Time Hierarchies for Sampling Distributions

Thomas Watson∗

June 24, 2012

Abstract

We show that “a little more time gives a lot more power to sampling algorithms.” We prove that for every constant $k \geq 2$, every polynomial time bound $t$, and every polynomially small $\epsilon$, 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 $t$. This implies the following general time hierarchy for sampling distributions on arbitrary-size domains such as $\{0, 1\}$: For every polynomial time bound $t$ and every constant $\epsilon > 0$, there exists a family of distributions that can be sampled exactly in polynomial time but cannot be sampled within statistical distance $1 - \epsilon$ in time $t$.

Our proof involves reducing the problem to a communication problem over a certain type of noisy channel. To solve the latter problem we use a 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. This type of code can be constructed using certain known traditional list-decodable codes, but we give a new construction that is elementary, self-contained, and tailored to this setting.

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 [CCM12], stable marriages [BGR08], words from context-free languages [GJK+97], chemical isomers [GJ99], contingency tables [KTV99, CDR10, and references within], and spanning trees [PW98, Wil99, 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, LV12, DW12, Vio11, BIL12].

∗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.
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 problems. First, observe that there is a trivial time hierarchy for exact sampling: In time \( t \), an algorithm can produce a particular 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, showing that algorithms running in a sufficiently smaller amount of time cannot sample the distribution within a statistical distance that is any constant less than 1. This is a corollary to our main theorem, which is a quantitatively tight result 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. Our results can be summarized as “a little more time gives a lot more power to sampling algorithms.”

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 Results

We start with our definitions. We let \([k] = \{1, \ldots, k\}\) and let \(\mathbb{N}\) denote the set of positive integers. For distributions \(D, D'\) on \([k]\), recall that the statistical distance is defined as \(|D - D'| = \max_{S \subseteq [k]} |\Pr_D(S) - \Pr_{D'}(S)|\). Our results hold for any reasonable uniform model of computation; for concreteness we may assume the model is Turing machines with access to unbiased independent coin flips.

For a function \(k : \mathbb{N} \to \mathbb{N}\), we define a \(k\)-family to be a sequence \(D = (D_1, D_2, D_3, \ldots)\) where \(D_n\) is a distribution on \([k(n)]\). For a function \(\delta : \mathbb{N} \to [0, 1]\), we say a randomized algorithm \(A\) \(\delta\)-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 \(t : \mathbb{N} \to \mathbb{N}\), 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.\(^1\) We define

\[
\text{SampTime}_{k, \delta}(t)
\]

\(^1\)We measure the running time here as a function of the value of the input, not the bit length of the input. Alternatively, we could view the input as the string \(1^n\) and measure the running time as a function of the bit length.
to be the class of $k$-families $\delta$-sampled by algorithms running in time $t$.\footnote{If we write something such as $\text{SAMPTime}_{O(\log n), 1/2−1/poly(n)}(\text{poly}(n))$, we formally mean the union of $\text{SAMPTime}_{k, \delta}(t)$ over all functions of the form $k = O(\log n)$, $\delta = 1/2 − 1/poly(n)$, and $t = poly(n)$.}

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

\[ \text{SAMPTime}_{k, 0}(\text{poly}(n)) \not\subseteq \text{SAMPTime}_{k, 1−1/k−\epsilon}(t) \]

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

**Corollary 1.** For every function $k(n) \geq \omega(1)$ and every constant $c \geq 1$,

\[ \text{SAMPTime}_{k, 0}(\text{poly}(n)) \not\subseteq \text{SAMPTime}_{k, 1−\epsilon}(t) \]

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

### 1.2 Discussion

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 $n$ to distributions on $[k(n)]$ (for all $n$). This would more realistically model algorithmic sampling problems, but our hierarchy results are stronger with the unary definition. Also, in average-case complexity (see [BT06]), unary sampling arises naturally: The random input to an algorithm is often modeled as coming from an efficiently samplable distribution on $\{0, 1\}^n$ (or $[2^n]$, in our notation) for all $n$. This can be viewed as a secondary motivation for our results.

For Corollary 1 it may seem like it would be cleaner to omit the domain size $k$ from the complexity classes and just say, for example, that the domain is always $\{0, 1\}^*$. However, this would make the corollary true for trivial reasons: A poly($n$)-time samplable distribution could be supported on bit strings of length $> t(n)$, whereas a $t$-time samplable distribution must be supported on bit strings of length $\leq t(n)$. Corollary 1 is only meaningful when the domain size is at most $2^k$.

We now explain how Theorem 1 implies Corollary 1 by the contrapositive. Supposing $k(n)$ and $c$ are a counterexample to Corollary 1, we claim that $k'$ and $c'$ are a counterexample to Theorem 1 where $k' = c' = c + 1$. A family $D \in \text{SAMPTime}_{k', 0}(\text{poly}(n))$ can be viewed as being in $\text{SAMPTime}_{k, 0}(\text{poly}(n))$ (since $[k'] \subseteq [k(n)]$ for all but finitely many $n$) and thus also in $\text{SAMPTime}_{k, 1−1/c}(n^c)$. Now to get a $\text{SAMPTime}_{k', 1−1/k'−\epsilon}(n^c)$ algorithm for $D$, we just run the $\text{SAMPTime}_{k, 1−1/c}(n^c)$ algorithm except that if it outputs a value $> k'$ then we output $k'$ instead. This modification does not cause the statistical distance to go up. Thus the new algorithm runs in time $n^c + O(1)$, and for all but finitely many $n$ it samples a distribution within statistical distance $1 − 1/c \leq 1 − 1/k' − 1/n^c$ from $D_n$.

Note that the $1 − 1/k − \epsilon$ statistical distance bound in Theorem 1 is tight since the theorem becomes false if $\epsilon = 0$. This is because the uniform distribution (which is samplable in constant time) is within statistical distance $1 − 1/k$ from every distribution on $[k]$. We 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 $\epsilon$ such that $2^{\epsilon(n)} \leq t(2^{\text{poly}(n)})$, we have $\text{SAMPTime}_{k, 0}(\text{poly}(t(\text{poly}(n))/\epsilon(\text{poly}(n)))) \not\subseteq \text{SAMPTime}_{k, 1−1/k−\epsilon}(t)$. Finally, we mention that...
our proofs of Theorem 1 and Corollary 1 relativize, and that they carry 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 for Theorem 1

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. In our setting, Brouwer’s fixed point theorem gives a barrier to using this “direct complementation” strategy: 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 \([k]\), 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. Another potential way to bypass the fixed point barrier would be to argue that \( A_i(n_i) \) cannot sample anything close to a fixed point, but it is not clear how to make this approach work.

Since a straightforward direct complementation does not 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 \( n \in \{n_i, \ldots, n_{i+1} - 2\} \) 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 \( n_i \). Thus \( A_i \) cannot agree with the diagonalizing algorithm for all \( n \in \{n_i, \ldots, n_{i+1} - 1\} \) 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 \( D = (D_1, D_2, \ldots) \) to denote the \( k \)-family 0-sampled by \( A_i \) (the algorithm we are diagonalizing against) and \( D^* = (D_1^*, D_2^*, \ldots) \) to denote the \( k \)-family 0-sampled by our diagonalizing algorithm. We can let \( D_{n+1}^* \) be concentrated entirely on the least likely outcome of \( D_n \), say \( M \in [2] \). Now for \( n = (n_i - 2), \ldots, n_i \), by induction we may assume that \( \Pr_{D_n^*}(M) \geq 1 - \epsilon(n+1)/2 \) and thus \( \Pr_{D_n^*}(M) \geq 1/2 + \epsilon(n+1)/2 \) (assuming \( |D_{n+1} - D_{n+1}^*| \leq 1/2 - \epsilon(n+1) \)). 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 \( \Pr_{D_n^*}(M) \geq 1 - \epsilon(n_i)/2 \) while \( \Pr_{D_n^*}(M) \leq 1/2 \), which gives a contradiction if \( |D_{n_i} - D_{n_i}^*| \leq 1/2 - \epsilon(n_i) \).

---

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”.

---
This simple argument breaks down when $k \geq 3$. Suppose we let $D_{n_{i+1}}$ be concentrated on $M \in [k]$, the least likely outcome of $D_{n_i}$. If $D_{n_{i+1}}$ is uniform on $\{1, \ldots, k-1\}$ then this would be consistent with any $M \in \{1, \ldots, k-1\}$, since $D_{n_{i+1}}$ 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 $D_{n_{i+1}}$ 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 $n_i, n_i + 1, \ldots, n_i + m - 1$ and let $M_a$ be the least likely outcome of $D_{n_i + \alpha}$ for $\alpha \in \{0, 1, \ldots, m-1\}$. Suppose that on these inputs, our diagonalizing algorithm could somehow obtain (with high probability) a list of $m$ candidates for the sequence $M_0, M_1, \ldots, M_{m-1}$, where at least one candidate is correct. Then we could have $D^*_{n_i + \alpha}$ but most of its probability mass on the $\alpha$th value from the $\alpha$th candidate sequence. If the $\alpha$th candidate sequence is the correct one, then we get $\Pr_{D^*_{n_i+\alpha}}(M_a) \geq 1 - \epsilon(n_i + \alpha)/2$ while $\Pr_{D^*_{n_i+\alpha}}(M_a) \leq 1/k$, 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 $M_0, M_1, \ldots, M_{m-1}$ in some way as $\gamma \in [k]^\ell$ and use a block of $\ell$ inputs $n_i^*, n_i^* + 1, \ldots, n_i^* + \ell - 1$ to “declare” the codeword $\gamma$, by having $D^*_{n_i^*+j-1}$ be concentrated entirely on $\gamma_j$ for $j \in [\ell]$. 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 $n_i, n_i+1, \ldots, n_i+m-1$).

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 $\ell$ 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 $\ell$ 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. 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 inputs $n_i, n_i + 1, \ldots, n_i + m - 1$ are the leaves of the tree. The final overall list corresponds to these leaves; input $n_i + \alpha$ would get the $\alpha$th message of the overall list. So $\alpha$ specifies a path down the tree, and there must be some path along which the original message is faithfully transmitted.

---

5. For notational reasons, it turns out to be more convenient for us to use 0-based indexing for the sequence of $M_\alpha$’s and 1-based indexing for the coordinates of the codeword $\gamma$.

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

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 $n, \ldots, n + \ell - 1$. 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 $A_i \in (1 - 1/k - \epsilon)$-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 $\rho_j$ with high probability, since under $D_{n+j-1}$ some symbols might occur with probability very close to the threshold used in defining $\rho_j$. 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 a construction of a list-decodable error-correcting code for this non-traditional setting (which is related to the notion of “list-recovery” from the list-decoding literature). Our list-decoder is deterministic, but since $\rho$ is random, the list of messages $T \subseteq S$ where $S$ is the list

---

7We actually only need a branching factor of $(k - 1)^2$, but for simplicity we round it up to $k^2$ in the proof.
of messages for the received word $[k] \setminus \kappa_1 \cdots [k] \setminus \kappa_\ell$.

By a fairly simple reduction to the traditional setting of list-decoding, one can use certain known constructions (such as [GI03]) to handle our non-traditional setting. We provide a direct, self-contained construction which is tailored to this setting and is much simpler than the known traditional constructions.

We now discuss our code construction. The codeword for a message is defined by interpreting the message as a bit string and evaluating all possible surjections $f : \{0, 1\}^{k-1} \to [k]$ 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

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 $k \geq 3$. We start by describing a few ingredients we use in the proof.

We need a construction of a code for the following model of error-correction. Codewords are length-$\ell$ 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 word $\rho \in (P([k]))^\ell$ if $\gamma_j \subseteq \rho_j$ 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: $\rho_j$ 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”. We give an elementary construction of a list-decodable code for this setting.

**Theorem 2.** For every constant $k \geq 3$ there exists a polynomial-time encodable code $C : \{0, 1\}^m \to [k]^\ell$ where $\ell = \Theta(m^{k-1})$ such that the following holds. For every received word $\rho \in (P([k]))^\ell$ 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 $\rho$; moreover, the list of all such $\mu$ can be found in polynomial time given $\rho$.

As mentioned in Section 2, an alternative version of Theorem 2 (that is adequate for our purpose) can be derived from sophisticated off-the-shelf components (specifically, from [GI03]). In Section 4 we give a thorough discussion of the above model of error-correction, describe the alternative construction, and give our self-contained proof of Theorem 2.

Now let $A_1, A_2, A_3, \ldots$ be an enumeration of all randomized algorithms that run in time $t$ and always output an element of $[k]$. We use a procedure Estimate($A_i, n, \zeta, \eta$) which returns a vector $(\pi_1, \pi_2, \ldots, \pi_k) \in [0, 1]^k$ 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 $1$, $\pi_1 + \pi_2 + \cdots + \pi_k = 1$.

---

8 In the formal proof we actually use $m$ to denote the bit length of the message, rather than the length of the sequence over $[k]$.
9 Recall that $P([k])$ denotes the power set of $[k]$. 

---

7
Define $d = \lceil \max(2^c, 3k \log_2 k) \rceil$

For $i \in \mathbb{N}$ define:

$$
n_i = \begin{cases} 
\text{a sufficiently large constant power of 2} & \text{if } i = 1 \\
(n_i^{d-1}) + 1 & \text{if } i > 1
\end{cases}
$$

$$
m_i = \lceil \log_2 k^{(k^2)\log_2 n_i} \rceil
$$

$$
l_i = \Theta(m_i^{k-1}), \text{ the codeword length from Theorem 2 for messages of length } m_i
$$

$$
C_i = \{0, 1\}^{m_i} \rightarrow [k]^{l_i}, \text{ the code from Theorem 2}
$$

$$
Dec_i = \text{the list-decoder from Theorem 2}
$$

For $i \in \mathbb{N}$, $b \in \{0, 1, \ldots, \log_2 n_i\}$, $\alpha \in \{0, 1, \ldots, (k^2)(\log_2 n_i)^{-b} - 1\}$, $j \in [l_i]$ define:

$$
n_{i,b} = n_i^{b^k}
$$

$$
n_{i,b,\alpha} = \begin{cases} 
n_{i,b} + \alpha & \text{if } b = 0 \\
n_{i,b} + \alpha l_i & \text{if } b > 0
\end{cases}
$$

$$
n_{i,b,\alpha,j} = \begin{cases} 
\text{undefined} & \text{if } b = 0 \\
n_{i,b,\alpha} + j - 1 & \text{if } b > 0
\end{cases}
$$

$$
N_{i,b,\alpha} = \begin{cases} 
\{n_{i,b,\alpha}\} & \text{if } b = 0 \\
\{n_{i,b,\alpha,1}, \ldots, n_{i,b,\alpha,l_i}\} & \text{if } b > 0
\end{cases}
$$

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 $O(t(n) \cdot \frac{1}{\zeta} \log \frac{1}{\eta})$ by simulating $A_i(n)$ $O\left(\frac{1}{\zeta^2} \log \frac{1}{\eta} \right)$ times and taking the empirical distribution. 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 $D^* = (D_1^*, D_2^*, \ldots)$, 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 $D^* \not\in \text{SAMPTime}_{k,1-1/k-\epsilon}(t)$. Suppose for contradiction there exists an $i$ such that $A_i (1 - 1/k - \epsilon)$-samples $D^*$. Let $D = (D_1, D_2, \ldots)$ 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 $\log_2 n_i$. 

\footnote{We are ignoring the logarithmic factor time overhead usually associated with simulating an algorithm using a universal algorithm.}
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 \hspace{1em} foreach \( \alpha' \in \{0, 1, \ldots, (k^2)^{\log_2 n_i} - 1\} \) do
5 \hspace{2em} let \( (\pi_{\alpha'}^0, \ldots, \pi_{k}^{\alpha'}) = \text{Estimate}(A_i, n_{i, 0, \alpha'}, 0, 0) \)
6 \hspace{2em} let \( M_{\alpha'} = \arg \min_{\kappa \in [k]} (\pi_{\kappa}^{\alpha'}) \) (breaking ties arbitrarily)
7 \hspace{1em} end
8 convert the sequence \( M_0, M_1, \ldots, M_{(k^2)^{\log_2 n_i} - 1} \) 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 \( \alpha_{\tau} \in \{0, 1, \ldots, k^2 - 1\} \)
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 \( Q = (1/\epsilon(n_{i, b+1}))^{\delta_{q+2}} \)
14 foreach \( r \in [Q] \) do
15 \hspace{1em} foreach \( j' \in [\ell_i] \) do
16 \hspace{2em} let \( n' = n_{i, b+1, \alpha', j'} \)
17 \hspace{2em} let \( (\pi_j^0, \ldots, \pi_k^j) = \text{Estimate}(A_i, n', \epsilon(n')/4, \eta) \) where \( \eta = \epsilon(n_{i, b+1})/4\ell_i Q \)
18 \hspace{2em} let \( \rho_{j'} = \{ \kappa \in [k] : \pi_{\kappa}^{j'} \geq 1/k + \epsilon(n')/4 \} \)
19 \hspace{1em} end
20 \hspace{1em} 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 \( h \)th smallest element of \( T_1 \cap \cdots \cap T_Q \)
24 end
25 if \( b = 0 \) then
26 convert \( \mu \) to a sequence \( M_0, M_1, \ldots, M_{(k^2)^{\log_2 n_i} - 1} \) 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
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 $\ell_i$ inputs, which represent the coordinates of codewords under the code $C_i$. Level $b$ of the tree starts at input $n_{i,b} = n_{i,0}^b$. There are $(k^2)^{(\log_2 n_i) - b}$ nodes across level $b$, indexed by $\alpha \in \{0, 1, \ldots, (k^2)^{\log_2 n_i} - b - 1\}$, and their blocks of inputs $N_{i,b,\alpha}$ are consecutive from left to right across the level. Writing $\alpha$ in base $k^2$ allows us to interpret $\alpha$ as specifying a path down the tree from the root to the current node. The input $n_i$ 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 $i, b, \alpha$ (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 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 $M_0, M_1, \ldots, M_{(k^2)^{\log_2 n_i} - 1}$ 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 2–24 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 $\mu^*$ (generated by the root) is faithfully transmitted.

**Claim 1.** For every $b \in \{0, 1, \ldots, \log_2 n_i\}$ there exists an $\alpha \in \{0, 1, \ldots, (k^2)^{\log_2 n_i} - b - 1\}$ 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 $\mu$ computed in the receiving phase (lines 3–24) equals $\mu^*$ (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 $\text{SAMPTIME}_{k,0}(\text{poly}(n))$. Consider the good $\alpha$ 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 $\mu$ computed in the receiving phase equals $\mu^*$. Thus the sequence $M_0, M_1, \ldots, M_{(k^2)^{\log_2 n_i} - 1}$ 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 $b = \log_2 n_i, \ldots, 0$. The base case $b = \log_2 n_i$ is trivial by the definition of $\mu^*$ (with $\alpha = 0$ and with probability 1, in fact). Now assume $b < \log_2 n_i$ and the claim holds for $b + 1$. Let $\alpha' \in \{0, 1, \ldots, (k^2)^{\log_2 n_i} - b - 1\}$ be the good $\alpha$ 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 $\mu$ computed in the receiving
phase equals $\mu^*$. 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 $q,h \in [k]$ such that $\alpha = (k^2)\alpha' + \alpha_0$ satisfies the desired properties, where $\alpha_0 = (q-1)k + (h-1)$. For any such $\alpha$, suppose $n \in N_{i,b,\alpha}$ and consider Algorithm 1 on input $n$. Note that $\alpha'$ computed on line 12 is indeed the $\alpha'$ 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,

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

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

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

Note that with probability 1 we have $\rho_{j'} \neq [k]$ for all $j'$ and thus $\rho$ 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

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

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 $\rho$, and all codewords consistent with $\rho$ 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 $p_\sigma$ 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 $p_\sigma \geq \exp \left(-\epsilon^*^{4q+4}\right)$ or $p_\sigma < \exp \left(-\epsilon^*^{4q}\right)$ 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

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

and we have either

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

or

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

regardless of which $n \in N_{i,b,\alpha}$ we are considering.\footnote{We can assume without loss of generality that $c$ is large enough in terms of $k$ for these inequalities to hold (recall that $\epsilon(n) = 1/n^c$).}

Defining

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

11
by a union bound over $\sigma \in S$ we find that

$$\Pr \left( (T_1 \cap \cdots \cap T_Q) = T^* \mid E_1 \cap \cdots \cap E_Q \right) \geq 1 - \epsilon(n)/4. \tag{3}$$

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

$$\Pr \left( \text{Algorithm 1 reaches the sending phase with } \mu = \mu^* \right) \geq \Pr \left( (T_1 \cap \cdots \cap T_Q) = T^* \right) \geq \Pr \left( (T_1 \cap \cdots \cap T_Q) = T^* \mid E_1 \cap \cdots \cap E_Q \right) \geq (1 - \eta\ell_1 Q) \cdot (1 - \epsilon(n)/4) \geq 1 - \epsilon(n)/2$$

where the fourth line follows by Inequality (2) and Inequality (3), and the fifth line follows by $\eta\ell_1 Q = \epsilon(n_{i,b+1})/4 \leq \epsilon(n)/4$ (where $\eta$ is as on line 17 of Algorithm 1). This finishes the proof of Claim 1.

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

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

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

**4 List-Decoding from Ubiquitous Informative Errors**

In Section 4.1 we discuss the model of error-correction used in the proof of Theorem 1. Then in Section 4.2 we give our self-contained proof of Theorem 2.

**4.1 Discussion of the Model of Error-Correction**

Recall that in our model of error-correction, the received word is $\rho \in (P([k]))^\ell$ where each $\rho_j \neq [k]$, and the goal is to find the list of all messages whose codeword $\gamma \in [k]^\ell$ is consistent with $\rho$ in the sense that $\gamma_j \in \rho_j$ for all $j \in [\ell]$.

We first remark that in this setting, 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 superset 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.

Our setting is related to the notion of “list-recoverable” codes which has been studied in the list-decoding literature. In list-recovery, each coordinate of the received word is a set of symbols, but there are several differences from our setting. We allow each coordinate of the received word to be as large as possible without becoming an erasure, whereas in list-recovery each coordinate is usually restricted to be a fairly small set. Also, sometimes in list-recovery a small fraction of coordinates of the received word are allowed to violate the size restriction and become erasures. Also, in list-recovery the correct codeword is only guaranteed to agree with many coordinates of the received word, whereas we assume it agrees with all coordinates.

A simple application of the probabilistic method shows that if we drop the requirement that the encoding and list-decoding can be done in polynomial time, then there exist codes for our model with list size \( k - 1 \) (where \( k \) is the alphabet size) and codeword length \( \ell = \Theta(m) \) (where \( m \) is the message length and the hidden constant depends on \( k \)). In other words, 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. 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.

For our application in Theorem 1, we do not need the list size to be \( k - 1 \), as long as it is a constant depending on \( k \). Such codes for our setting follow from certain known constructions of traditional list-decodable codes. Recall that a code is said to be \((\beta, L)\)-list-decodable if for every received word in \([k]^\ell\), there are at most \( L \) codewords at relative Hamming distance \( \leq \beta \), and the list of all such codewords can be found in polynomial time. Every \((1 - 1/(k - 1), L)\)-list-decodable code is also list-decodable under our model with list size \((k - 1)L\): Given a received word \( \rho \in (\mathcal{P}([k]))^\ell \) where each \( |\rho_j| = k - 1 \), we can form new “received words” \( \rho^{(1)}, \ldots, \rho^{(k-1)} \) by letting \( \rho^{(g)} \in [k]^\ell \) consist of the \( g \)th smallest symbol in each coordinate of \( \rho \). Since a codeword consistent with \( \rho \) must have relative Hamming distance \( \leq 1 - 1/(k - 1) \) from some \( \rho^{(g)} \), running the traditional list-decoder on each \( \rho^{(g)} \) will reveal all the codewords consistent with \( \rho \).

For a traditional list-decodable code construction to be used for our application via the above connection, there are several crucial properties it should satisfy: (i) It should work for constant-size alphabets (some constructions only work for large alphabets). (ii) It should work for every constant-size alphabet (some constructions require the alphabet to be a finite field). (iii) The list size should be a constant depending on the alphabet size (some constructions have list size polynomial in the message length).

The construction of Guruswami and Indyk [GI03], which uses expanders and spectral techniques, satisfies all these properties and is \((\beta, L)\)-list-decodable with \( L = O(1/(1 - 1/k - \beta)^3) \) assuming \( \beta < 1 - 1/k \). Taking \( \beta = 1 - 1/(k - 1) \), the list size is \( O(k^6) \), which becomes \( O(k^7) \) after applying the reduction from our setting. The list-decoder is randomized, but that is not a problem for our application in the proof of Theorem 1. Thus the result of [GI03] yields an alternative version of Theorem 2 that is sufficient for our application. This alternative construction has the following advantages: The codeword length is \( \Theta(m) \), and the encoding and list-decoding can be done in linear time. But it has the following disadvantages: The list size is \( O(k^7) \) rather than the optimal \( k - 1 \), the list-decoder is randomized, and the proof is much more complicated than our proof of Theorem 2. Although it is convenient to use this off-the-shelf machinery, our code construction demonstrates that such machinery is overkill and that elementary techniques suffice.

We now mention an interesting contrast between our setting and the traditional error-correction
setting. In the traditional setting, many code constructions are linear (assuming the alphabet is a finite field). In our model of error-correction, linear codes cannot achieve the optimal list size of $k-1$ (where $k$ is the alphabet size). Here is a counterexample. Recall that the property for achieving optimal list size is that for every set of $k$ codewords, there exists a coordinate on which all $k$ symbols appear among those codewords. Suppose the alphabet is $GF(5)$, and let $x_1, x_2, x_3, x_4$ be any linearly independent message vectors, and let $x_5 = 3 \ast x_1 + x_2 + x_3 + x_4$. Then for any given coordinate of the codewords, if $y_1, ..., y_5 \in GF(5)$ are the symbols of the codewords in that coordinate, then they must satisfy $y_5 = 3 \ast y_1 + y_2 + y_3 + y_4$ if the code is linear. It can be verified by brute force that this particular relation over $GF(5)$ forces two of the $y_i$’s to be equal.

4.2 Proof of Theorem 2

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

4.2.1 A Combinatorial Lemma

For a set $S$ and number $a$, we let $(S \atop a)$ 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 $\sigma_i$ denote the $i$th bit of $\sigma$, and we let $\sigma_I$ denote the length-$|I|$ string consisting of the bits of $\sigma$ indexed by $I$.

Lemma 1. For all $1 \leq a \leq b$ and every set of distinct strings $\sigma^1, ..., \sigma^a \in \{0,1\}^b$, there exists an $I \in \binom{[b]}{a-1}$ such that $\sigma^j_I, ..., \sigma^a_I \in \{0,1\}^{a-1}$ 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 $\sigma^1_{I'}, ..., \sigma^{a-1}_{I'}$ are distinct. If $\sigma^a_{I'}$ is different from each of $\sigma^1_{I'}, ..., \sigma^{a-1}_{I'}$ then we can take an arbitrary $I \supseteq I'$ of size $a - 1$. Otherwise, $\sigma^a_{I'} = \sigma^h_{I'}$ for exactly one $h \in [a - 1]$. Since $\sigma^a \neq \sigma^h$, there exists an $i \in [b] \setminus I'$ such that $\sigma^a_i \neq \sigma^h_i$, and we can take $I = I' \cup \{i\}$.

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.

4.2.2 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 $\text{Surj}_k$ 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

$$C(\mu) = (f(\mu_I))_{I \in \binom{[m]}{k-1}, f \in \text{Surj}_k}.$$
Algorithm 2: List-decoder for Theorem 2

| Input: | $\rho \in (\mathcal{P}([k]))^f$ 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 $\rho$ |

1. Let $S_{k-1} = \text{List}(\rho, [k-1])$
2. For each $n = k, \ldots, m$ do
   3. Suppose $S_{n-1} = \{\sigma^1, \ldots, \sigma^{S_{n-1}}\} \subseteq \{0,1\}^{n-1}$
   4. Find an $I \in \binom{[n-1]}{k-2}$ such that $\sigma_I^1, \ldots, \sigma_I^{S_{n-1}}$ are distinct
   5. Let $S_n = \{s \in \{0,1\}^n : s_{[n-1]} \in S_{n-1} \text{ and } s_{I \cup \{n\}} \in \text{List}(\rho, I \cup \{n\})\}$
3. Output the set of all $\mu \in S_n$ such that $C(\mu)$ is consistent with $\rho$

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 $\rho \in (\mathcal{P}([k]))^f$ 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 $\rho$, and that moreover, these messages can be found in polynomial time given $\rho$.

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 $\rho$ and $I$ by trying all possibilities.

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

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

Proof. Consider any set of $k$ distinct strings $\sigma^1, \ldots, \sigma^k \in \{0,1\}^{k-1}$. There exists an $f \in \text{Surj}_{k}$ such that $\{f(\sigma^1), \ldots, f(\sigma^k)\} = [k]$. 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 $\sigma^h \notin \text{List}(\rho, I)$. \qed

Now to see that $C$ is list-decodable in principle, suppose for contradiction that there are $k$ distinct messages $\mu^1, \ldots, \mu^k$ whose codewords are all consistent with $\rho$. Applying Lemma 1 with $a = k$ and $b = m$, there exists an $I \in \binom{[m]}{k-1}$ such that $\mu_I^1, \ldots, \mu_I^k$ 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 $\rho$, there are at most $k - 1$ messages whose codewords are consistent with $\rho$. Algorithm 2 finds this list of messages in polynomial time given $\rho$. The correctness of the algorithm follows immediately from the following claim and line 7 of the algorithm.

Claim 3. For all $n = (k-1), \ldots, m$, 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 $\rho$ we have $\mu_{[n]} \in S_n$.

\footnote{Because 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.}
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 \( s^1, \ldots, s^k \in S_n \). Then since \( |\text{List}(\rho, I \cup \{n\})| \leq k - 1 \) (by Lemma 2) and \( s^h_{I \cup \{n\}} \in \text{List}(\rho, I \cup \{n\}) \) for all \( h \in [k] \), there must exist \( h_1 \neq h_2 \) such that \( s^{h_1}_{I \cup \{n\}} = s^{h_2}_{I \cup \{n\}} \). Since \( s^{h_1}_{[n-1]}, s^{h_2}_{[n-1]} \in S_{n-1} \) and \( s^{h_1}_I = s^{h_2}_I \), we must have \( s^{h_1}_{[n-1]} = s^{h_2}_{[n-1]} = \sigma^h \) for some \( h \). But now \( s^{h_1}_{[n-1]} = s^{h_2}_{[n-1]} \) and \( s^n_h = s^n_h \), 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 \( \rho \). 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 \( \mu_{[n]} \in S_n \).

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 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

In Corollary 1, \( \epsilon \) can be taken to be some unspecified \( o(1) \) function of \( n \). A natural open problem is to investigate how fast \( \epsilon \) can approach 0 for interesting choices of \( k(n) \) such as \( 2^n \). 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. It is also open to see whether something interesting can be said about space hierarchies for sampling distributions.

Another open problem is to obtain explicit constructions of list-decodable codes as in Theorem 2 that simultaneously achieve constant rate (instead of polynomially small rate) and optimal list size \( k - 1 \).

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.

Acknowledgments

I thank Oded Goldreich, Luca Trevisan, and anonymous reviewers for helpful comments.
References


