Linear Time Encodable and List Decodable Codes

(Extended Abstract)

Venkatesan Guruswami
Department of Computer Science & Engineering
University of Washington
Seattle, WA 98195.
venkat@cs.washington.edu

Piotr Indyk
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139.
indyk@theory.lcs.mit.edu

ABSTRACT

We present the first construction of error-correcting codes which can be (list) decoded from a noise fraction arbitrarily close to 1 in *linear time*. Specifically, we present an explicit construction of codes which can be encoded in linear time as well as list decoded in linear time from a fraction $(1-\varepsilon)$ of errors for arbitrary $\varepsilon > 0$. The rate and alphabet size of the construction are constants that depend only on ε. Our construction involves devising a new combinatorial approach to list decoding, in contrast to all previous approaches which relied on the power of decoding algorithms for algebraic codes like Reed-Solomon codes.

Our result implies that it is possible to have, and in fact explicitly specifies, a coding scheme for *arbitrarily large noise thresholds* with only constant redundancy in the encoding and *constant* amount of work (at both the sending and receiving ends) for each bit of information to be communicated. Such a result was known for certain probabilistic error models, and here we show that this is possible under the stronger *adversarial* noise model as well.

1. INTRODUCTION

Constructing efficiently encodable and decodable codes with high error-correction capability is the central problem in coding theory. Since the concept of error-correcting codes was conceived in 40's, there has been an amazing progress towards this goal. In particular, it is now known how to construct efficient codes which can correct a constant fraction of errors, and which can be decoded and encoded in linear time [18, 17]. In fact, this goal can be achieved even for codes with minimum distance arbitrarily close to the length n of the code [10, 11] (and in fact the latter paper achieves linear time encoding and decoding with near-optimal rate). This allows one to identify the original message uniquely, even when the fraction of errors is arbitrarily close to 50%. Note that if the fraction of errors exceeds 50%, unique decoding of the corrupted codeword is impossible.

In a different direction of recent research, it was shown [5, 20, 12] that one can exceed the 50% barrier (and, in fact, correct a fraction of errors arbitrarily close to 100%), if one allows to output a *list* of potential codewords. This approach (called *list decoding*) has been introduced in the late 50's by Elias [7] and Wozencraft [22]. Until recently, however, no efficient algorithms for list decoding were known. In [20, 12] the authors presented the first *polynomial time* algorithms for decoding Reed-Solomon codes from any fraction of errors smaller than 1. The runtime of their algorithm has been subsequently improved (see [8, 1] and the references there-in) leading to (roughly) $O(n \log^2 n)$-time decoding algorithms. The Reed-Solomon decoding algorithm in turn has since been used in various ways, including code concatenation, to construct list decodable codes over fixed alphabets that can correct the maximum information-theoretically possible fraction of errors.

The improvements in the efficiency of algorithms for list-decodable codes suggested that it might be possible to repeat the progress of unique decoding algorithms and construct *linear* time encodable and *list*-decodable codes. Unfortunately, this does not seem likely using current list decoding techniques. This is due to the fact that all codes for which polynomial time list decoding algorithms were known so far (i.e., Reed-Solomon codes and their Algebraic Geometry-based generalizations) are algebraic in nature.[1] Even the algorithm for *unique* decoding of Reed-Solomon codes has running time $\Theta(n \log^2 n)$.[2] At the same time, the codes of [18] cannot be easily modified to correct more than $n/2$ errors, since their minimum distance is much smaller than this bound.

In this paper we depart from the approach used in the aforementioned papers. Instead, we show how to list decode codes constructed using *expanders* [2, 10] using graph partitioning techniques, notably *spectral partitioning*. Our main result is a construction of constant rate, linear-time encodable codes over alphabet size q that are capable of correcting up to $(1 - 1/q - \delta)$ fraction of errors in linear time, for any $\delta > 0$ and integer $q \geq 2$. In particular, by picking a large enough alphabet, we can correct a fraction $(1-\varepsilon)$ of errors

STOC'03, June 9–11, 2003, San Diego, California, USA.

[1] This holds even for the "extractor codes" of [21], since the algorithms for list decoding of those codes use the algorithms of [20, 12] as a black-box. The same comment applies to the codes of [10].

[2] Here and in the remainder of this paper n is used to denote code block length.

for arbitrary $\varepsilon > 0$.

The rate of our codes, though constant, is rather small — it is $2^{-2^{O(\varepsilon^{-3})}}$ if one wants to correct a fraction $(1-\varepsilon)$ of errors. Much better rates (polynomial in ε) are known if one allows near-linear (or certainly quadratic) list decoding times. However, we do expect that the rate of our construction will be improved subsequently, as the understanding of list decoding expander-based codes increases.

In any case, the linear time encoding/decoding time allows us to conclude the following fact: the amount of work per bit needed to communicate information over noisy channel is a *constant*, independent of the message length. We note that Shannon's classical noisy coding theorem showed that a constant amount of redundancy per bit of information suffices to recover from an arbitrary fraction of errors that are effected *probabilistically*. Shannon's work did not deal with efficient encoding or decoding, but with standard concatenated coding techniques that use the Spielman codes [18] (or other linear time encodable/decodable codes that can correct a small fraction of errors), it is known [19] how to get linear time encodable and decodable codes which work with redundancy arbitrarily close to the channel capacity, and in particular with a constant amount of redundancy, for every noise threshold (at least for the binary symmetric channel).

In contrast, our work deals with the *adversarial* noise model. Spielman's work [18] achieved a constant amount of work per bit of information transmitted under adversarial error models. However, the fraction of errors that his codes could correct was very small (about 10^{-6} for binary codes). While this was subsequently improved in [11] to the optimal error-correction radius, this still fell short of handling noise levels beyond 50% due to the inherent limitation of unique decoding. By devising a new approach to list decoding, we are able to construct a coding scheme with constant redundancy and constant amount of work (at both the encoding and decoding ends) per bit of information transmitted, *for an arbitrary noise threshold of adversarially effected errors.*

2. PRELIMINARIES AND DEFINITIONS

Decoding. For any alphabet set Σ, and any two vectors $x, y \in \Sigma^n$, we use $D(x, y)$ to denote the Hamming distance between x and y (i.e., the number of positions on which x and y differ). For a sequence $\mathcal{L}$ of lists $L_1 \dots L_n$ of symbols, and a vector x of n symbols, define $\mathrm{COMP}(x, \mathcal{L})$ to be the number of indices j such that $x_j \in L_j$.

The most general notion of list-decodability used in this paper is as follows. Recall that a code C of block length n over alphabet Σ is simply a subset of Σ^n, and elements of C are called codewords. The (minimum) distance of the code is smallest Hamming distance between a pair of distinct codewords. Such a code C is said to be (α, β, l, L)-*(list) recoverable*, if for any sequence $\mathcal{L}$ of n lists $L_1 \dots L_n$, where $L_j \subseteq \Sigma$, $|L_j| \le l$ for at least $(1-\beta)$ fraction of j and $L_j = \Sigma$ for remaining j's, there are at most L codewords $c \in C$ such that $\mathrm{COMP}(c, \mathcal{L}) \ge \alpha n$. We say that C is (α, β, l, L)-*(list) recoverable* in time $T(n)$, if there is a procedure which finds the list of (at most L) such codewords in time $T(n)$ given the lists $L_1, \dots, L_n$.

We say that a code is (α, l, L)-recoverable, if it is $(\alpha, 0, l, L)$-recoverable. Similarly, we say that a code is (α, L)-(list) decodable if it is $(\alpha, 1, L)$-recoverable. In the context of a fixed codeword $c \in C$, we say that the list L_j (or the j-th position) is *corrupted*, if $c_j \notin L_j$. Otherwise, we say L_j (or the j-th position) is *correct*.

Expanders. All code constructions in this paper use *expanders*. A bipartite graph $G = (A, B, E)$ is an (α, β)-expander, if for any $X \subseteq A$, $|X| \ge \alpha |A|$, the set $\Gamma(X)$ of X's neighbors in B has size at least $\beta |B|$. However, in addition to this notion, we will use more general *isoperimetric* properties of graphs. Specifically, we will make use of the following fact.

FACT 1. *Let $G = (V, E)$ be a d-regular graph on n nodes with second eigenvalue λ. Then the set of edges between a pair of subsets of vertices X and Y, denoted $E(X, Y)$, satisfies the inequality:*

$$\left| \frac{|E(X,Y)|}{d|X|} - \frac{|Y|}{n} \right| \le \frac{\lambda}{d} \sqrt{\frac{|Y|}{|X|}} \ . \tag{1}$$

It is known [15] how to construct graphs (called *Ramanujan graphs*) which achieve $\lambda = 2\sqrt{d-1}$.

In order to convert a graph $G = (V, E)$ into a "similar" bipartite graph $G' = (A, B, E')$, we define the notion of *double cover*: G' is a double cover of G, if $A = V \times \{0\}$, $B = V \times \{1\}$, and $\{i, j\} \in E$ iff $\{(i, 0), (j, 1)\}, \{(i, 1), (j, 0)\} \in E'$.

Expander codes. For the purpose of constructing codes using expanders, we use the following scheme, first proposed in [2] (cf. [10, 11]). Assume we are given a code $C \subseteq \{0 \dots q-1\}^n$, and a bipartite graph $G = (A, B, E)$ with $|A| = |B| = n$ and with right degree d. Given these two components, we construct the code C' in the following way. For any $x \in \{0 \dots q-1\}^n$, define $G(x)$ to be a vector $y \in (\{0 \dots q-1\}^d)^n$ created as follows. For $j \in B$ let $\Gamma_k(j)$ be the k-th neighbor of j in A, for $k = 1 \dots d$. The j-th symbol y_j of y is defined as $\langle x_{\Gamma_1(j)}, \dots, x_{\Gamma_d(j)} \rangle$. In other words, we "send" a copy of each symbol x_i along all edges going out of the vertex i, and the symbol y_j is obtained by concatenating all symbols "received" by j. The code $C' = G(C)$ is now obtained by taking all vectors $G(c)$ for $c \in C$.

It is easy to see [2] that if C has minimum distance αn, and G is an (α, β)-expander, then the minimum distance of C' is at least βn. Thus, the construction has "distance amplification" property. The price for that is the decrease in rate (by a factor $1/d$ compared to the rate of C) and larger alphabet (of size q^d).

Model of computation. The model of computation used in this paper is the *unit cost* RAM. In this model, we allow the algorithm to perform any "reasonable" operation, on $O(\log n)$-bit long words, in unit time. In addition, we allow the algorithm to use indirect addressing (i.e., table lookup); we charge the algorithm for any preprocessing cost needed to compute the lookup table.

The unit-cost RAM is a standard and widely used model of computation. However, other (more restrictive) models of computation are known. In particular, the results of [18] hold even in so-called *logarithmic-cost* RAM, i.e., where the cost of an operation is linear in the bit-length of its arguments. Our algorithms can be modified to achieve $O(n)$ time in such models as well, we leave details to the full version of this paper.

Notation. We will use $\|x\|_p$ to denote the l_p norm of a vector x. Also, for a square matrix A, we use $\|A\|_p$ to denote the l_p norm of A, i.e., $\max_{\|x\|_p=1} \|Ax\|_p$. Throughout, we will be thinking of vectors x as column vectors.

3. WARMUP

3.1 List decodable codes from list recoverable codes

Much of our technical work lies in Section 4 where we construct a family of $(1-\varepsilon_l, l, l)$-*recoverable* codes for every l and some positive ε_l. Our final goal is a family of $(\delta, O(1))$-*decodable* codes for every $\delta > 0$. We now highlight how one can get list-decodable codes from list-recoverable codes using suitable expander graphs. Such a reduction has already been used in [10] (cf. [9, Chap. 9]), but the "voting scheme" used in the implementation here is slightly different.

LEMMA 1. *Let C be a code of block length n and rate r over alphabet Σ. Also suppose that C is $(1-\varepsilon', l, L)$-decodable in time $T(n)$. Further assume that there exists a d-regular Ramanujan graph G on n vertices for $d \geq \frac{4}{\varepsilon'\varepsilon^2}$. Then there exists a code C' of block length n and rate r/d over alphabet Σ^d which is explicitly specified given C, G, and which is $(\frac{1}{l}+\varepsilon, L)$-decodable in time $O(T(n)+n)$. Furthermore, C' is linear-time encodable if C is.*

Proof: The code C' will just be $G(C)$. The claims about block length, rate, alphabet size, and encoding time follow immediately. Let $G = (A, B, E)$. Since G is Ramanujan, its second largest eigenvalue satisfies $\lambda \leq 2\sqrt{d}$, and by the isoperimetric property (Fact 1), we can show that, if the degree d is large enough, then G has the following property:

(**) for any $S \subseteq B$, $|S| \geq (\frac{1}{l}+\varepsilon)|B|$, the fraction of $i \in A$ which have at most a fraction $\frac{1}{l}$ of their neighbors inside the set S is at most ε'.

The list decoding algorithm for C' proceeds as follows. Given a string $y \in (\Sigma^d)^n$, for each $i \in A$, create a list $L_i \subseteq \Sigma$ consisting of the l most frequent elements in the multiset T_i of elements corresponding to $i \in A$ in the symbols y_j for neighbors j of i: formally $T_i = \{a_k \mid (i,j) \in E;\ y_j = \langle a_1, a_2, \ldots, a_d\rangle;\ \Gamma_k(j) = i\}$.

Let $c' \in C'$ be a codeword such that $c'_j = y_j$ for at least $(\frac{1}{l}+\varepsilon)n$ of the positions j. Let $c \in C$ be the codeword for which $c' = G(c)$. By the Property (**), we must have $c_i \in L_i$ for at least a fraction $(1-\varepsilon')$ of the lists L_i. Thus we can find c, and hence c', by running the $(1-\varepsilon', l, L)$-recovering algorithm for C on input the lists L_i. The output list size is at most L, and the running time is $T(n)$ plus the time to compute the lists, which is at most $O(nd) = O(n)$ word operations (since d is a fixed constant depending only on $\varepsilon, \varepsilon'$). □

3.2 Codes that are $(1,2,2)$-recoverable in linear time

With Lemma 1 in mind, our real technical goal is the construction of $(1-\varepsilon_l, l, L)$-recoverable codes for some $\varepsilon_l > 0$. Before we proceed with this task in the next section, we first present a much simpler construction, that nevertheless illustrates our methods and tools. Specifically, we show how to construct a constant-rate code C' that is $(1,2,2)$-recoverable in linear time. In other words, C' has the property that given a sequence of lists $\mathcal{L} = L_1 \ldots L_n$, where each list contains at most two characters, we can compute in linear time all (at most 2) codewords $c' \in C'$ such that $c'_i \in L_i$ for all $i = 1 \ldots n$. This is perhaps the simplest non-trivial list-recovering setting. It has been investigated earlier in [6] in the context of cryptanalysis of certain code-based cryptosystems.

In order to construct C', we need a "base" code $C \subseteq \Sigma^n$ and a bipartite graph $G = (A, B, E)$, such that:

- C can be decoded from 90% of erasures (no errors) in linear time. Such code, with constant rate, has been provided e.g., in [4].
- The graph $G = (A, B, E)$ has the following *fault tolerance* property: if we delete $n/10$ nodes in A, then the square of the remaining graph has a connected component containing more than, say, $n/3$ nodes in A. It was shown in [3] that such a graph can be obtained by taking a double cover of a Ramanujan graph with sufficiently high (constant) degree d.

The code C' is defined as $G(C)$. A simple counting argument using the large minimum distance of C', it is easy to show that $G(C)$ is *combinatorially* $(1,2,2)$-decodable. It remains to show that given lists $L_1 \ldots L_n$, we can reconstruct the list of at most codewords $c \in C$ such that $G(c)_j \in L_j$, $j = 1 \ldots n$, in linear time. For each $i \in A$, $j \in B$, we define $L(i,j)$ to be the set of symbols in Σ that L_j "suggests" as potential symbols for c_i. More formally, $L(i,j)$ contains symbols a_k, such that $\langle a_1, \ldots a_d\rangle \in L_j$ and $\Gamma_k(j) = i$.

Define $K_i = \cap_{\{i,j\}\in E} L(i,j)$. We assume that all K_i's are non-empty, since otherwise no codeword compatible with $\mathcal{L}$ exists. Define I to be the set of indices $i \in A$ such that K_i has size 1. For such i's, denote the symbol in K_i by x_i.

Fix a codeword c such that $G(c)_i \in L_j$, $j = 1 \ldots n$. Clearly, for all $i \in I$, we must have $c_i = x_i$. The decoding procedure consists of two cases. The first case occurs when the set I has size at least $n/10$. In this case, we know at least 10% of symbols of c, and thus we can recover c using the linear-time erasure-decoding algorithm for the code C. It remains to consider the second case, when the size of the set I is smaller than $n/10$. Consider any $i \notin I$. Observe that, for all $\{i,j\} \in E$, all sets $L(i,j)$ must be equal to K_i. The set K_i contains two distinct symbols that are candidates for c_i. Note that although for each c only one of this symbols is correct, each symbol in K_i can be correct for *some* codeword c. From now on, we will consider each K_i (resp. L_j) as ordered sequences (not sets) of two symbols $(K_i)_0$ and $(K_i)_1$ (resp. $(L_j)_0$ and $(L_j)_1$).

We need to recover c from K_i's in a "consistent" manner. To this end, we create an auxiliary graph HH. Intuitively, the graph HH is obtained by "splitting" each node in G into two nodes, each corresponding to two decoding options, and then putting edges between compatible options. Formally, $HH = (A', B', E')$ is a bipartite graph such that $A' = A \times \{0,1\}$ and $B' = B \times \{0,1\}$. For each $i \notin I$, and $\{i,j\} \in E$, the edge set E' contains edges $\{(i,0),(j,t)\}$ and $\{(i,1),(j,1-t)\}$, where $t \in \{0,1\}$ is such that

$$(K_i)_0 = ((L_j)_t)_k$$

where k is such that $\Gamma_k(j) = i$.

Let H be the square of HH. Since HH is bipartite, all edges in H are either between two vertices in A' or two vertices in B'.

Define $V(c) = \{(i,t) : i \notin I, c_i = (K_i)_t\}$. In other words, $V(c)$ contains the nodes in A' that are compatible with c. The key fact, easily proved by induction, is that if $(i,t) \in V(c)$, and $(i',t') \in A'$ is connected to (i,t) in H, then $(i',t') \in V(c)$. Moreover, the induced subgraph

$H_{|V(c)}$ is isomorphic to $G^2_{|(A-I)}$, and thus contains a connected component $S \subseteq V(c)$ of size at least $n/3$. Thus, if we enumerate all large connected components of H, we will find a large subset S' of $V(c)$. Given S', we can recover c from a vector x such that x_i is equal to $(K_i)_t$ if $(i,t) \in S'$, and is declared as an erasure otherwise.

Since the graph H has only $O(n)$ edges, the set S' can be found in $O(n)$ time. Thus, the whole decoding process can be performed in linear time. In addition, encoding C' can be done in linear time as well if C is linear-time encodable.

The remainder of this paper is as follows. In the next section we show how to extend the above construction and decoding algorithm to obtain codes that are $(1-\varepsilon_l, l, l)$-recoverable, for any constant l and some constant $\varepsilon_l > 0$. The decoding algorithm runs in $O(n \log n)$ time. Then in section 5 we show how to use such codes as building blocks to construct codes that are (α, L)-decodable in $O(n)$ time, for arbitrarily small α.

The most technically involved part is the construction of (and the decoding algorithm for) $(1-\varepsilon_l, l, l)$-recoverable codes. Although on the high-level it is similar to the construction in this section, for $\varepsilon_l > 0$, the graph H will contain "wrong" edges, e.g., between $V(c)$ and $V(c')$ for two different codewords c and c'. Thus, we no longer can retrieve $V(c)$ by computing connected components of H. Instead, we need to retrieve "dense" components of H with few inter-component edges between them, which complicates matters.

4. $(1-\varepsilon, L, L)$-LIST RECOVERABLE CODES

In this section we describe how to construct for each $l \geq 1$, an $(1-\varepsilon_l, l, L)$-recoverable code which can be encoded and list-recovered in $O(n \log n)$ time. Note that ε_l will depend on l, and in particular ε_l must be very small (around $1/2^{2^{l^{O(1)}}}$) for the construction to work. The output list size L will actually be l (note that it has to be at least l since we can give l codewords as the input lists without any error). If the relative distance of the code is large enough (greater than $(1-1/(l+1)^2+\varepsilon_l)$ for example), as will be the case in our construction, a straightforward combinatorial argument shows that the number of codewords consistent with the input lists at a fraction $(1-\varepsilon_l)$ or more of positions will be at most l. The same argument also shows that if α is small enough, then such a code will also be $(1-\varepsilon_l, \alpha, l, l)$-recoverable.

4.1 The Construction

When $l = 1$, we just need a linear time code to correct a fraction ε_1 of errors, and we know such constructions for any ε_1 strictly less than $1/4$ [11]. For concreteness, let us pick a construction over the binary alphabet with $\varepsilon_1 = 1/5$.

Our construction of a $(1-\varepsilon_l, l, l)$-recoverable code will be recursive, i.e. assuming we have a construction of such a code C_l, we will construct a $(1-\varepsilon_{l+1}, l+1, l+1)$-recoverable code C_{l+1}. We will lose a further factor in the rate and ε_{l+1} will become smaller still, but the overall code will have positive rate, and will have linear time encoding and list recovering algorithms. In order for our recursive construction to work, we will need to allow a small fraction α of erasures in both C_l and C_{l+1}.

We now describe this construction. Let C_l be a code over alphabet size Q_l with rate r_l and block length n. Assume that C_l can be encoded in linear time and can be $(1-\varepsilon_l, \alpha, l, l)$-list recovered (for some small ε_l and α which we will specify explicitly once we analyze the recursive construction) also in linear time. To construct C_{l+1} which will be $(1-\varepsilon_{l+1}, \alpha, l+1, l+1)$-recoverable, we need two expander graphs:

- an $n \times n$ bipartite graph G_1 with degree d_1, such every set of αn left nodes has at least $(1-\alpha)n$ neighbors on the right side. (This is the "erasure-reducing" graph.) Note that there are explicit graphs with this property with $d_1 = O(1/\alpha^2)$.
- a bipartite graph G_2 that is a double cover of a Ramanujan graph $G_R = (V_R, E_R)$ on n vertices with degree d_2. The graph G_2 also has the following *weak expansion property*: every set of αn left nodes has more than αn neighbors on the right side. The graph G_2 will be used for spectral partitioning (and its being Ramanujan as opposed to just a good vertex expander is crucial for this purpose).

Then we construct C_{l+1} as $C_{l+1} = G_2(G_1(C_l))$. Therefore the rate of C_{l+1} is $r_l/d_1 d_2$ and its alphabet is $\Sigma_{l+1} = \Sigma_l^{d_1 d_2}$. Let $\Sigma = \Sigma_l^{d_1}$ denote the alphabet of the "intermediate" code $G_1(C_l)$.

By the construction, it is clear that the following two properties of C_{l+1} can be ensured:

1. C_{l+1} can be unique decoded in linear time from a fraction $1/5$ of errors. This is true already for C_l and it is obvious how to do the same for C_{l+1}, namely do a majority voting using G_2 and then G_1. This will result in a received word for C_l with at most a fraction $1/5$ of errors, which can be corrected using the assumed algorithm for C_l.
2. The relative distance of C_{l+1} is large enough to ensure that it is combinatorially $(1-\varepsilon_{l+1}, \alpha, l+1, l+1)$-recoverable, i.e., the output list size only needs to be $l+1$.

4.2 The Decoding

Let $\mathcal{L}$ be a collection of n lists $L_1, L_2, \ldots, L_n$ of which at most αn equal Σ_{l+1} (i.e. are erasures) and remaining have size at most $l+1$ each. Our goal is to output a list of all codewords c in C_{l+1} for which $c_i \in L_i$ for all but $\varepsilon_{l+1} n$ values of i.

Consider the way symbols are distributed by G_2. For each edge (i,j) in G_2, let $L_2(i,j) \subseteq \Sigma$ denote the set of symbols that L_j "suggests" for the i-th symbol of the left codeword (see the formal definition in the warmup section). Note that $L_2(i,j)$ is a set, so duplicates are removed. For each i (left node of G_2), define $K_i = \cap_{\{i,j\} \in E} L_2(i,j)$.

Let I be the set of indices i such that $|K_i| \leq l$.

Case 1: $|I| > \alpha n$.

Let T be the set of left nodes of G_1 which have at least one neighbor in I. By the expansion property of G_1, we have $|T| \geq (1-\alpha)n$. For each $i' \in T$, define $\tilde{L}_{i'}$ to be the symbols corresponding to position i' in the list K_i where $i \in I$ is an arbitrary node for which (i', i) is an edge in G_1. Note that each $\tilde{L}_{i'}$ has at most l elements. For $i' \notin T$, define $\tilde{L}_{i'}$ to be Σ_l, the alphabet of the code C_l. We are now in a position to complete the decoding using the algorithm for C_l, since at most a fraction α of positions are erased (i.e. have $\tilde{L}_{i'} = \Sigma_l$), and the number of corrupted lists (i.e. whose l elements do not include the corresponding codeword symbol) among the $\tilde{L}_{i'}$'s is at most $\varepsilon_{l+1} d_1 d_2 n$. Thus if $\varepsilon_{l+1} \leq \frac{\varepsilon_l}{d_1 d_2}$, we can complete the decoding using the $(1-\varepsilon_l, \alpha, l, l)$-list recovering algorithm for C_l applied to the lists $\tilde{L}_{i'}$.

Note that the strong expansion property of G_1 enables us to go from a fraction α non-erasures to a fraction α of erasures. The errors do not play a major role since we can make ε_{l+1} as small as we wish to make the construction work (only requirement is that ε_{l+1} is a positive constant).

The above dealt with the case when a left node of G_2 had some neighbor whose list size was at most l (as opposed to $l+1$), so we could recurse. Now we focus on the opposite case, in which we directly perform the decoding.

Case 2: $|I| \le \alpha n$.

Consider any $i \notin I$. Observe that for all j for which (i,j) is an edge of G_2, the sets $L_2(i,j)$ are equal. Let $i_1, i_2, \ldots, i_{d_2}$ be the neighbors of i in G_2. We view each of $L_2(i,i_1), L_2(i,i_2), \ldots, L_2(i,i_{d_2})$ as an *ordered* list of $l+1$ elements with order induced by some fixed ordering of $L_{i_1}, \ldots, L_{i_{d_2}}$ respectively. (The d_2-tuples in L_{i_1} for example contain a different symbol of Σ from node i, and accordingly an ordering of elements in L_{i_1} induces a natural ordering of the elements in $L_2(i,i_1)$.)

4.3 Defining the k-copy H and a subset $V(c)$ of its vertices

Define $V = \{1,2,\ldots,n\}$ to be the set of left side vertices of G_2. Consider a bipartite graph $HH = (A,B,E)$, where $A = B = V \times \{1 \ldots l+1\}$, with left degree d_2 constructed as follows. For any $i \notin I$, we construct an edge between (i,t) and (j,s) if and only if (i,j) is an edge of G_2 and $(L_2(i,j))_s = (K_i)_t$ (as per the ordering of elements in $L_2(i,j)$ and K_i defined above). Note that for each i,t, the value of s is unique. For $i \in I$, we construct an edge between (i,t) and (j,t) for $t = 1,2,\ldots,l+1$ and for every j which is a neighbor of i in G_R.

In the following, we will consider the graph H defined as $HH^2_{|A}$. This graph has the crucial property of being a k-copy (defined below) of G_R^2, the square of the Ramanujan graph G_R, for $k = l+1$.

Definition 1. *A graph $H = (V \times \{1\ldots k\}, E)$ is a k-copy of a graph $G = (V, E')$, if for every edge $\{i,j\} \in E'$ there is a permutation $\pi : \{1\ldots k\} \to \{1 \ldots k\}$ such that all edges $\{(i,t),(j,\pi(t))\}$ are present in E, for $t = 1\ldots k$, and these are the only edges in H. (Note that the definition allows the permutations π for different edges to be different.)*

The graph H has degree $d = d_2^2$ (with each node having d_2 self loops). We find it convenient to retain any multiple edges and self loops, since this way the adjacency matrix of G_R^2 will have a spectrum which is exactly the square of the spectrum of G_R (which we know well since G_R is a Ramanujan graph), and in particular the second largest eigenvalue of G_R^2 is at most $(2\sqrt{d_2})^2 = 4\sqrt{d}$. Since H was defined to be an induced subgraph of HH^2 (while retaining self loops and multiple edges), the following fact, which we will make use of later, holds:

Fact 2. *The adjacency matrix of the graph H is symmetric and positive semidefinite.*

Let us consider a single codeword which is consistent with at least $(1-\varepsilon_{l+1})$ of the lists $L_1, L_2, \ldots, L_n$. Clearly if we are able to find each such codeword then we will have accomplished the list recovering task (i.e. finding all consistent codewords) as well. Let c be the corresponding codeword prior to the final redistribution using the Ramanujan graph G_2 (i.e. c is a codeword of $G_1(C_l)$. Define

$$V(c) = \{(i,t) : i \notin I,\ i \text{ has no neighbors among } j\text{'s for which } L_j \text{ is corrupted, and } c_i = (K_i)_t\}\ .$$

Note that $|V(c)| \ge (1-\beta)n$, for $\beta = \alpha + d_2\varepsilon_{l+1}$. For the ease of exposition, we assume that all elements of $V(c)$ are of the form $(i,1)$. Observe that, for any $(i,1),(i',1) \in V(c)$, there is an edge (in H) between these two nodes if and only if there is an edge $\{i,i'\}$ in G_R^2.

Our goal now is to find a set which is "close" to $V(c)$, since then we can recover a slightly corrupted version of c, namely a string, say r, which differs from c in, say, at most a fraction $1/5$ of places. We can then complete the decoding using the algorithm for C_l by the following two steps:

1. Compute a received word of the code C_l by picking a symbol for each position using the majority vote from the neighboring symbols of r (if there is no clear majority, we pick an arbitrary symbol).
2. By the property of r and the graph G_1, the above received word is within fractional distance $1/5$ of some codeword of C_l (in fact it will much closer to a codeword of C_l since the fraction of left nodes of G_1 which all have at least $1/2$ their neighbors within a fraction $1/5$ of the right side will be small if G_1 is a good expander). Thus, we can decode the above received word using the algorithm for C_l.

4.4 Finding $V(c)$

The previous argument shows that it suffices to find a subset of vertices of H which is close to $V(c)$. In this section, we discuss how this can be done. The set $V(c)$ turns out to be a very dense component of H and moreover induces a subgraph which is essentially a copy of G_R^2 and thus is almost a Ramanujan graph. These facts enable us to find $V(c)$ by finding a suitable dense component of H. Let us record the key properties of $V(c)$.

Lemma 2. *The number of edges in H leaving $V(c)$ is at most βdn. Or equivalently, the average degree of the subgraph $H_{|V(c)}$ is at least $(1-\beta)d$.*

Proof: Define $I(c)$ to be the projection of $V(c)$ on the first coordinates, i.e. $I(c) = \{i : (i,1) \in V(c)\}$ (recall the assumption for definiteness that all elements of $V(c)$ are of the form $(i,1)$). We have $|I(c)| = |V(c)| \ge (1-\beta)n$. Therefore at most βdn edges leave $I(c)$ in G_R^2. Since nodes $(i,1)$ and $(i',1)$ in $V(c)$ are adjacent in H precisely when (i,i') is an edge in G_R^2, the lemma follows. □

The above suggests that finding a dense component of H should do the job. Unfortunately, doing it in near linear time is rather tricky. The crux of the idea is to use spectral partitioning: at a coarse level spectral partitioning enables one to decompose a graph into two parts or clusters based on the sign of a certain eigenvector of the adjacency (or related) matrix of a graph. The hope is that the properties of the eigenvector force it to have the same sign on most of the nodes in each cluster.

To be more specific, we achieve our goal by iterating the following procedure. Each iteration prunes $\Omega(n)$ nodes of the graph H, but makes sure that only a small fraction of nodes from $V(c)$ are removed. Thus, after $O(k)$ iterations, the leftover set is close to $V(c)$.

More formally, each iteration has the following functionality. As an input, we get a set $X \subseteq V \times \{1 \ldots k\}$ such that: (i) $|V(c) - X| \leq \gamma n$, and (ii) the average degree in $H_{|X}$ is $\geq d(1-\gamma)$.

The goal is compute $Y \subset X$ such that $|X - Y| \geq \Omega(n)$, and Y satisfies the above properties, with γ replaced by a slightly larger quantity γ''. Initially $X = V \times \{1 \ldots k\}$. We can stop when there are around n nodes left, and since the number of iterations to reach this stage is $O(k)$, we will be left with a set close to $V(c)$ provided we start with small enough γ.

Before giving the details of how each iteration proceeds, we try and give some intuition about its working. Since H is a d-regular graph, its principal eigenvector is the all 1's vector with eigenvalue d. It is well known that the eigenvector with the second largest eigenvalue is one which maximizes $y^T H y$ subject to $y \cdot 1 = 0$. Define $H' = H_{|X}$ be the current subgraph in consideration. The following lemma (proof omitted) guides our intuition (recall that we defined $\beta = \alpha + d_2\varepsilon_{l+1}$ above):

LEMMA 3. *Let $X \subseteq V \times \{1 \ldots k\}$ be a set of size at least rn such that $H_{|X}$ has average degree at least $d(1-\gamma)$ and $|V(c) - X| \leq \gamma n$ for some $\gamma \geq 2\beta$. Let H' denote the adjacency matrix of $H_{|X}$. Then there exists a vector $\hat{x} \in \mathbb{R}^X$ that has the same sign on all coordinates in $V(c) \cap X$ and which satisfies $\hat{x}.1 = 0$ and $\hat{x}^T H' \hat{x} \geq d(1 - 2\gamma - 3\gamma/(r-1))$.*

The above suggests that we might find a vector x such that $x.1 = 0$ that maximizes $x^T H' x$ and then retain just nodes u of H' for which x_u is of one sign, say positive for concreteness, in the set Y, and hope that most nodes of $V(c)$ are retained in the process. However, there could be two candidate codewords c and c', and the vector x might have most of its mass on $V(c')$ and no mass on $V(c)$. But in this case we can throw out components of x which are large, and since $\|x_{|V(c)}\|_2$ is small, we will retain most of $V(c)$. Of course we do not know we are in this case since we do not know $V(c)$ or $V(c')$, but we just "guess" whether this is the case. Since the number of iterations is small, we can just try out all possible guesses. In the other case, x has substantial mass on $V(c)$ and by Lemma 3, by retaining the positive indices of x, we will make good progress towards finding $V(c)$ (strictly speaking we only know that x has the same sign on the indices in $V(c)$, but we can "guess" if this sign is positive or negative). The formal arguments will follow shortly. We now present the details of each iteration. All graph operations are defined with respect to $H' = H_{|X}$.

Algorithm *Spectral-Decode*:

Input: A set $X \subseteq V \times \{1, 2, \ldots, k\}$ with $|X| \geq 21n/20$, that satisfies $|V(c) - X| \leq \gamma n$ and the condition that average degree of $H' = H_{|X}$ is at least $(1-\gamma)n$.

Output: A set $Y \subseteq X$ with $|X - Y| \geq n/4$ which satisfies the above conditions with a slightly larger parameter γ'' instead of γ.

1. Compute a unit vector x orthogonal to 1, which maximizes $x^T H' x$; here 1 denotes a unit vector with all coordinates equal.
2. **(Spectral partitioning step)** Guess if $\|x_{|V(c)}\|_2^2 \leq \delta$ (the exact value of δ will be specified later)
   - if YES, then create Y by removing from X all i's such that $x_i^2 \geq a\delta/n$ (a is "large" and will be specified during the analysis)
   - if NO, then define $S^+ = \{i \in X : x_i > 0\}$, $S^- = \{i \in X : x_i < 0\}$
   - guess if $1 \cdot x_{|V(c)} \geq 0$; without loss of generality assume YES
   - define $Y = X \cap S^+$
3. **(Density increasing step)** Keep removing from Y nodes of low degree (details later), until the resulting graph $H_{|Y}$ has average degree $\geq d(1-\gamma'')$. (We will specify the value of γ'' as a function of γ later.)
4. Output the resulting set Y.

We repeat this process until $|Y| \leq 21n/20$, and pick the initial γ small enough so that we have $|V(c) - Y| \leq n/20$ upon termination. We can then decode a string within fractional distance $1/5$ of c from Y, and then use it complete the decoding as discussed earlier.

We remark the above iterative algorithm must be run for all possible choices of guesses. We fix attention on one $c \in G_1(C_l)$ and will prove that when all guesses are correct with respect to $V(c)$, we will find a set Y that is close to $V(c)$. Specifically, we need to prove that the algorithm removes at least $\Omega(n)$ nodes in each iteration (so the number of iterations is small and the overall running time is $O(n)$) and also, for the correct guesses, it does not remove too many nodes of $V(c)$ (so that we can claim that $|V(c) - Y|$ will be less than $n/20$ upon termination). We prove these facts in Appendix A (the final result is Theorem 3). We also prove in Appendix A.1 that *Spectral-Decode* can be implemented to run in $O(n \log n)$ time. Choosing parameters in Theorem 3 and analyzing the parameters of the recursive construction lets us conclude our main result concerning list recoverable codes (detailed argument appears in Appendix A.4).

THEOREM 1. *For every $l \geq 1$, there exists an $\varepsilon_l = 2^{-2^{O(l)}}$ and an explicitly constructible $(1-\varepsilon_l, l, l)$-recoverable code family of rate $r_l = 2^{-2^{O(l)}}$ over an alphabet of size $2^{2^{2^{O(l)}}}$ which can be encoded in linear time and can be $(1-\varepsilon_l, l, l)$-list recovered with high probability in $O(n \log n)$ time.*[3]

5. LINEAR TIME ENCODABLE AND LIST DECODABLE CODES

We can now use the idea describe in Section 3.1 to get list decodable codes from the list recoverable ones from the previous section, and thus prove our main result:

THEOREM 2. *For every $\delta > 0$, for every integer $q \geq 2$, there is an explicit family of q-ary codes which is linear time encodable and $(\frac{1}{q} + \delta, O(1/\delta^3))$-decodable in linear time by a randomized algorithm (the algorithm will output the correct list of codewords with overwhelming probability). The rate of the code family is $2^{-2^{O(\delta^{-3})}}$.*

In order to obtain a linear decoding time as opposed to the $O(n \log n)$ decoding time, we will use the list recoverable

[3]The decoding algorithm is randomized due to the fact that the computation of x which maximizes $x^T H' x$ in each iteration of *Spectral-Decode* is randomized.

codes guaranteed by Theorem 1, with the following slight change: the alphabet size of C will not be a constant, but instead be n^a for some constant $a \ll 1$. We note here that the list recovering algorithm in Section 4 uses $O(n \log n)$ word operations and does not use the fact that the alphabet is of constant size (thus the run time will be $O(n \log n)$ in the unit cost model even with alphabet size n^a for $a < 1$). After suitable concatenation, the overall blocklength will be $\Theta(n \log n)$ so that decoding complexity will be linear in the blocklength. We omit the formal details.

6. REFERENCES

[1] Michael Alekhnovich. Linear Diophantine equations over polynomials and soft decoding of Reed-Solomon codes. *Proceedings of FOCS 2002*, to appear.

[2] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ronny Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[3] Noga Alon and Fan R. K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Math.* 72 (1988), pp. 15-19.

[4] Noga Alon, Jeff Edmonds and Michael Luby. Linear time erasure codes with nearly optimal recovery. *Proceedings of FOCS'95*.

[5] Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld and Madhu Sudan. Reconstructing Algebraic Functions from Mixed Data. *Proceedings of FOCS'92*.

[6] Daniel Bleichenbacher and P. Nguyen, Noisy Polynomial Interpolation and Noisy Chinese Remaindering. *Proceedings of EUROCRYPT '2000*, LNCS vol. 1807, Springer-Verlag, pp. 53-69, 2000.

[7] Peter Elias. List decoding for noisy channels. *Wescon Convention Record*, Part 2, Institute of Radio Engineers (now IEEE), pp. 94-104, 1957.

[8] G. L. Feng. Two Fast Algorithms in the Sudan Decoding Procedure. Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing, pp. 545-554, 1999.

[9] Venkatesan Guruswami. *List Decoding of Error-Correcting Codes.* Ph.D thesis, Massachusetts Institute of Technology, August 2001.

[10] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, Las Vegas, NV, October 2001.

[11] Venkatesan Guruswami and Piotr Indyk. Near-optimal Linear-Time Codes for Unique Decoding and New List-Decodable Codes Over Smaller Alphabets. *Proceedings of STOC'02*.

[12] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.

[13] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 181-190, 2000.

[14] J. Kuczynski and H. Wozniakowski. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM J. Matrix Anal. Appl.,* 15 (1994), pp. 672-691.

[15] Alex Lubotzky, R. Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[16] Ronny Roth and Gitit Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, 46(1):246–257, January 2000.

[17] Daniel Spielman. *Computationally Efficient Error-Correcting Codes and Holographic Proofs.* Ph.D thesis, Massachusetts Institute of Technology, June 1995.

[18] Daniel Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1732, 1996.

[19] Daniel Spielman. The Complexity of Error-Correcting Codes. *11th International Symposium on Fundamentals of Computation Theory*, Lecture Notes in Computer Science # 1279, pp. 67-84; Krakow, Poland, September 1997.

[20] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[21] Amnon Ta-Shma and David Zuckerman. Extractor Codes. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 193-199, 2001.

[22] J. M. Wozencraft. List Decoding. *Quarterly Progress Report*, Research Laboratory of Electronics, MIT, Vol. 48 (1958), pp. 90-95.

APPENDIX

A. ANALYSIS OF THE SPECTRAL PARTITIONING ALGORITHM

A.1 Runtime analysis

We now prove that the algorithm runs in $O(n \log n)$ time. Since we will later prove that number of runs of *Spectral-Decode* is at most $O(k)$, it suffices to prove that *Spectral-Decode* runs in $O(n \log n)$ time. The crux of *Spectral-Decode* above is the computation of the vector x. This can be done in $O(n \log n)$ time using iterative algorithms for computing eigenvalues. In particular, we use the following theorem proved in [14].

FACT 3. *Given a symmetric, positive semidefinite matrix A of size $n \times n$, such that for any x we can compute Ax in time T. Moreover, let $\varepsilon \in (0, 1]$, and let $\lambda_{\max}$ be the largest eigenvalue of A. Then there is a randomized algorithm that finds a unit vector $x \in \mathbb{R}^n$ such that $\|Ax\|_2 \geq (1-\varepsilon)\lambda_{\max} = (1-\varepsilon)\max_{\|x\|_2=1} \|Ax\|_2$ in time $O(T \log n/\varepsilon)$.*

The particular algorithm used in the theorem is the *power method.* It proceeds by choosing an initial unit vector $y \in \mathbb{R}^n$ at random, and then computing $Ay, A(A(y))$ etc. Clearly, the running time of this algorithm is bounded by kT, where k is the number of iterations. The authors of [14] show that if $k > c \log(n)/\varepsilon$ for a certain constant c, then $x = A^k y$ satisfies the conditions of the theorem with high (e.g., constant) probability. We stress that the running time does *not* depend on the distribution of the eigenvalues (and on the eigenvalue gap in particular).

We need two lemmas.

LEMMA 4. *Let B be a symmetric, positive semidefinite matrix of size $n \times n$, with the largest eigenvalue λ, and let $z \in \mathbb{R}^n$ be a unit vector such that $\|Bz\|_2 > (1-\varepsilon/2)\lambda$. Then $z^T Bz \geq (1-2\sqrt{\varepsilon})\max_{\|y\|_2=1} y^T By$.*

LEMMA 5. *Let $a \in \mathbb{R}^n$ be a unit vector, and let A be a symmetric, positive semidefinite matrix. Then*

$$\max_{\|x\|_2, x\cdot a=0} x^T Ax = \max_{\|y\|_2=1} y^T By$$

where $B = (I - aa^T)^T A(I - aa^T)$ is symmetric and positive semidefinite. Moreover, for any $y \in \mathbb{R}^n$, if we compute $y' =$

$y - (y \cdot a)a$, *then* $y' \cdot a = 0$, $\|y'\|_2 \leq \|y\|_2$ *and* $y^T By = y'^T By' = y'^T Ay'$.

Proof: Follows from the well-known fact that $(I - aa^T)y = y - (a^T y)a$ is a projection of y onto the space orthogonal to a. ◻

Now we can perform the approximate computation of $\max_{\|x\|_2=1, x\cdot 1=0} x^T H' x$ as follows. Firstly, recall that by Fact 2, H is symmetric and positive semidefinite. Therefore, $H' = H_{|X}$ is also symmetric and positive semidefinite. From Lemma 5 it suffices to approximate $\max_{\|y\|_2=1} y^T By$, where $B = (I - 1\ 1^T)^T H'(I - 1\ 1^T)$ with 1 being the unit vector with all coordinates equal, or in other words $B = (I - J)^T H'(I - J)$ where J is the all 1's matrix. By Lemma 4, this can be done by approximately computing the largest eigenvalue of B. By Fact 3 the latter task can be done in $O(n \log n)$ time, since H' has $O(n)$ entries and multiplication by $(I - J)$ can be done in $O(n)$ time as well.

Note that there we are able to compute the x which maximizes $x^T H' x$ but instead only find an x which comes within $(1 - \varepsilon)$ of the best value. But ε can be made as small as we want (it enters in the constant factor in the big-Oh notation for the runtime, though), and all arguments we will make in the analysis will work with this small error. This is because the only thing we will use about the x computed by Step 1 is that it has large $x^T H' x$, and a factor $(1-\varepsilon)$ slack with respect to the best possible value does not matter. Therefore, in order to keep the presentation readable, we will ignore this aspect and pretend that Step 1 of *Spectral-Decode* indeed returns an x that maximizes $x^T H' x$.

A.2 Analysis of each iteration of Spectral partitioning

We now analyze Step 2 of each iteration of *Spectral-Decode* (i.e. the spectral partitioning based step). We will analyze Step 3, the "average degree increasing step", in the next subsection. The quantitative goals are to lower bound the number of nodes removed in each iteration (i.e. the size of $X - Y$), and the relation between γ and γ''. We assume $|X| \geq 21n/20$ and γ is much bigger than β (jumping ahead the exact dependence will be that $\gamma = \Omega(\beta^{1/4} k)$), and once the relation between γ and γ'' is known, we can pick the initial γ (and thus β) small enough so that upon termination we will have $|V(c) - Y| \leq n/20$ (together with $|Y| \leq 21n/20$).

Here are some more remarks about the various constants that might aid the reading. The quantity $\beta = \alpha + d_2 \varepsilon_{l+1}$ is be thought of as being really small. The degree d of H is large, but much smaller than $1/\beta$ (to be precise it will be $\approx \beta^{-1/2}$). The quantity γ is small, but much larger than β or even $1/\sqrt{d}$ (again, to be precise, it will be $\approx \beta^{1/4} k$). The quantity γ'', the fraction of "mistakes" at the end of an iteration starting with fraction γ of mistakes will be polynomially larger than γ (specifically $\gamma'' \approx \sqrt{k\gamma}$).

In the analysis of each iteration of the spectral partitioning step (Step 2), there are two cases to consider based on how large $\|x_{|V(c)}\|_2$ is.

Case A: I.e., $\|x_{|V(c)}\|_2^2 \leq \delta$. In this case, since we only remove nodes for which $x_i^2 \geq a\delta/n$, we can remove at most n/a nodes of $V(c)$ and thus

$$|V(c) - Y| \leq (\gamma + 1/a)n \tag{2}$$

We need now to show that $X - Y$ is fairly large. This is done as follows. Consider a vector y obtained from x by setting $y_i = x_i$ if $x_i^2 \geq a\delta/n$, and $y_i = 0$ otherwise. Note that the support (i.e. set of non-zero indices) of y is precisely the set of nodes removed, i.e. the set $X - Y$, so we need to prove that y has large support. To this end, we first observe (Lemma 6 below) that since $\|x - y\|_2$ is small, $y^T H' y$ is close to $x^T H' x$ and thus large. Second, we argue that this implies that y must be non-zero on several entries owing to the expansion properties of the k-copy H.

Lemma 6. *For the vector y as defined above, we have* $y^T H' y \geq d(1 - 62\gamma - 3\sqrt{ka\delta})$.

Proof: By decomposing $x^T H' x$ into components corresponding to Y and $X - Y$, we obtain

$$\begin{aligned} y^T H' y &\geq x^T H' x - (x-y)^T H'(x-y) - 2y^T H'(x-y) \\ &\geq x^T H' x - d\|x-y\|_2^2 - 2\|y\|_2 d\|x-y\|_2 \\ &\geq x^T H' x - dka\delta - 2d\sqrt{ka\delta} \\ &\geq x^T H' x - 3d\sqrt{ka\delta}\ . \end{aligned}$$

since $ka\delta \leq 1$. Finally, by Lemma 3 applied with $r = 21/20$ (since we stop the iterations as soon as the set size drops below $21n/20$), we have

$$x^T H' x \geq d(1 - 62\gamma) \tag{3}$$

(recall that x is a vector which maximizes $x^T H' x$ and thus $x^T H' x \geq \hat{x}^T H' \hat{x}$). The lemma follows. ◻

To use the above lemma to conclude that y must have large support, we next show the following:

Lemma 7. *Let $d \geq 64k^2$. Then, if $|T| \leq n/4$, any unit vector $v \in \mathbb{R}^{nk}$ which is non-zero only on positions in T satisfies $v^T Hv \leq d/2$.*

Note that if $y^T H' y$ is large then so is $\tilde{y}^T H \tilde{y}$ where $\tilde{y}$ is obtained by extending y to all vertices in $V \times \{1, 2, \ldots, k\}$ by setting coordinates outside X to 0. The above lemma then implies that $\tilde{y}$ must have large support, and therefore y has large support (of size at least $n/4$) as well, provided

$$62\gamma + 3\sqrt{ka\delta} \leq \frac{1}{2}\ . \tag{4}$$

Proof of Lemma 7: Let $v = \langle v_{i,t} \rangle$ for $i = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, k$ be the vector in question. Wlog assume that all coordinates of v are nonnegative. Since v is a unit vector and has at most $|T|$ non-zero coordinates, we have $\|v\|_1 \leq \sqrt{|T|}$. Define a vector $z \in \mathbb{R}^n$ as $z_i = \sum_{t=1}^k v_{i,t}$ for each $i = 1, 2, \ldots, n$. Since H is a k-copy of G_R^2, clearly $v^T Hv \leq z^T G_R^2 z$. Now $z \cdot 1$ (where 1 is an n-dimensional unit vector which is $1/\sqrt{n}$ in each coordinate) is at most $\|v\|_1/\sqrt{n} \leq \sqrt{|T|/n} \leq 1/2$ if $|T| \leq n/4$. In addition, $\|z\|_2^2 = \sum_i \left(\sum_t v_{i,t}\right)^2 \leq \sum_i (k \sum_t v_{i,t}^2) \leq k$ (the last step follows since $\|v\|_2^2 \leq 1$).

The second largest eigenvalue of G_R^2 is at most $4\sqrt{d}$. (Recall that G_R is a Ramanujan graph of second eigenvalue λ_2 at most $2\sqrt{d_2}$, and therefore the second eigenvalue of G_R^2 is at most $\lambda_2^2 \leq 4d_2 = 2\sqrt{d}$.) Therefore $z^T Gz \leq d(z \cdot 1)^2 + 4\sqrt{d}\|z\|_2^2 \leq d/4 + 4\sqrt{d}k \leq d/2$ provided $d \geq 64k^2$. ◻

Thus we are done with the case when $\|x_{|V(c)}\|_2^2 \leq \delta$ — we deleted only a fraction $1/a$ of nodes in $V(c)$, and removed at least $\Omega(n)$ elements from X. We now proceed with the second case.

Case B: $\|x_{|V(c)}\|_2^2 \geq \delta$, and $1 \cdot x_{V(c)} \geq 0$. We will assume that $\|x_{|V(c)}\|_2^2 = \delta$ for cleaner exposition (the reader can verify that our main claims will remain true if made the calculations with $\delta' = \|x_{|V(c)}\|_2^2$ and then used $\delta' \geq \delta$). Recall that $Y = X \cap S^+$, and we need to show that $|V(c) - S^+|$ is small and $|X - Y|$ is large.

Again, we first start from showing that $V(c) - S^+$ is small. The intuition is that $H_{|V(c)}$ is pretty much G_R^2, and since $\|x_{|V(c)}\|_2^2$ is non-negligible, all entries $x_{|V(c)|}$ should be more or less equal to maximize the value of $x^T Hx$ (and therefore positive). Note that in the following analysis we implicitly assume that x is padded with 0's at positions outside H', and so $x^T Hx$ is well defined and equals $x^T H' x$ where the latter x is the unpadded one.

Decompose x into z, y, such that z corresponds to nodes in $V(c)$, and y takes care of the remaining coordinates. Assume that the entries of x are sorted such that z precedes y in x, and also that entries of z are sorted in decreasing order. Assume that $z_{n-t} > 0$ and $z_{n-t+1} \leq 0$. Let $t = \mu n$. We need to show that μ is small.

Observe that

$$x^T Hx \leq y^T Hy + 2y^T H_b z + z^T H_1 z \tag{5}$$

where H_b is the graph between $V(c)$ and its complement, and $H_1 = H_{|V(c)}$. Recall that by Lemma 2 H_b has at most $d\beta n$ edges. We will upper bound the first and second terms on the right side of the above inequality, and then use the fact that $x^T Hx$ is large (Lemma 3) to conclude that $z^T H_1 z$ must be large. This will imply that z has many positive coordinates or μ is small.

We know that $\|y\|^2 \leq 1 - \delta$. Thus the first term is at most $(1 - \delta)d$. To take care of $yH_b x^1$, we will need the following lemma concerning the l_2 norm of adjacency matrices of sparse subgraphs of an expander.

Lemma 8. *Let E be a Δ-regular graph over n vertices with second eigenvalue λ. Let F be a subgraph of E containing $\delta\Delta n$ edges of E. Then the 2-norm of F is at most $\sqrt{\delta}\Delta b + \lambda(1 + 2/b) + 2\Delta/b$, for any $b > 2$.*

Now, to bound the term $y^T H_b z$, construct $y' \in \mathbb{R}^n$, defined as $y'_i = \sum_{t=1}^{k} x_{i,t}$ for $i = 1, 2, \ldots, n$. As in the proof of Lemma 7, we have $\|y'\|_2^2 \leq k$ since $\|x\|_2 = 1$. Let G_b be the graph over V obtained from H_b in a way similar to how one can get G_R^2 from the k-copy H; i.e., we ignore the second coordinates in the node names, and replace multiple edges by one edge. Therefore, G_b has also at most $d\beta n$ edges. Clearly $y^T H_b z \leq y'^T G_b y'$ (since two nodes (i, t_1) and (i', t_2) can be adjacent in H_b only if i and i' are adjacent in G_R^2). But now G_b is a subgraph of the expander graph G_R^2, which makes arguing about the quantity $y'^T G_b y'$ much easier, since we can bound $\|G_b\|_2$ from above. Specifically, by using Lemma 8 with the choice $\delta = \beta$ and $\Delta = d$, setting $b = 1/\beta^{1/4}$, and using the fact that the second eigenvalue of G_R^2 is at most $4\sqrt{d}$, we conclude that $\|G_b\|_2 \leq 3\beta^{1/4}d + 4\sqrt{d}(1 + 2\beta^{1/4}) \leq 3\beta^{1/4}d + 5\sqrt{d} \leq 4\beta^{1/4}d$ (assuming $d \geq 25\beta^{-1/2}$).

Therefore,

$$y'^T G_b y' \leq \|G_b\|_2 \|y'\|_2^2 \leq 4\beta^{1/4} dk \ . \tag{6}$$

Together with (5) and the fact that $y^T Hy \leq (1 - \delta)d$, this implies that $z^T H_1 z \geq d(\delta - 62\gamma - 8\beta^{1/4}k)$.

It remains to consider the term $z^T H_1 z$. Since H_1 is a subgraph of G_R^2 induced by $V(c)$, we will extend z to z' which will include all nodes of G_R^2. Since any z can be converted into z' by adding 0 entries, if we prove that z' cannot have many non-positive entries, we are done. Also $z^T H_1 z = z'^T G_R^2 z'$, so we have

$$z'^T G_R^2 z' \geq d(\delta - 62\gamma - 8\beta^{1/4}k) \ . \tag{7}$$

We need to bound $z'^T G_R^2 z'$ under the assumption that z' has μn non-positive entries. To this end we use the bound on the second eigenvalue of G_R^2. Let $v_1, v_2, \ldots, v_n$ be the (unit) eigenvectors of G_R^2 sorted according the eigenvalues $\lambda_1, \lambda_2 \ldots \lambda_n$. Note that $v_1 = 1$, $\lambda_1 = d$, and $\lambda_i \leq 4\sqrt{d}$ for $i \geq 2$.

Represent z' as $\sum a_i v_i$; note that $a_1 = z' \cdot 1$ (here 1 denotes the vector which has $1/\sqrt{n}$ in each coordinate. Now,

$$\begin{aligned} z'^T G_2^R z' &= \sum a_i^2 \lambda_i \\ &= a_1^2 d + (1 - a_1^2) 4\sqrt{d} \\ &\leq d\big((z' \cdot 1)^2 + 4/\sqrt{d}\big) \leq d\big((z' \cdot 1)^2 + \beta^{1/4}\big) \end{aligned}$$

where in the last step we use $d \geq 25/\beta^{-1/2}$ (a fact we already used in bounding $y'^T G_b y'$). Together with (7), we get $(z' \cdot 1)^2 \geq \delta - 62\gamma - 8\beta^{1/4}k - \beta^{1/4} \geq \delta - 65\gamma$ provided

$$\gamma \geq 3\beta^{1/4}k \ . \tag{8}$$

(The above gives us a necessary lower bound on γ for our argument to work.)

Since z' has μn non-positive entries, we also have by a simple application of Cauchy-Schwartz inequality that

$$z' \cdot 1 \leq \frac{1}{\sqrt{n}} \cdot \sqrt{\|z'\|_2^2 (1 - \mu) n} = \sqrt{(1 - \mu)\delta} \ .$$

Combining with the above lower bound on $(z' \cdot 1)^2$, we get $\mu \leq \frac{65\gamma}{\delta}$, i.e. μ is small, as desired.

Therefore, in this case we have $|V(c) - S^+| \leq \mu n \leq 65\gamma n/\delta$, which implies

$$|V(c) - Y| \leq \Big(\gamma + \frac{65\gamma}{\delta}\Big) n \ . \tag{9}$$

We next need now to show that we remove a large number of nodes from X, or in other words S^- is large. We do it as follows. Firstly, we already proved that $|z' \cdot 1| \geq \sqrt{\delta - 65\gamma}$, or in other words $\sum_i z_i \geq \sqrt{(\delta - 65\gamma)n}$; note that $\sum_i z_i$ is positive since we assumed that $x_{|V(c)} \cdot 1 \geq 0$ and z is precisely the vector $x_{|V(c)}$. It thus also follows that $\|x_{|S^+}\|_1 \geq \sqrt{(\delta - 65\gamma)n}$. Since $x \cdot 1 = 0$, this means $\|x_{|S^-}\|_1 \geq \sqrt{(\delta - 65\gamma)n}$. If $|S^-| \geq n/4$ there is nothing to prove, otherwise by Cauchy-Schwartz, we have $\|x_{|S^-}\|_2^2 \geq \|x_{|S^-}\|_1^2/|S^-| \geq 4(\delta - 65\gamma)$. Furthermore, by Lemma 7, $x_{|S^-}^T H'_{|S^-} x_{|S^-} \leq \frac{d}{2}\|x_{|S^-}\|_2^2$. Therefore, we would have

$$\begin{aligned} x^T H' x &\leq x_{|S^-}^T H'_{|S^-} x_{|S^-} + x_{|S^+}^T H'_{|S^+} x_{|S^+} \\ &\leq \frac{d}{2}\|x_{|S^-}\|_2^2 + d(1 - \|x_{|S^-}\|_2^2) \leq d(1 - 2(\delta - 65\gamma)) \ . \end{aligned}$$

Since we know that $x^T H' x \geq d(1 - 62\gamma)$ by (3), we have a contradiction provided $\delta > 96\gamma$ (this condition will be satisfied since we will pick $\delta = \Omega(\sqrt{\gamma})$). Thus if $\delta > 96\gamma$, then $|S^-| \geq n/4$ in this case and therefore we remover at least $n/4$ nodes in each iteration.

Wrapping up the partitioning step. We have concluded that in Case A, we have $|V(c) - Y| \leq (\gamma + 1/a)n$ as per (2), and in Case B we have $|V(c) - Y| \leq (\gamma + 65\gamma/\delta)n$ as per (9) provided $\gamma \geq 3\beta^{1/4}k$ (this is Condition (8)). Recall that we have not so far specified how to pick parameters a, δ: it makes sense to pick them so that $1/a = 65\gamma/\delta$. Furthermore, in order to guarantee that we remover at least $n/4$ nodes from X (i.e. to ensure $|X - Y| \geq n/4$), we needed to assume Condition (4) that $62\gamma + 3\sqrt{ka\delta} \leq 1/2$ and $\delta > 96\gamma$. Since γ is small, both these conditions will be met by the choice $\delta = \left(\frac{5\gamma}{k}\right)^{1/2}$ (and $a = \frac{\delta}{65\gamma} = 1/(13\sqrt{5k\gamma})$).

To summarize, in the spectral partitioning step, we remove at least $n/4$ nodes from X to get Y, and

$$|V(c) - Y| \leq (\gamma + 13\sqrt{5k\gamma})n \leq 30\sqrt{k\gamma}\; n\;.$$

Thus, we go from γn "mistakes" to $\gamma' = O(\sqrt{k\gamma})n$ mistakes.

A.3 Increasing the density

At the end of the spectral partitioning step, the set Y has one of the properties we seek, namely that $|V(c) - Y|$ is small. However, we cannot directly start the next iteration with this Y since we also need the average degree of $H_{|Y}$ to be large (very close to d). The density increasing step in each iteration accomplishes this task. We now analyze this step. Note that the step keeps on removing low-degree nodes from Y till the resulting graph has large average degree. Therefore it is clear that when it terminates the resulting graph has large average degree and we only need to argue that we do not remove too many nodes of $V(c)$ during this process. Once again the fact that $H_{|V(c)}$ is basically just a copy of the expander G_R^2, together with the isoperimetry properties of a Ramanujan expander will let us establish this fact.

Now we move to the density-increasing procedure. Let $W = Y \cap V(c)$. We know that $|W| \geq (1 - \beta - \gamma')n$. We will alternatively think about the procedure as if it was removing nodes with degree smaller than $(1 - \gamma''/2)d$, until the number of such nodes is smaller than $\gamma'' n/2$. Then the average degree of the graph becomes $(1 - \gamma''/2)^2 d \geq (1 - \gamma'')d$.

We can split the whole process into stages. In the first stage, we remove all nodes which had degree at most $(1 - \gamma'')d$ at the beginning of the process. In the second stage, we remove the low-degree nodes created during the first stage and so on. Since after each stage we remove at least $\gamma'' n/2$ nodes, the whole process can have at most $t \leq 2k/\gamma''$ stages.

We need to investigate how W changes during the process. Firstly, we view W as a subset of the vertex set V_R of G_R^2. The reason why we can do this is the following. Consider just the subgraph induced by W; since $W \subseteq V(c)$ this is an induced subgraph of G_R^2. Imagine a copy $\tilde{G}$ of G_R^2 with vertex set $\tilde{V} \supseteq W$ — such a graph can be obtained by adding $n - |W|$ new nodes and "re-routing" any edges of H leaving W appropriately into the new nodes (this can be done consistently to get a copy of G_R^2 since edges within W are exactly as in G_R^2). Define $W_0' = \tilde{V} - W$, i.e. the newly added "imaginary" nodes. Note that $|W_0'| \leq (\beta + \gamma')n \leq 2\gamma' n$. Let W_i', $i = 1, 2, \ldots, t$ be the set of nodes in W removed in stage i. The nodes in W_i' have degree at most $(1 - \gamma''/2)d$ in the subgraph induced by $\tilde{V} - \cup_{j=0}^{i-1} W_j'$ (which is why they were removed in stage i), and thus have at least $\gamma'' d/2$ edges into $\cup_{j=0}^{i-1} W_j'$. We now use this fact together with the isoperimetric property of G_R^2 to bound cardinalities of each W_i' from above by $2\gamma' n$.

Assume we have shown that $|W_j'| \leq 2\gamma' n$ for $j = 0, 1, 2, \ldots, i-1$ for some i, $1 \leq i \leq t$. Let us apply the isoperimetric inequality from Fact 1 to the choice $Y = \cup_{j=0}^{i-1} W_j'$ and $X = W_i'$. Since the nodes in W_i' have at least $\gamma'' d/2$ edges into $\cup_{j=0}^{i-1} W_j'$, we have $|E(X, Y)| \geq \gamma'' |X| d/2$. Also, $|Y|/n \leq 2i\gamma' \leq 4k\gamma'/\gamma''$ and the second eigenvalue of G_R^2 is at most $4\sqrt{d}$. Plugging these bounds into Equation (1), we get

$$\frac{\gamma''}{2} \leq \frac{4k\gamma'}{\gamma''} + 8\sqrt{\frac{k\gamma'}{\gamma'' d}} \cdot \sqrt{\frac{n}{|X|}}$$

which gives $|X|/n = |W_i'|/n \leq \frac{64}{d\gamma''}$ provided $\gamma'' \geq \sqrt{16k\gamma'}$. In this case, $\frac{|W_i'|}{n} \leq \frac{16}{d\sqrt{k\gamma'}}$ and the latter quantity is at most $2\gamma'$ provided $d \geq 8/\sqrt{k\gamma'^3}$.

The total number of nodes removed from $V(c)$ in the density increasing steps is at most $\sum_{j=0}^{t} |W_j'| \leq (t+1)2\gamma' n = (2k\gamma'/\gamma'' + 2\gamma')n$. It follows that if Y' is the subset of Y at the end of execution of the density increasing Step 3 of the algorithm, then $|V(c) - Y'| \leq (3\gamma' + \frac{2k\gamma'}{\gamma''})n \leq \gamma'' n$ if we pick $\gamma'' = \sqrt{8k\gamma'}$.

Thus during Step 3 we go from $\gamma' n$ mistakes to $O(\sqrt{k\gamma'})n$ mistakes. Together with the analysis of the spectral partitioning step in the previous subsection, we conclude the following:

THEOREM 3. *For $\gamma \geq 3\beta^{1/4}k$, if $|V(c) - X| \leq \gamma n$, then for the set Y obtained at the end of running Algorithm* Spectral-Decode, *we have $|X - Y| \geq n/4$ and $|V(c) - Y| \leq O(k^{3/4}\gamma^{1/4})n$.*

A.4 Computation of parameters of C_{l+1}

We want the value of γ (the fraction of elements of $V(c)$ missed in the set Y) to be at most $1/20$ and $|Y| \leq 21n/20$ after iterating Algorithm *Spectral-Decode* a bunch of times. Since we remove at least $n/4$ nodes of H in each iteration, the number of iterations, say T, is at most $O(k) = O(l)$ (recall that $k = l + 1$). In order for γ be at most $1/20$ after T iterations, Theorem 3 implies that it suffices for the initial value of γ, say γ_0, to satisfy $O(k\gamma^{1/4^T}) \leq 1/20$, or equivalently $\gamma_0 = 2^{-2^{O(l)}}$. Since $\gamma_0 \geq 3\beta^{1/4}k$ this in turn limits β to be at most $2^{-2^{O(l)}}$. Recall that βn was an upper bound on the number of elements in $V_R - V(c)$ and that $\beta = \alpha + d_2\varepsilon_{l+1}$. Therefore we must have α and $\varepsilon_{l+1}d_2$ also to be $2^{-2^{O(l)}}$. The degree of G_1, d_1, can be $O(1/\alpha^2)$ which is $2^{2^{O(l)}}$, and d_2 must satisfy $d = d_2^2 = \max\{\Omega(\beta^{-1/2}), \Omega(k^2), \Omega(1/\sqrt{k\gamma_0^3})\}$ and $d_2 = 2^{2^{O(l)}}$ again suffices.

We also had the condition $\varepsilon_{l+1} \leq \frac{\varepsilon_l}{d_1 d_2}$, and for $d_1, d_2 = 2^{2^{O(l)}}$, one can have $\varepsilon_l = 2^{-2^{cl}}$ that satisfies this condition (for a suitably large constant c). Since the rate satisfies a similar recurrence $r_{l+1} = \frac{r_l}{d_1 d_2}$, one can also have $r_l = 2^{-2^{O(l)}}$. The alphabet size satisfies the relation $|\Sigma_{l+1}| = |\Sigma_l|^{d_1 d_2}$, and this can be satisfies with $|\Sigma_l| \leq 2^{2^{2^{O(l)}}}$. We therefore conclude the result of Theorem 1.