A Toolkit for List Recovery

Nick Wasylyshyn

A THESIS

in

Mathematics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Master of Arts

2016

Supervisor of Thesis

Brett Hemenway, Research Scientist

Graduate Group Chairperson

David Harbater, Professor of Mathematics

Thesis Committee:
Brett Hemenway, Professor of Computer and Information Science
Robin Pemantle, Professor of Mathematics

Acknowledgments

I would like to thank my family, friends, and department for their unwavering and unbelievable support. In multiple senses, I would not be here without them.

ABSTRACT

A Toolkit for List Recovery

Nick Wasylyshyn

Brett Hemenway, Advisor

Error–correcting codes, or codes for short, allow for communication across noisy channels by adding redundancy in a way that makes reconstruction of corrupted messages possible. We begin by introducing error–correcting codes as a primer to the main topic, list recovery. Whereas traditional codes assume that only a few characters in the message are corrupted and return one codeword, list–recoverable codes assume tame corruption of every character and return a list of possible codewords.

After defining and discussing list recovery, we move to the main contribution of our paper, a large collection of fundamental results about list–recoverable codes. As list–recoverable codes have not been studied on their own until recently, these results do not appear in an organized fashion in the existing literature.

Contents

Chapter 1

Background Material

1.1 Error–correcting Codes

The general problem of coding theory is to construct efficient encoding and decoding algorithms that allow messages to be transmitted accurately across a channel that may introduce errors. A message consists of k characters (k is called the *dimension* of the code) from an *alphabet* Σ, which is the collection of all possible characters. In our intuitive examples, we will use as our alphabet the set of lowercase letters in the Latin alphabet, which we denote A; in practice, an alphabet is often a finite field $\mathbb{F}_q$. The *message space* M is a subset of Σ^k consisting of all possible messages.

A message is encoded into a *codeword*, which will have n characters from the alphabet (n is called the *block length*), via an *encoding*, which is an injective function from $M \to \Sigma^n$. As we will see, n is often much bigger than k. A *code* is the collection of all possible codewords.

As an example, suppose our message space M is the set of six–letter English words, our encoding is the identity map $M \to A^6$, and our code is also the set of six-letter English words. If no errors may be introduced, this code works perfectly.

Of course, perfect communication is not a safe assumption to make. Moreover, there is no interesting coding theory to be done in this setting; the first code we think of, the identity map, is in some sense ideal. When even one error is introduced, however, it fails completely. For example, if the message is “mother”, the *received word* may be “bother”, which is another codeword. Even worse, a message such as “cereal”, which is not one letter away from any other codeword, may be misinterpreted; for instance, the received word may be “cerial”, which is one letter away from both “cereal” and “serial”.

We might correct for this in the same way that we correct for misunderstandings in speech: by repeating the message. This is known as a repetition code. Repeating a message twice does not correct for errors, as the received word “motherbother” may come from either the message “mother” or the message “bother”; repeating a word three times can correct one error (because two of the three repetitions will be untouched) but not two. In general, if we want to correct for e errors, we need to repeat the message $2e + 1$ times. This is, of course, very inefficient: The block length of our code is three times its dimension. We can quantify its efficiency by defining the *rate* of the code.

Definition 1.1.1. The rate of a code of dimension k and block length n is k/n.

We want the code to be as robust to noise as possible, and we also want the rate to be high for efficient transmission of messages. When the message space is all of Σ^k, as is usually the case, the rate can be no higher than 1 for any correct decoding to be possible. Such an encoding can never correct for errors, however; some redundancy is necessary. Before we get to codes that achieve a good balance between rate and error–correcting, let's come up with a bound for the maximum number of errors that a code can correct.

Definition 1.1.2. The Hamming distance $\Delta(c_1, c_2)$ between two codewords c_1, c_2 in a code C is the number of characters in which they differ. The minimum distance d of C is the minimum Hamming difference between two distinct codewords in C.

Remark 1.1.3. The minimum distance of a code is frequently expressed as a fraction $\delta = d/n$. This number, called the relative distance of a code, is often preferred, especially when studying what happens as $n \to \infty$.

From the above definition, we see that C can correct up to $(d-1)/2$ errors if d is odd and up to $d/2 - 1$ errors if d is even. For if a received message has at least $d/2$ errors, it may be closer to a different codeword than to the correct one. Therefore, we can say very informally that a good code is one that can correct close to $(d-1)/2$ errors while having a rate close to 1.

There is a very important class of codes, called linear codes, for which the minimum distance is easier to check. A linear code is a code in which any linear combination of codewords is also a codeword. In other words, with $\Sigma = \mathbb{F}_q$, a linear code of block length n is a linear subspace of $\mathbb{F}_q^n$. In this setting, the minimum distance reduces to the minimum positive integer d such that there exists a codeword with d nonzero entries. Reed–Solomon codes, which will be defined below, are an example of linear codes.

In addition to minimum distance and rate, we will also be concerned with the total number of codewords in a code; the binary code consisting of two codewords, one of all 0's and one of all 1's, has great minimum distance but will almost never be useful. Let us bound how many codewords a code can have and from that derive a bound on the minimum distance of a code in terms of its block length and dimension. Both bounds are known as the Singleton Bound, named after Richard C. Singleton ([5]).

Theorem 1.1.4 (Singleton Bound)**.** *If C is a code over an alphabet of size q with block length n and minimum distance d, then C has at most q^{n-d+1} codewords.*

Proof. Let C be any such code. Since each codeword in C differs from every other in at least d positions, it must be the case that every codeword of C truncated after the first $n-(d-1)$ characters must differ from every other in at least one position. There are q^{n-d+1} distinct words of length $n-(d-1)$, so C can have at most q^{n-d+1} codewords. $\square$

This method of simply truncating codewords seems crude, but it turns out that the Singleton Bound is strict. A linear code that meets this bound is called a maximum distance separable code.

Let us see how big a code's minimum distance can be in the usual case where the message space is all of Σ^k.[1]

Corollary 1.1.5 (Singleton Bound)**.** *If C is a code with dimension k, block length n, and minimum distance d, then $d \leq n - k + 1$.*

Proof. Let C be such a code over an alphabet of size q. Because encoding is an injective function, C has q^k codewords. Thus, by the Singleton Bound, $q^k \leq q^{n-d+1}$, so we must have $k \leq n - d + 1$. The result comes from rearranging. $\square$

Reed–Solomon codes, introduced in 1960 by the mathematicians after which they are named ([3]), are the most well known example of maximum distance separable codes; because of this, their versatility, and their efficient decoding, they are still widely used today.

1.1.1 Reed–Solomon codes

Let $\mathbb{F}_q$ denote the finite field of q elements. With $k \leq n \leq q$, we define a Reed–Solomon code of block length n and dimension k as follows.

Definition 1.1.6. To encode the message $m = (m_0, \ldots, m_{k-1})$, we map $m \mapsto f_m$, a polynomial defined by $f_m(x) = \sum_{i=0}^{k-1} m_i x^i$. We then pick $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ distinct elements, and encode m as $(f_m(\alpha_1), \ldots, f_m(\alpha_n))$; this is our codeword.

[1]In practice, this is always be the case. From this point on, it will be assumed that the message space is all of Σ^k unless otherwise specified.

Note that the degree of f_m is $k-1$, which implies if f_m and $f_{m'}$ agree on the at least k points, they are the same polynomial. Thus, two codewords which agree on at most $n-(k-1)$ points come from different messages; plugging this in to the Singleton Bound, we see that Reed–Solomon codes achieve the bound and can correct up to $(n-k)/2$ or $(n-k-1)/2$ errors depending on the parity of $n-k+1$.

Example 1.1.7. Suppose that $k=6$, $n=10$, and $q=11$, and say that we want to transmit the message $m=(1,2,3,4,5,6)$. This corresponds to the polynomial $f_m(x)=1+2x+3x^2+4x^3+5x^4+6x^5$. If we choose $\alpha_1,\ldots,\alpha_{10}=1,\ldots,10$, then the codeword corresponding to m is $(f_m(1),\ldots,f_m(10))=(10,2,3,4,10,1,3,5,8,8)$. No polynomial besides f_m will take the same values on more than five of the α_i, so a decoding algorithm could restore say $(0,0,3,4,10,1,3,5,8,8)$ to $(10,2,3,4,10,1,3,5,8,8)$ and determine that the original message was $(1,2,3,4,5,6)$.

The PhD thesis [4] of Atri Rudra contains many applications of Reed–Solomon codes to list decoding and list recovery which are outside the scope of this paper. Still, it is useful to see that codes with many good properties exist and are mathematically elegant. Now, let us enter the topic of this paper by describing the setting of list recovery.

1.2 List–Recoverable Codes

Often, the transmitted message is corrupted to the point where there are multiple codewords within the radius of error; in this case, choosing the closest codeword, even when this is well–defined, may lead to an incorrect decoding. It may be better to instead return a list of possible codewords. This is the goal of list–recoverable codes. There are two main problems in this realm: list decoding and list recovery. Colloquially, the list–decoding problem is as follows: Given a corrupted codeword, return a list of possible codewords. This leads to the following definition:

Definition 1.2.1. Fix $\varepsilon \in [0,1]$, $L \geq 1$. A code C of block length n is (ε, L)–list decodable if for all received words w, $|\{c \in C | \Delta(c,w) \leq \varepsilon n\}| \leq L$.

That is, there are no more than L possible codewords within ε fraction of any received word. If L is 1, this is equivalent to saying that C can correct εn errors, so list decodability is a generalization of decodability.

Example 1.2.2. Any code of block length n over an alphabet of size q is trivially $(1/n, nq-(n-1))$–list decodable, as any of the n characters in the received word may be replaced by at most q characters, and we subtract off $n-1$ to account for over–counting the received word.

Example 1.2.3. Take as an example the message space of three–letter English words with the encoding being the identity map. This code is not $(1/3, 32)$–list decodable because there are 33 English words that differ from the word "pat" in at most one character.

While the list–decoding problem aims to recover codewords from one received word in which at most a given fraction of the characters are corrupted, the list–recovery problem assumes corruption at every character in the codeword, but in a structured way. That is, if every position in the codeword has a short list of possible characters, return all codewords that interpolate through a high proportion of these possible characters. We define this formally below, beginning with the definition of a list–set.

Definition 1.2.4. For a code of block length n over an alphabet Σ, a list–set of size ℓ is a sequence of sets $(S_1, \ldots, S_n)$ with $S_i \subset \Sigma$ and $|S_i| = \ell$ for all i.

This allows us to define a list–recoverable code:

Definition 1.2.5. Fix $\varepsilon \in [0, 1]$ and $\ell, L \geq 1$. A code C of block length n an alphabet Σ is $(1 - \varepsilon, \ell, L)$–list recoverable if for every list–set of size ℓ, there are at most L codewords $c_1, \ldots, c_L$ such that for all i, the jth character of c_i is in S_j for at least $1 - \varepsilon$ fraction of the j.

Remark 1.2.6. We fix the parameters of list–recoverable codes from left to right: First we choose what fraction of errors we're willing to accept, then how large of lists the code has to tolerate. From there, we try to minimize L, the number of codewords that come from our sets.

Remark 1.2.7. We use $1 - \varepsilon$ to be consistent with the definition for list–decodable codes; in both cases, ε is likely to be small and the larger ε is the more errors are allowed. Often, $\varepsilon = 0$.

Example 1.2.8. Consider again our first example, that of six–letter English words and the identity encoding. Calculating the list–recoverable properties of this code is difficult, even when we fix ε and ℓ, but we show that this code is not $(1, 2, 5)$–list recoverable. (It is probably not $(1, 2, 6)$–list recoverable either, but this may require a different choice of messages to demonstrate.)

S_1	S_2	S_3	S_4	S_5	S_6
t	h	o	r	g	s
s	r	i	u	t	h

Figure 1.1: The codewords "shirts", "shorts", "shouts", "though", "trough", and "trouts" can all be interpolated through the S_i, so this code is not $(1, 2, 5)$–list recoverable.

When $\varepsilon > 0$, a code that is list–recoverable is sometimes called list–recoverable from errors. There is a variant of list–recoverability, called list–recoverability from erasures, defined below.

Definition 1.2.9. Fix $\varepsilon \in [0, 1]$ and $\ell, L \geq 1$. A code C is $(1-\varepsilon, \ell, L)$–list recoverable from erasures if for every sequence of sets $(S_1 \ldots, S_n)$, with $S_i \subset \Sigma$ and $|S_i| \leq \ell$ for at least $(1-\varepsilon)n$ i and $S_i = \mathbb{F}_q$ for the rest, there are at most L codewords $c \in C$ with $c \in S_1 \times \cdots \times S_n$.

Example 1.2.10. With the same example as above, we see that our code is not $(5/6, 2, 8)$–list recoverable from erasures because, in addition to the six codewords demonstrated above, an erasure in the fifth character would allow for the words "shires", "shirks", and "shores" as possibilities. There are many more ways to make words by erasing one character (not necessarily the fifth) from our current lists, and this is only with the S_i defined as we originally chose. So we continue to see that the identity encoding on the set of six–letter English words is nowhere close to being a good code from the standpoint of list–recoverability.

The difference between list recovery from errors and list recovery from erasures may seem subtle; if there are errors or erasures in a received word, that is equivalent to any character from the alphabet being a possibility at that index. The difference comes only at the time of receiving the message: in the list recovery with errors model, we will have a bound on the number of errors that occur, but we will not know where they occur (as every S_i has the same size). In the erasures setting, the erasures may occur at any collection of indices, but we know which indices have been erased at the time of receiving the word (because those S_i will be equal to the full alphabet).

A code that is $(1-\varepsilon, \ell, L)$–list recoverable is therefore $(1-\varepsilon, \ell, L)$–list recoverable from erasures. We will show in the next chapter what we can say about a code that is $(1-\varepsilon, \ell, L)$–list recoverable from erasures in terms of its list recoverability from errors. Before then, we need a few more definitions to make writing and proving things about list recovery smoother.

Definition 1.2.11. If C is a code of block length n over an alphabet Σ and $S \subset C$, then the list–cover of S is a sequence of sets $(S_1, \ldots, S_n)$ such that S_i is the set of characters that appear in the ith position of some codeword in S. We write list–cover$(S) = (S_1, \ldots, S_n)$.

Example 1.2.12. In Example 1.2.7, $(S_1, \ldots, S_6) =$ list–cover({shirts, trough}).

Definition 1.2.13. If C is a code of block length n over an alphabet Σ and $S_i \subset \Sigma$ for $i \in \{1, \ldots, n\}$, then the subcode of $(S_1, \ldots, S_n)$, denoted subcode$(S_1, \ldots, S_n)$, is equal to the set of codewords in C whose ith character is in S_i for all i.

Note that in general, if $\{c_i\}$ is a set of codewords, then the subcode of the list–cover of $\{c_i\}$ contains $\{c_i\}$.

Example 1.2.14. Again using Example 1.2.7, the subcode of the list–cover of the set {shirts, trough} contains {shirts, shorts, shouts, though, trough, trouts}.

With these definitions, we can redefine list–recoverability more succinctly in the case where $\varepsilon = 0$:

Definition 1.2.15. A code is $(1, \ell, L)$–list recoverable if the subcode of every list–cover of size ℓ has size at most L.

Finally, we define uniquely defining indices and uniquely defining codes, concepts which will show up in the next chapter.

Definition 1.2.16. Fix a list–set $(S_1, \ldots, S_n)$. The index $i \in \{1, \ldots, n\}$ is a uniquely defining index if for each $\sigma \in S_i$ there is a unique codeword in the subcover of $(S_1, \ldots, S_n)$ whose ith character is σ.

In other words, an index is uniquely defining if knowing the character at that index tells us which codeword it comes from, subject to the constraint that the codeword must interpolate through the list–set.

Example 1.2.17. In a list–set of size one, every index is trivially a uniquely defining index.

Example 1.2.18. If our message space is the set of four–letter English words and our encoding is the identity encoding, consider the example below. 1 is a uniquely defining index, because knowing that the first character is "q" tells us that the codeword is "quiz", whereas knowing that the first character is "j" tells us that the codeword is "jazz". Likewise, the second and third indices are uniquely defining, but the fourth is not, for two reasons. First, knowing that the fourth character is "z" does not tell us what the codeword is. Second, there is no codeword in the subcode of this list–set whose fourth character is "x".

S_1	S_2	S_3	S_4
q	u	i	z
j	a	z	x

Definition 1.2.19. A code of block length n is called (ℓ, ε)–uniquely defining if every list–set of size ℓ has at least εn uniquely defining indices.

Visibly, every code is $(1, 1)$–uniquely defining. Almost any example of English words with the identity encoding fails to be even $(2, \varepsilon)$–uniquely defining for $\varepsilon > 0$, however; the set of English words of a given length is simply too dense for every list–set of any size (greater than 1) to have a uniquely defining index. As we shall see in the next chapter, however, list–recoverable codes are often non–trivially uniquely defining.

Chapter 2

Fundamental Results of List–Recoverable Codes

The majority of these results come from unpublished notes by Brett Hemenway; our goal with this section is to facilitate the writing of future papers by saving authors the trouble of proving again these results and by helping with intuition.

Throughout, we assume that ℓ is smaller than both the size of the alphabet and the number of codewords in our code. Except in fringe cases, this is not a limiting assumption.

2.1 Basic Results of List–Recoverable Codes

Proposition 2.1.1. *A code C of block length n is $(1, \ell, L)$–list recoverable if and only if the following holds: For $c_1, \dots, c_{L+1} \in C$, if $(S_1, \dots, S_n)$ is the list–cover of*

$\{c_1, \ldots, c_{L+1}\}$, *then* $\max_{i \in \{1,\ldots,n\}} |S_i| > \ell$.

Proof. The forward direction is clear: If there is some list–set of size $r \le \ell$ whose subcode contains $L+1$ codewords, then C is not $(1, r, L)$–list recoverable and therefore cannot be $(1, \ell, L)$–list recoverable.

For the reverse direction, suppose that C is not $(1, \ell, L)$–list recoverable. Then there is some list–set $(\widetilde{S}_1, \ldots, \widetilde{S}_n)$ of size ℓ whose subcode contains at least $L+1$ elements $c_1, \ldots, c_{L+1}$. Let $(S_1, \ldots, S_n) = \text{list–cover}(\{c_1, \ldots, c_{L+1}\})$. Note that $S_i \subseteq \widetilde{S}_i$ for all i. This of course implies that $|S_i| \le |\widetilde{S}_i|$ for all i, so in particular $\max_{i \in \{1,\ldots,n\}} |S_i| \le \ell$, which is what we wanted. □

This result is not deep, but it provides, along with Definition 1.2.16, an alternative characterization of list–recoverable codes when $\varepsilon = 0$. It also gives us a sense of what the following, slightly more complicated, proofs will look like. The next result is perhaps intuitively clear, but proving it requires some formalism.

Proposition 2.1.2. *If a code C is of block length n and is $(1-\varepsilon, \ell, L)$–list recoverable, then $\ell \le L$.*

Proof. We may assume without loss of generality that $\varepsilon = 0$, as introducing errors cannot decrease the number of possible codewords that agree with our lists.

Fix ℓ; choose any ℓ codewords $c_1, \ldots, c_\ell \in C$; and let $(\widetilde{S}_1, \ldots, \widetilde{S}_n)$ be their list–cover. For each i, let S_i be $\widetilde{S}_i$ with enough characters (possibly zero) added so that $|S_i| = \ell$. Then $(S_1, \ldots, S_n)$ is a list–set whose subcover is of size at least ℓ as it contains $c_1, \ldots, c_\ell$. Thus, C is not $(1, \ell, \ell-1)$–list recoverable. So if C is $(1, \ell, L)$–list recoverable, then $L \geq \ell$. $\square$

The next proposition shows in a simple way how choosing a smaller value of ℓ always allows us to choose a smaller value for L. Precisely,

Proposition 2.1.3. *If C is $(1-\varepsilon, \ell, L)$–list recoverable and $\ell > 1$, then C is also $(1-\varepsilon, \ell-1, L-1)$–list recoverable.*

Proof. Suppose not. Then there exist $c_1, \ldots, c_L \in C$ and sets $\widetilde{S}_1, \ldots, \widetilde{S}_n$ of size $\ell - 1$ such that for at least $(1-\varepsilon)n$ of i, the ith character of c_j is in $\widetilde{S}_i$.

Let c_{L+1} be a codeword distinct from $c_1, \ldots, c_L$, and let S_i be $\widetilde{S}_i$ plus the ith character of c_{L+1} if that character is not already in $\widetilde{S}_i$ or plus some other character if it is. This forms a list–set of size ℓ whose subcode contains at least $L+1$ codewords. Thus, C is not $(1-\varepsilon, \ell, L)$–list recoverable either. $\square$

So our intuition doesn't fail us. The converse of this statement, however, is nowhere close to being true. We saw this implicitly in the previous chapter, looking at English words of six characters. This code (as is any code) is trivially $(1, 1, 1)$–list recoverable, as no two distinct codewords agree on every character. However, we demonstrated in the example that our code is not even $(1, 2, 5)$–list recoverable.

Other codes fail much more dramatically. For instance, if our code C were the set of binary sequences of length n with the identity encoding, then C would be $(1, 2, 2^n)$–list recoverable (but not $(1, 2, 2^n-1)$–list recoverable) because at every one of the 2^n codewords interpolates through the only list–set of size 2. We generalize this as follows.

Proposition 2.1.4. *If C is a code of block length n over an alphabet of size q, then C is $(1-\varepsilon, \ell, L)$–list recoverable for some $L \leq \sum_{i=0}^{\varepsilon n} \binom{n}{i}(q-\ell)^i \ell^{n-i}$.*

Proof. In the worst case, every sequence of n characters forms a codeword, so we assume that $\Sigma = \mathbb{F}_q$ and $C = \Sigma^n$. Fix a list–set $(S_1, \ldots, S_n)$ of size ℓ. Since $C = \Sigma^n$, it doesn't matter which list–set we choose.

We allow for a maximum of ε fraction errors. That is, we include every codeword for which at least $1-\varepsilon$ fraction of characters lie in the correct list in our list–set. The index i in the summand in the statement of the proposition counts the number of errors.

If there are exactly i errors, there are $\binom{n}{i}$ ways to choose in which indices the errors lie. Each of the i errors corresponds to a character at that index lying among the $q-\ell$ characters not in that position in the list–set, and for each of the $n-i$ positions where there isn't an error there are ℓ choices of characters.

As every sequence of n characters is a codeword, these choices are independent. Summing up over all possible number of errors, we get the desired result. $\square$

A code coming anywhere close to this bound is not worthwhile from a list–recovery standpoint. As one would expect, and as we shall show for small cases in the next section, we can improve this bound tremendously by assuming that C has a minimum distance of greater than 1.

2.2 Results Relating List Recovery to Minimum Distance

Theorem 2.2.1. *Every code C of block length n and minimum distance d over an alphabet of size q is $(1-\varepsilon, 1, L)$–list recoverable for some*

$$L \leq \frac{d(q-1)^2 q}{(q-1-q\varepsilon)^2(nq-dq-n)}$$

provided that

$$1 - \frac{q\varepsilon}{q-1} > \sqrt{1 - \frac{d}{\left(1-\frac{1}{q}\right)n}}.$$

Note that when $\ell = 1$, as is the case here, list recovery reduces to list decoding; this is a result about list decoding. A more general statement of this theorem which allows for s erasures in addition to the e errors, along with the proof, appears as Theorem 1 in [1]. Its proof is too technical for our purposes. Our statement appears slightly different because we have translated the result into the language of list recovery and simplified the result to the case in which no erasures are allowed.

Proposition 2.2.2. *Every code C of block length n and minimum distance d is $(1, \ell, \ell^{n-d+1})$–list recoverable.*

Proof. This is simply the Singleton Bound in the setting of list recovery: Having at most ℓ choices at each character is equivalent to being a code over an alphabet of size ℓ. $\square$

We can do much better than this when the minimum distance of C is high. Specifically,

Proposition 2.2.3. *If C is a code of block length n and minimum distance greater than $\frac{\ell-1}{\ell}n$, then C is $(1, \ell, \ell)$–list recoverable.*

Proof. Let $c_1, \ldots, c_\ell$ be ℓ distinct codewords in C, and let v be any element in the subcode of their list–cover. By the pigeonhole principle, v must agree with at least one of the ℓ c_i in at least $\frac{1}{\ell}$ fraction of positions; since the minimum distance is greater than $\frac{\ell-1}{\ell}n$, any two codewords that agree on at least $\frac{1}{\ell}$ fraction of characters are the same codeword. Thus, $v \in \{c_1, \ldots, c_\ell\}$ and C is $(1, \ell, L)$–list recoverable for some $L \leq \ell$.

Since we showed in Proposition 2.1.2 that a code being $(1-\varepsilon, \ell, L)$–list recoverable implies that $\ell \leq L$, we must therefore have that $\ell = L$. That is, C is $(1, \ell, \ell)$–list recoverable. $\square$

In the case $d = \frac{\ell-1}{\ell}n$, the Proposition 2.2.2 only guarantees that C is $(1, \ell, L)$–list recoverable for some $L \leq \ell^{n-\frac{\ell-1}{\ell}n+1}$; $\ell^{n-\frac{\ell-1}{\ell}n+1}$ simplifies to $\ell \cdot \ell^{n/\ell}$. Proposition 2.2.3 therefore improves our L by a factor of $\ell^{n/\ell}$, which is substantial considering that ℓ is often small compared to n.

We now generalize this by providing a condition relating minimum distance and ε that assures that a code is $(1-\varepsilon, \ell, \ell)$–list recoverable:

Proposition 2.2.4. *If C is a code with relative distance δ, then C is $(1-\varepsilon, \ell, \ell)$–list* *recoverable provided that*

$$d > 1 - \frac{1}{\binom{\ell+1}{2}} + \frac{2\varepsilon}{\ell}.$$

Proof. Fix any $\ell + 1$ codewords $c_1, \ldots, c_{\ell+1}$ in C. We will use a counting argument to show that their list–cover includes at least one list of size $\ell + 1$. To do this it suffices to show that $\sum_{i=1}^{n} |S_i| > \ell n$. We construct the lists piecemeal.

To begin, assume that $S_i = \emptyset$ for all i. Accounting for c_1 adds at least $(1 - \varepsilon)n$ elements to the S_i because c_1 agrees with at least $1 - \varepsilon$ fraction of the lists. Accounting for c_2 adds at least $(1-\varepsilon-(1-\delta))n$ elements, since the overlap between c_1 and c_2 is at most $(1 - \delta)n$. By the same reasoning, accounting for c_i therefore adds at least $(1 - \varepsilon - (i - 1)(1 - \alpha))n$ elements to the lists.

After adding all of $c_1, \ldots, c_{\ell+1}$, we have

$$n\sum_{i=1}^{\ell+1}((1-\varepsilon)+(i-1)(1-\delta)) = n\left((\ell+1)(1-\varepsilon) - \frac{\ell(\ell+1)}{2}(1-\delta)\right)$$

elements spread throughout the S_i. We therefore must show that

$$(\ell+1)(1-\varepsilon) - \frac{\ell(\ell+1)}{2}(1-\delta) > \ell.$$

Solving for δ yields the desired result. □

2.3 Results About List Recovery From Erasures

Last chapter, we posited that the main difference between list recovery from errors and list recovery from erasures is that with list recovery from errors forces us to account for the possibility of an erasure at any index, even after receiving the word. In some sense, therefore, list recovery from errors is the union over all possible erasures of the codewords that can be recovered from the list recovery from erasures setting. We make this precise as follows. (In the statement of the proof, c^i denotes the ith character of c.)

Proposition 2.3.1. *With C $(1-\varepsilon, \ell, L_0)$–list recoverable from erasures, define $\mathcal{C}_A = \{c \in C | c^i \in S_i \text{ for all } i \notin A\}$ for each $A \subset \{1, \ldots, n\}$ with $|A| = \varepsilon n$. Also, define $L = |\bigcup_A \mathcal{C}_A|$. Then C is $(1-\varepsilon, \ell, L)$–list recoverable from errors, and the set of recovered codewords is $\bigcup_A \mathcal{C}_A$.*

Proof. Let $\mathcal{C}$ be the set of codewords of C that interpolate through at least $1-\varepsilon$ fraction of the S_i. If $c \in \mathcal{C}$, that means that there is some A such that $c^i \in S_i$ outside of A; thus, $c \in \mathcal{C}_A$. This is true for every $c \in \mathcal{C}$, so $\mathcal{C} \subseteq \bigcup_A \mathcal{C}_A$.

To see that $\bigcup_A \mathcal{C}_A \subseteq \mathcal{C}$, fix A and fix $c \in \mathcal{C}_A$. This means that c is consistent with all the S_i outside of A, which is at least $1-\varepsilon$ fraction of the lists, so $c \in \mathcal{C}$. □

We have seen that list recovery from erasures is "easier" than list recovery from errors. Although the above result does not tell us much *a priori* about how much easier it is—all we can easily say about the relationship between L_0 and L follows from the union bound, which is not very good—it gives us some idea of the structure of the relationship between list recovery and list recovery from errors. We now give the analogy of Proposition 2.1.4 to the mode of list recovery from erasures, which gives an upper bound on L given ℓ and the number of errors we allow.

Proposition 2.3.2. *If C is a code of block length n over an alphabet of size q, then C is $(1-\varepsilon, \ell, L)$–list recoverable from erasures for some $L \leq \ell^{n(1-\varepsilon)} q^{\varepsilon n}$.*

Proof. This proof begins the same way as the proof of Proposition 2.1.4, by assuming that $\Sigma = \mathbb{F}_q$ and $C = \Sigma^n$. C is all of Σ^n, so it doesn't matter where the ε fraction of erasures occur; choose them to be the last ε fraction of the positions. So we have $S_1, \ldots, S_n$ with $|S_i| = \ell$ for $i \leq n(1-\varepsilon)$ and $|S_i| = q$ for the rest.

Since every word is a codeword, we can choose the characters independently to find that there are a total of $\ell^{n(1-\varepsilon)} q^{\varepsilon n}$ codewords consistent with these lists. This holds for every set of lists, so C is $(1, \ell, \ell^{n(1-\varepsilon)} q^{\varepsilon n})$–list recoverable. □

2.4 Results Relating List Recovery to Rate

Proposition 2.4.1. *If C is a linear $(1, 2, 2)$–list recoverable code of block length n and dimension k over $\mathbb{F}_q$, then the rate of C is less than $\frac{1}{2} + \frac{1}{n}$.*

Proof. Let $R = k/n$ be the rate of C, and let C have minimum distance d. Note that since C has dimension k, C is a k–dimensional subspace of $\mathbb{F}_q^n$. We prove the contrapositive: Suppose $R \geq \frac{1}{2} + \frac{1}{n}$; this implies that $Rn \geq \frac{n}{2} + 1$, so that $Rn - 1 \geq \frac{n}{2}$.

By the Singleton Bound, $d \leq n - k + 1 = (1-R)n + 1$. Pick two witnesses c_1, c_2 to this minimum distance, and let A be the set of indices where they differ. Then $|A| \leq (1-R)n + 1$.

Now let V be the set of $z \in \mathbb{F}_q^n$ such that $z\big|_A = 0$. C is a linear code, so V is a subspace of $\mathbb{F}_q^n$ of dimension at least $n - ((1-R)n + 1) = Rn - 1$. Then $C \cap V$ is a subspace of $\mathbb{R}_q^n$ of dimension

$$\begin{aligned}
\dim(C \cap V) &= \dim(C) + \dim(V) - \dim(C + V) \\
&\geq Rn + (Rn - 1) - n \\
&\geq \left(\frac{n}{2} + 1\right) + \frac{n}{2} - n \\
&= 1
\end{aligned}$$

so there is some nonzero vector $v \in C \cap V$. Since C is a linear code, we therefore have $c_1 + v \in C$; meanwhile, $(c_1 + v)\big|_A = c_1\big|_A$. In other words, c_1 and $c_1 + v$ agree everywhere that c_1 and c_2 differ. So, the list–cover of $\{c_1, c_2, c_1 + v\}$ is of size at most 2. But this means that there are at least three codewords that interpolate through a list–set of size 2, so C cannot be $(1, 2, 2)$–list recoverable. $\square$

Remark 2.4.2. The dimension of C, $k = Rn$, is an integer, so whether or not n is even will determine whether or not it is possible for a $(1, 2, 2)$–list recoverable code to have a rate greater than one half. If n is even, then $Rn < \frac{n}{2} + 1$ implies that $Rn \leq \frac{n}{2}$ or $R \leq \frac{1}{2}$. Meanwhile, if n is odd and $Rn = \frac{n}{2} + \frac{1}{2}$, then the rate of C is greater than one half.

Proposition 2.4.3. *Let C is a maximum distance separable code of dimension k*

and block length $n \geq 2k-1$. Then C is $(1,2,2)$–list recoverable.

Proof. The Singleton Bound gives us that the minimum distance d of C is equal to $n-k+1$. Thus, two codewords matching in at least $n-(d-1)=k$ characters must be equal.

Fix $c_1, c_2 \in C$, and fix $c_3 \in \text{subcode}(\text{list–cover}(\{c_1, c_2\}))$. Then c_3 must agree with either c_1 or c_2 in at least $\lceil \frac{n}{2} \rceil = k$ positions, so $c_3 \in \{c_1, c_2\}$. Thus, C is $(1,2,2)$–list recoverable. □

Remark 2.4.4. Note that when the inequality in the proof statement is an equality, that is, when $n = 2k-1$, the rate of C is $\frac{k}{2k-1} > \frac{1}{2}$. Since maximum distance separable codes exist (Reed–Solomon codes are an example, as noted in the previous chapter), there do exist $(1,2,2)$–list recoverable codes whose rate is greater than one half.

Finally, we prove an upper bound on the rate that holds for $(1,\ell,\ell)$–list recoverable linear codes.

Proposition 2.4.5. *If C is a linear code of block length n that is $(1,\ell,\ell)$–list recoverable, then we have $R \leq \frac{1}{\ell+1} + \frac{\ell}{n(\ell+1)}$.*

Proof. Fix ℓ codewords $c_1, \ldots, c_\ell \in C$ with the following specifications: The first $Rn-1$ characters of c_1 are the same as the first $Rn-1$ characters of c_2, the next $Rn-1$ characters of c_2 match the next $Rn-1$ characters of c_3, and so forth until we run out of either characters (in which case the construction stops, potentially partway through a batch of characters) or codewords (in which case we let the ℓth batch of characters from c_ℓ match the ℓth batch of characters in c_1).

Such a collection of codewords is always possible to find by the following reasoning: Suppose that C is a code over $\mathbb{F}_q$. Choosing a list–set of size ℓ for a code of block length n is *a priori* choosing elements from an $\mathbb{F}_q$–vector space of dimension ℓn. For each i, requiring that c_i be a codeword in C requires $(1-R)n$ linear constraints, as C is a subspace of $\mathbb{F}_q^n$ of dimension Rn. Each congruence of $Rn-1$ characters introduces another $Rn-1$ constraints. Adding up over the ℓ codewords, this is a total of at most $\ell((1-R)n+(Rn-1)) = \ell(n-1) < \ell n$ linear constraints, so a solution exists.

Now, let A be the set of indices in the list–cover of the c_i whose elements have at most $\ell-1$ symbols. By our construction, $|A| \geq \min\{n, \ell(Rn-1)\}$: In the case in which we ran out of characters, $|A| = n$; in the other, $|A| \geq \ell(Rn-1)$. Let V be the subspace of $\mathbb{F}_q^n$ consisting of words that are zero on the complement of A. Then the dimension of V is at least the minimum of n and $\ell(Rn-1)$.

Suppose for the sake of contradiction that $R > \frac{1}{\ell+1} + \frac{\ell}{n(\ell+1)}$. This simplifies to $\ell(Rn-1)+Rn > n$, and note that (obviously) $n + Rn > n$. We aim to show that $V \cap C \neq \{0\}$. The dimension of C is Rn, so in either case the sum of the dimensions of V and C is greater than n; so they must have nontrivial intersection.

Now, fix $v \in V \cap C \setminus \{0\}$, and consider the set $\{c_1, \dots, c_\ell, c_1 + v\}$. These $\ell+1$ codewords have a list–cover of sets of size at most ℓ; this contradicts the fact that

C is $(1,\ell,\ell)$–list recoverable, so we must have had that $R \leq \frac{1}{\ell+1} + \frac{\ell}{n(\ell+1)}$. □

2.5 Results About Uniquely Defining Indices

Proposition 2.5.1. *If C is a $(1,\ell,\ell)$–list recoverable code, then any collection of $t \leq \ell+1$ codewords has a uniquely defining index.*

Proof. Suppose $\{c_1,\dots,c_t\}$ has no uniquely defining index, and let $(S_1,\dots,S_n)$ be their list–cover. This means that in every S_i there is a character that is the ith character of at least two codewords in $\{c_1,\dots,c_t\}$. Therefore, $\max_{i\in\{1,\dots,t\}} |S_i| < t$, so C cannot be $(1,t-1,t-1)$–list recoverable: There are at least t codewords that interpolate through a list–set of size at most $t-1$.

However, by applying Proposition 2.1.3 repeatedly, C is $(1,t-1,t-1)$–list recoverable; this is a contradiction. Therefore, $\{c_1,\dots,c_t\}$ must have a uniquely defining index. □

We can do even better when the code is $(1-\varepsilon,\ell,\ell)$–list recoverable. This is not surprising, as it is a stronger statement for any ℓ, L for a code to be $(1-\varepsilon,\ell,L)$–list recoverable than to be $(1,\ell,L)$–list recoverable, but the way that our lower bound for the number of uniquely defining indices scales with the errors allowed makes this result interesting nonetheless.

Proposition 2.5.2. *If C is a $(1-\varepsilon,\ell,\ell)$–list recoverable code, then any collection of $t \leq \ell+1$ codewords has more than ε fraction of uniquely defining indices.*

Proof. Suppose that $\{c_1, \ldots, c_t\}$ is a collection of codewords with at most ε fraction of uniquely defining indices, and let $(S_1, \ldots, S_n)$ be their list–cover. This means that in the remaining $1-\varepsilon$ (or more) fraction of the S_i, there is at least one element that is the ith character in at least two codewords in $\{c_1, \ldots, c_t\}$.

Again by Proposition 2.1.3, C must be $(1-\varepsilon, t-1, t-1)$–list recoverable. We can therefore delete the remaining ε fraction of lists (so that all that remain have size at most $t-1$) and use list recovery to find a list–set of size $t-1$ for which only $t-1$ codewords agree with at least $1-\varepsilon$ fraction of the lists. However, we already have that $c_1, \ldots, c_t$ all agree with at least $1-\varepsilon$ fraction of the lists. This is a contradiction, so C must have more than ε fraction of uniquely defining indices. $\square$

Chapter 3

Further Reading

This paper has considered only basic results of list recovery with no concern for how to construct list–recoverable codes and how efficient they are. Although list recovery is a very young topic in coding theory, several very useful codes already exist with good list recovery properties. See in particular [2] for codes with high rate based on expander graphs and [4] for many examples including derivatives of Reed–Solomon codes.

Bibliography

[1] V. Guruswami and M. Sudan. List Decoding Algorithms for Certain Concatenated Codes. *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, p. 181–190, 2000.

[2] B. Hemenway and M. Wooters. Linear–Time List Recovery of High–Rate Expander Codes. *Automata, Languages, and Programming*, vol. 9134, p. 701-712, 2015.

[3] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, p. 300–304, 1960.

[4] A. Rudra. List Decoding and Property Testing of Error Correcting Codes. Ph.D. thesis, University of Washington, 2007.

[5] R.C. Singleton. Maximum Distance q–nary codes. *IEEE Trans. Inform. Theory*, vol. 10, p. 116-118, 1964.