq \exp(-(\epsilon^*)^{4q+4})$ or $p_\sigma < \exp(-(\epsilon^*)^{4q})$ where $\epsilon^* = \epsilon(n_{i, b+1})$. We fix this value of q and the corresponding value $Q = (1/\epsilon^*)^{4q+2}$. For each $\sigma \in S$, we have

$$\Pr(\sigma \in T_1 \cap \cdots \cap T_Q \mid E_1 \cap \cdots \cap E_Q) = (p_\sigma)^Q$$

and we have either

$$(p_\sigma)^Q \geq \exp(-(\epsilon^*)^2) \geq 1 - \epsilon(n)/4k$$

or

$$(p_\sigma)^Q < \exp(-(1/\epsilon^*)^2) \leq \epsilon(n)/4k$$

regardless of which $n \in N_{i,b,\alpha}$ we are considering.¹² Defining

$$T^* = \{\sigma \in S : (p_\sigma)^Q > 1/2\},$$

by a union bound over $\sigma \in S$ we find that

$$\Pr((T_1 \cap \dots \cap T_Q) = T^* \mid E_1 \cap \dots \cap E_Q) \geq 1 - \epsilon(n)/4. \quad (3)$$

Since $(p_{\mu^*})^Q = 1 > 1/2$, we have $\mu^* \in T^*$. Now we fix $h \in [k]$ to be such that μ^* is the lexicographically h^{th} smallest element of T^* . Then when $(T_1 \cap \dots \cap T_Q) = T^*$, we have $|T_1 \cap \dots \cap T_Q| \geq h$ and so Algorithm 1 reaches the sending phase, and the μ computed in the receiving phase equals μ^* , as desired. Thus for every $n \in N_{i,b,\alpha}$ where $\alpha = (k^2)\alpha' + (q-1)k + (h-1)$ we have

$$\begin{aligned} & \Pr(\text{Algorithm 1 reaches the sending phase with } \mu = \mu^*) \\ & \geq \Pr((T_1 \cap \dots \cap T_Q) = T^*) \\ & \geq \Pr(E_1 \cap \dots \cap E_Q) \cdot \Pr((T_1 \cap \dots \cap T_Q) = T^* \mid E_1 \cap \dots \cap E_Q) \\ & \geq (1 - \eta \ell_i Q) \cdot (1 - \epsilon(n)/4) \\ & \geq 1 - \epsilon(n)/2 \end{aligned}$$

where the fourth line follows by Inequality (2) and Inequality (3), and the fifth line follows by $\eta \ell_i Q = \epsilon(n_{i,b+1})/4 \leq \epsilon(n)/4$ (where η is as on line 17 of Algorithm 1). This finishes the proof of Claim 1. \square

Proof of Claim 2. Line 1 can be done in $\text{poly}(n)$ time by direct computation. If $b = \log_2 n_i$ then

$$n \geq n_i^{d^{\log_2 n_i}} = 2^{n_i^{\log_2 d} \log_2 n_i} \geq 2^{n_i^c \log_2 n_i}$$

since $d \geq 2^c$, and so the number of iterations on line 4 is $\text{polylog}(n)$ and the computation on line 5 takes time $O(t(n_i) \cdot 2^{t(n_i)}) = O(n_i^c \cdot 2^{n_i^c}) \leq \text{poly}(n)$. Suppose $b < \log_2 n_i$. Lines 10–13 are simple calculations, and we have $Q \leq (1/\epsilon(n^d))^{4k+2} \leq \text{poly}(n)$ since $n_{i,b+1} = n_{i,b}^d \leq n^d$. We also have $m_i, \ell_i \leq \text{poly}(n_i) \leq \text{poly}(n)$ and so the loops on lines 14 and 15 have $\text{poly}(n)$ iterations. For lines 16 and 17, we have $n' \leq n_{i,b}^{d^2} \leq n^{d^2}$ and $\epsilon(n')/4 \geq 1/\text{poly}(n)$ and $\eta \geq 1/\text{poly}(n) \geq 1/\exp(\text{poly}(n))$ so the Estimate procedure takes time $\text{poly}(n)$. The list-decoding on line 20 takes time $\text{poly}(m_i) \leq \text{poly}(n)$. The sending phase (lines 25–32) trivially takes time $\text{poly}(n)$ since C_i is polynomial-time encodable. Overall, the running time is $\text{poly}(n)$. \square

4 Proof of Theorem 2

Before giving our construction of a code C satisfying the properties in Theorem 2, we give a key lemma.

¹²We can assume without loss of generality that c is large enough in terms of k for these inequalities to hold (recall that $\epsilon(n) = 1/n^c$).

4.1 A Combinatorial Lemma

For a set S and number a , we let $\binom{S}{a}$ denote the set of all subsets of S of size a . For a string $\sigma \in \{0, 1\}^b$ and $i \in [b]$ and $I \subseteq [b]$, we let σ_i denote the i^{th} bit of σ , and we let σ_I denote the length- $|I|$ string consisting of the bits of σ indexed by I .

Lemma 1. *For all $1 \leq a \leq b$ and every set of distinct strings $\sigma^1, \dots, \sigma^a \in \{0, 1\}^b$, there exists an $I \in \binom{[b]}{a-1}$ such that $\sigma_I^1, \dots, \sigma_I^a \in \{0, 1\}^{a-1}$ are distinct.*

Proof. By induction on a , with $a = 1$ and $a = 2$ being trivial. Suppose $a \geq 3$. By the induction hypothesis there exists an $I' \in \binom{[b]}{a-2}$ such that $\sigma_{I'}^1, \dots, \sigma_{I'}^{a-1}$ are distinct. If $\sigma_{I'}^a$ is different from each of $\sigma_{I'}^1, \dots, \sigma_{I'}^{a-1}$ then we can take an arbitrary $I \supseteq I'$ of size $a - 1$. Otherwise, $\sigma_{I'}^a = \sigma_{I'}^h$ for exactly one $h \in [a - 1]$. Since $\sigma^a \neq \sigma^h$, there exists an $i \in [b] \setminus I'$ such that $\sigma_i^a \neq \sigma_i^h$, and we can take $I = I' \cup \{i\}$. \square

It is not difficult to see that the $a - 1$ bound in Lemma 1 is tight (there do not always exist $a - 2$ coordinates on which a distinct bit strings remain distinct). We remark in passing that Lemma 1 can be viewed in terms of a certain “dual” of VC-dimension: While the VC-dimension of a set of bit strings is the size of a *largest* set of coordinates on which every pattern appears *at least* once, we are interested in the size of a *smallest* set of coordinates on which every pattern appears *at most* once.

4.2 Code Construction

We now give our construction of the code C for an arbitrary constant $k \geq 3$ and message length $m \geq k$. By convention we use the notation $\mu \in \{0, 1\}^m$ for messages, $\gamma \in [k]^\ell$ for codewords, and $\rho \in (\mathcal{P}([k]))^\ell$ for received words.

We define Surj_k to be the set of all surjections $f : \{0, 1\}^{k-1} \rightarrow [k]$. The coordinates of a codeword are indexed by $\binom{[m]}{k-1} \times \text{Surj}_k$, in other words by pairs I, f where I is a subset of $[m]$ of size $k - 1$ and $f : \{0, 1\}^{k-1} \rightarrow [k]$ is a surjection. We let $\ell = |\binom{[m]}{k-1} \times \text{Surj}_k| = \Theta(m^{k-1})$, and we define the code $C : \{0, 1\}^m \rightarrow [k]^\ell$ by

$$C(\mu) = (f(\mu_I))_{I \in \binom{[m]}{k-1}, f \in \text{Surj}_k}.$$

In other words, the I, f coordinate of the codeword is the evaluation of f on the bits of the message indexed by I . Encoding can clearly be done in polynomial time.

It just remains to exhibit a polynomial-time list-decoder for C . Let us fix an arbitrary received word $\rho \in (\mathcal{P}([k]))^\ell$ with $\rho_{I,f} \neq [k]$ for all I, f . We need to show that there are at most $k - 1$ messages whose codewords are consistent with ρ , and that moreover, these messages can be found in polynomial time given ρ .

For each $I \in \binom{[m]}{k-1}$ we define $\text{List}(\rho, I)$ to be the set of all $\sigma \in \{0, 1\}^{k-1}$ such that $f(\sigma) \in \rho_{I,f}$ for all $f \in \text{Surj}_k$. Note that the set $\text{List}(\rho, I)$ can be found efficiently given ρ and I by trying all possibilities.

Observation 1. *If $\mu \in \{0, 1\}^m$ is such that $C(\mu)$ is consistent with ρ , then for all $I \in \binom{[m]}{k-1}$, $\mu_I \in \text{List}(\rho, I)$.*

Lemma 2. *For all $I \in \binom{[m]}{k-1}$, $|\text{List}(\rho, I)| \leq k - 1$.*

Algorithm 2: List-decoder for Theorem 2**Input:** $\rho \in (\mathcal{P}([k]))^\ell$ with $\rho_{I,f} \neq [k]$ for all I, f **Output:** set of all $\mu \in \{0,1\}^m$ such that $C(\mu)$ is consistent with ρ

- 1 let $S_{k-1} = \text{List}(\rho, [k-1])$
- 2 **foreach** $n = k, \dots, m$ **do**
- 3 suppose $S_{n-1} = \{\sigma^1, \dots, \sigma^{|S_{n-1}|}\} \subseteq \{0,1\}^{n-1}$
- 4 find an $I \in \binom{[n-1]}{k-2}$ such that $\sigma_I^1, \dots, \sigma_I^{|S_{n-1}|}$ are distinct
- 5 let $S_n = \left\{ s \in \{0,1\}^n : s_{[n-1]} \in S_{n-1} \text{ and } s_{I \cup \{n\}} \in \text{List}(\rho, I \cup \{n\}) \right\}$
- 6 **end**
- 7 output the set of all $\mu \in S_m$ such that $C(\mu)$ is consistent with ρ

Proof. Consider any set of k distinct strings $\sigma^1, \dots, \sigma^k \in \{0,1\}^{k-1}$. There exists an $f \in \text{Surj}_k$ such that $\{f(\sigma^1), \dots, f(\sigma^k)\} = [k]$.¹³ Therefore since $\rho_{I,f} \neq [k]$ there exists an $h \in [k]$ such that $f(\sigma^h) \notin \rho_{I,f}$, which implies that $\sigma^h \notin \text{List}(\rho, I)$. \square

Now to see that C is list-decodable in principle, suppose for contradiction that there are k distinct messages μ^1, \dots, μ^k whose codewords are all consistent with ρ . Applying Lemma 1 with $a = k$ and $b = m$, there exists an $I \in \binom{[m]}{k-1}$ such that μ_I^1, \dots, μ_I^k are distinct. But for all $h \in [k]$, we have $\mu_I^h \in \text{List}(\rho, I)$ by Observation 1. Thus $|\text{List}(\rho, I)| \geq k$, which contradicts Lemma 2. Hence for our arbitrary received word ρ , there are at most $k-1$ messages whose codewords are consistent with ρ . Algorithm 2 finds this list of messages in polynomial time given ρ . The correctness of the algorithm follows immediately from the following claim and line 7 of the algorithm.

Claim 3. *For all $n = (k-1), \dots, m$, the following three properties hold: $S_n \subseteq \{0,1\}^n$, $|S_n| \leq k-1$, and for every $\mu \in \{0,1\}^m$ such that $C(\mu)$ is consistent with ρ we have $\mu_{[n]} \in S_n$.*

Proof. By induction on n . The base case $n = k-1$ is immediate from Lemma 2 and Observation 1, so assume $n \geq k$ and the claim holds for $n-1$. By the induction hypothesis, $|S_{n-1}| \leq k-1$ and so line 4 of the algorithm will succeed by Lemma 1 (with $a = |S_{n-1}|$ and $b = n-1$).

We now verify the three properties of S_n . The property $S_n \subseteq \{0,1\}^n$ is immediate. To see that $|S_n| \leq k-1$, suppose for contradiction that there are k distinct strings $s^1, \dots, s^k \in S_n$. Then since $|\text{List}(\rho, I \cup \{n\})| \leq k-1$ (by Lemma 2) and $s_{I \cup \{n\}}^h \in \text{List}(\rho, I \cup \{n\})$ for all $h \in [k]$, there must exist $h_1 \neq h_2$ such that $s_{I \cup \{n\}}^{h_1} = s_{I \cup \{n\}}^{h_2}$. Since $s_{[n-1]}^{h_1}, s_{[n-1]}^{h_2} \in S_{n-1}$ and $s_I^{h_1} = s_I^{h_2}$, we must have $s_{[n-1]}^{h_1} = s_{[n-1]}^{h_2} = \sigma^h$ for some h . But now $s_{[n-1]}^{h_1} = s_{[n-1]}^{h_2}$ and $s_n^{h_1} = s_n^{h_2}$, which contradicts our assumption that s^{h_1} and s^{h_2} are distinct. Thus we have verified that $|S_n| \leq k-1$. To verify the third property, consider an arbitrary $\mu \in \{0,1\}^m$ such that $C(\mu)$ is consistent with ρ . By the induction hypothesis, $\mu_{[n-1]} \in S_{n-1}$, and by Observation 1, $\mu_{I \cup \{n\}} \in \text{List}(\rho, I \cup \{n\})$. By line 5 of the algorithm, this means that $\mu_{[n]} \in S_n$. \square

We now discuss the running time of the algorithm. Line 4 can be implemented in polynomial time since an efficient algorithm for finding I can be gleaned from the proof of Lemma 1 (or less

¹³Because of this, we do not actually need to use all possible surjections in the definition of the code C . We can instead use any collection of functions with the property that for every set of k distinct strings in $\{0,1\}^{k-1}$, there exists a function in the collection that assigns each of the k strings a different value.

elegantly, since k is a constant, we can just try all possible subsets of size $k - 2$). Line 5 can be implemented efficiently by looking at each string in S_{n-1} and considering extending it with each possible symbol in $[k]$ and checking whether the $I \cup \{n\}$ coordinates form a string in $\text{List}(\rho, I \cup \{n\})$. Line 7 runs in polynomial time since C is efficiently encodable and consistency is easy to check.

Since our list-decoding algorithm is correct and runs in polynomial time, this completes the proof of Theorem 2.

5 Open Problems

A natural open problem is to prove a robust hierarchy theorem for sampling distributions on superconstant-size domains within polynomial time. It is also open to prove an “almost-everywhere” robust hierarchy for sampling non-unary families of distributions, where the algorithms running in the smaller time bound are required to fail on all but finitely many input lengths, instead of just on infinitely many input lengths.

Another open problem is to obtain explicit constructions of list-decodable codes as in Theorem 2 with constant rate instead of polynomially small rate. More generally, it would be interesting to obtain further results about codes with “errors” that are even more useful than erasures.

Another open problem is to find applications of our hierarchy theorem. A standard technique in complexity theory is “indirect diagonalization”, where a separation is proved by assuming the separation does not hold and deriving a contradiction with a known diagonalization-based result such as a hierarchy theorem. It would be interesting to use our hierarchy theorem in an indirect diagonalization.

Most generally, we advocate further study of sampling problems from a complexity theory perspective.

References

- [Aar11] Scott Aaronson. The equivalence of sampling and searching. In *Proceedings of the 6th International Computer Science Symposium in Russia*, pages 1–14, 2011.
- [Bac88] Eric Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, 1988.
- [Bar02] Boaz Barak. A probabilistic-time hierarchy theorem for slightly non-uniform algorithms. In *Proceedings of the 6th International Workshop on Randomization and Computation*, pages 194–208, 2002.
- [BD99] Russ Bubley and Martin Dyer. Faster random generation of linear extensions. *Discrete Mathematics*, 201(1-3):81–88, 1999.
- [BGR08] Nayantara Bhatnagar, Sam Greenberg, and Dana Randall. Sampling stable marriages: Why spouse-swapping won’t work. In *Proceedings of the 19th ACM Symposium on Discrete Algorithms*, pages 1223–1232, 2008.
- [CCM10] Prasad Chebolu, Mary Cryan, and Russell Martin. Exact counting of Euler tours for generalized series-parallel graphs. *CoRR*, abs/1005.3477, 2010.

- [CDR10] Mary Cryan, Martin Dyer, and Dana Randall. Approximately counting integral flows and cell-bounded contingency tables. *SIAM Journal on Computing*, 39(7):2683–2703, 2010.
- [Coo73] Stephen Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, 1973.
- [DW12] Anindya De and Thomas Watson. Extractors and lower bounds for locally samplable sources. *ACM Transactions on Computation Theory*, 4(1), 2012.
- [FS04] Lance Fortnow and Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 316–324, 2004.
- [FS11] Lance Fortnow and Rahul Santhanam. Robust simulations and significant separations. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming*, pages 569–580, 2011.
- [FST05] Lance Fortnow, Rahul Santhanam, and Luca Trevisan. Hierarchies for semantic classes. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 348–355, 2005.
- [FV07] Alan Frieze and Eric Vigoda. A survey on the use of Markov chains to randomly sample colorings. In *Combinatorics, Complexity, and Chance*. Oxford University Press, 2007.
- [GGN10] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM Journal on Computing*, 39(7):2761–2822, 2010.
- [GHP05] Dima Grigoriev, Edward Hirsch, and Konstantin Pervyshev. Time hierarchies for cryptographic function inversion with advice. Technical Report TR05-076, Electronic Colloquium on Computational Complexity, 2005.
- [GST11] Oded Goldreich, Madhu Sudan, and Luca Trevisan. From logarithmic advice to single-bit advice. *Studies in Complexity and Cryptography*, pages 109–113, 2011.
- [HS65] Juris Hartmanis and Richard Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.
- [Its10] Dmitry Itsykson. Structural complexity of AvgBPP. *Annals of Pure and Applied Logic*, 162(3):213–223, 2010.
- [JS89] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.
- [JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004.
- [Kal03] Adam Kalai. Generating random factored numbers, easily. *Journal of Cryptology*, 16(4):287–289, 2003.

- [KLM89] Richard Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.
- [KTV99] Ravi Kannan, Prasad Tetali, and Santosh Vempala. Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Structures and Algorithms*, 14(4):293–308, 1999.
- [KvM10] Jeff Kinne and Dieter van Melkebeek. Space hierarchy results for randomized and other semantic models. *Computational Complexity*, 19(3):423–475, 2010.
- [LRS01] Michael Luby, Dana Randall, and Alistair Sinclair. Markov chain algorithms for planar lattice structures. *SIAM Journal on Computing*, 31(1):167–192, 2001.
- [LV99] Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Structures and Algorithms*, 15(3-4):229–241, 1999.
- [LV11] Shachar Lovett and Emanuele Viola. Bounded-depth circuits cannot sample good codes. In *Proceedings of the 26th IEEE Conference on Computational Complexity*, pages 243–251, 2011.
- [MS04] Ben Morris and Alistair Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM Journal on Computing*, 34(1):195–226, 2004.
- [Per05] Konstantin Pervyshev. Time hierarchies for computations with a bit of advice. Technical Report TR05-054, Electronic Colloquium on Computational Complexity, 2005.
- [Per07] Konstantin Pervyshev. On heuristic time hierarchies. In *Proceedings of the 22nd IEEE Conference on Computational Complexity*, pages 347–358, 2007.
- [PW98] James Propp and David Wilson. How to get a perfectly random sample from a generic Markov chain and generate a random spanning tree of a directed graph. *Journal of Algorithms*, 27(2):170–217, 1998.
- [SFM78] Joel Seiferas, Michael Fischer, and Albert Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, 1978.
- [Vig01] Eric Vigoda. A note on the Glauber dynamics for sampling independent sets. *Electronic Journal of Combinatorics*, 8(1), 2001.
- [Vio11] Emanuele Viola. Extractors for circuit sources. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science*, pages 220–229, 2011.
- [Vio12] Emanuele Viola. The complexity of distributions. *SIAM Journal on Computing*, 41(1):191–218, 2012.
- [vMP07] Dieter van Melkebeek and Konstantin Pervyshev. A generic time hierarchy with one bit of advice. *Computational Complexity*, 16(2):139–179, 2007.
- [Wil96] David Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 296–303, 1996.

- [Wil04] David Wilson. Mixing times of lozenge tiling and card shuffling Markov chains. *Annals of Applied Probability*, 14(1):274–325, 2004.
- [Žák83] Stanislav Žák. A Turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983.