Error Correction through Language Processing

Anxiao (Andrew) Jiang
Computer Science and Eng. Dept.
Texas A&M University
College Station, TX 77843
ajiang@cse.tamu.edu

Yue Li
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125
yli@caltech.edu

Jehoshua Bruck
Electrical Engineering Dept.
California Institute of Technology
Pasadena, CA 91125
bruck@caltech.edu

***Abstract*—There are two fundamental approaches for error correction. One approach is to add external redundancy to data. The other approach is to use the redundancy inside data, even if it is only the residual redundancy after a data compression algorithm. The first approach, namely error-correcting codes (ECCs), has been studied actively over the past seventy years. In this work, we explore the second approach, and show that it can substantially enhance the error-correction performance.**

This work focuses on error correction of texts in English as a case study. It proposes a scheme that combines language-based decoding with ECC decoding. Both analysis and experimental results are presented. The scheme can be extended to content-based decoding for more types of data with rich structures.

I. Introduction

There are two fundamental information-theoretic approaches for error correction. The first one is to add external redundancy to data, namely, to use error-correcting codes (ECCs). A vast amount of effort has been devoted to this area in the past seventy years, and many significant results have been obtained. Today we have capacity-achieving LDPC codes and polar codes, along with other successful codes including Turbo codes, BCH codes, Reed-Solomon codes, etc.

In this work, we explore the second approach, which is to use the redundancy inside data for error correction. Compared to ECCs, study in this area has been much more limited. (There is related work in joint-source-channel coding [1], denoising [7] and language modeling [4]. Yet the problem addressed here still has its uniqueness due to its pure focus on error correction, the open language model and its integration with ECC decoding.) There has been lots of research effort on data compression, and excellent compression algorithms (e.g, Huffman, Arithmetic and LZ coding) have been developed. However, they do not fully remove redundancy for many feature-rich data, such as languages, for multiple reasons. On the theoretical side, languages lead to some of the deepest problems in artificial intelligence, for which our understanding is still limited. On the practical side, most storage and communication devices require economic hardware or software implementations, and a compression algorithm that dives fully into language-like data can be too complex to use. So even with compression, there can still be residual redundancy in data to use for error correction. And our conjecture is that for many types of data, such redundancy can be plenty.

There are good practical reasons to study the second approach. In data storage, the amount of data institutions and individuals store keeps growing fast, and our dependency on stored data has never been higher. It is necessary to keep data highly reliable (ideally, never to be lost), so all means are needed for this goal. The two approaches – ECC and using internal redundancy – are compatible because the latter requires no change to the stored data. At normal times, the storage system can correct errors as usual using ECCs. But if errors exceed the correction capability of the ECC, – which is a rare but important event, – the second approach can be combined with ECC for a more powerful error correction performance. This way, a better balance between data reliability and average efficiency can be achieved.

We illustrate a framework for error correction in Fig. 1. A systematic (n, k) ECC codeword $(x_1, \cdots, x_k, x_{k+1}, \cdots, x_n)$ is transmitted through a channel, where $(x_1, \cdots, x_k)$ are information bits. The channel outputs a noisy codeword $(y_1, \cdots, y_k, y_{k+1}, \cdots, y_n)$. A *content-based decoder* uses the redundancy in information bits to decode $(y_1, \cdots, y_k)$ as $(z_1, \cdots, z_k)$. (In this paper we focus on language, and the decoder becomes a *language-based decoder*.) The noisy codeword $(z_1, \cdots, z_k, y_{k+1}, \cdots, y_n)$, possibly along with its soft information output by the content-based decoder, continues to be decoded by a regular ECC decoder, which outputs an estimation of the original codeword. The dotted line shows that the output of the ECC decoder may be given to the content-based decoder as input again, thus forming an iterative decoding system (although this method is not explored here). The framework can be extended to non-binary or non-systematic codes, and joint decoding algorithms for content-based decoding and ECC may also be used.

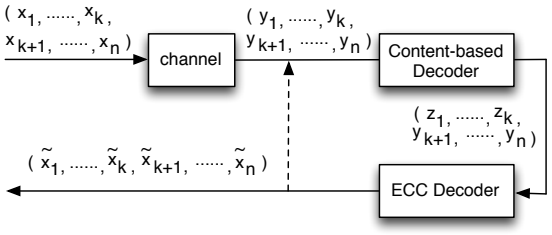

Fig. 1. A framework for error correction that combines content-based decoding with ECC decoding.

Let us use an example to illustrate the potential of language-based decoding.

Example 1 *Consider the well known paragraph by Shannon in Fig. 2 (a). We compress it using a Huffman code designed for*

Wikipedia texts in English, and show the binary codeword in Fig. 2 *(b). We then add i.i.d. errors to the codeword with BER (bit error rate)* 1%*, and show the noisy codeword in Fig.* 2 *(c).*

To decode the noisy codeword, we first use Huffman code to decode it, and show the resulting character string in Fig. 2 *(d). It is worthwhile to note that although the errors in bits have an error-propagation effect for characters, there are still many character substrings that we can recognize as (probably) correct. Let us consider those recognized substrings as decoded, and show the result in Fig.* 2 *(e); for clarity, the unrecognized segments are surrounded by angled brackets "<" and ">".*

To decode an unrecognized segment, let us use exhaustive search by first checking all error patterns of 1 error, then 2 errors, and so on. We stop as soon as a solution emerges that can be recognized as (probably) correct. If a segment has n *bits and* t *errors, at most* $\sum_{i=0}^{t} \binom{n}{i}$ *solutions will be checked (where we include the trivial error pattern of no error for completeness). That "number of all solutions" can be large; so to reduce the effort of human scanning, we narrow it down to solutions with two properties: (1) the bit string can be mapped successfully to a character string by Huffman code; (2) all words in the character string appear at least 500 times in Wikipedia (which is a quite small word-count threshold for Wikipedia). We call solutions with property (1)* decodable solutions*, and call solutions of both properties* "valid" solutions*.*

There are 15 unrecognized segments in total. We show their statistics and valid solutions in Fig. 2 *(f). It is noticeable that valid solutions are very sparse; and among them, it is easy for a human to recognize which one is the (probably) correct solution. For example, for the 9th unrecognized segment, among 438128 solutions, only 4 are valid solutions; and based on the text, it is easy to see that only "the actual message is one selected" is the correct answer. In fact, it is easy to recognize the correct answer for all 15 segments. (We comment that although some words might look strange in the valid solutions, they do appear in large text corpuses.)*

Note that 1% *is a large BER for storage systems. (Many storage systems use error-correcting codes designed for BER of* 0.4% *or less.) However, the noisy paragraph here has been corrected completely via human scanning without using any additional redundancy (as ECCs do).*

The objective of our work is to replace human scanning by machine computation, namely, to make language-based decoding fully automated. And instead of fully correcting data by language alone (namely, hard decoding), we are more interested in soft decoding by language and combining it with ECCs. As illustrated in Fig. 1, the ECC decoder can receive soft information from the content-based decoder as improved intrinsic information on its codeword bits.

This paper focuses on a technique we call *word recognition*, namely, to recognize words in a noisy codeword by flipping no or only a few bits. This leads to the partition of codeword bits into *stable* and *unstable* regions based on whether a region consists of recognized words/characters or not. The regions have polarized error probabilities, where stable regions exhibit significantly lower BERs. We analyze the performance of the error correction scheme both analytically and experimentally.

This work is a continuation of Shannon's well known 1951 paper on the entropy of the English language [5]. We have showed some preliminary results in [2], where a decoding algorithm by dynamic programming was used. The work here is new in several significant ways. First, its decoding is based on new algorithms and techniques, which achieves faster decoding speed and more robust decoding performance. Second, it uses soft decoding instead of hard decoding, with a clear performance gain. The technique of word recognition can be extended to more advanced techniques in language processing, which remain as our future work.

II. Language-based Decoding by Word Recognition

In this section, we present and analyze an efficient language-based decoding technique that we call *word recognition*.

A. Sparsity of Words and Word Recognition

We define a word as a maximal sequence of letters in a text. For simplicity, we do not differentiate lowercased and uppercased words, so "Information" and "information" are seen as the same word (of two forms). Given this definition, we can collect words and their counts (numbers of appearances) from a large set of texts (called a *corpus*). The count $C(w)$ of a word w approximates its likelihood of appearance. A text can contain words, punctuation marks, numbers and special characters (such as Latin symbols). We use this generic language model for our decoding technique that focuses on words. Note that if more rules are adopted in the language model, the accuracy of decoding can be further improved.

The redundancy of language comes from multiple factors, one of which is the sparsity of words. For $i \in \mathbf{N}^+$, let L_i denote the number of words of i letters. A first-step approximation for L_i is a Gaussian distribution $L_i \approx \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(i-\mu)^2}{2\sigma^2}}$, with the average word length $\mu \approx 8$ [6]. Since there are 26^i possible letter strings of length i, the density of words of length i, $D_i \triangleq \frac{L_i}{26^i} \approx \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(i-\mu)^2}{2\sigma^2}} 26^{-i}$, decreases exponentially fast for large word length.

In the following, we analyze the effectiveness of word recognition given the sparsity of words. Let $\mathcal{H}$ be a Huffman code that maps characters to binary codewords. Then for a word $w = (l_1 l_2 \cdots l_m)$ of m letters, its Huffman codeword is $\mathcal{H}(w) \triangleq (\mathcal{H}(l_1), \mathcal{H}(l_2), \cdots, \mathcal{H}(l_m))$. Let $\mathcal{C}$ be the set of characters (including letters and other characters). Let $c \in \mathcal{C}$ be a character, whose frequency of appearance is $f(c)$. Then for a Huffman code optimized for this frequency distribution, the length of the binary Huffman codeword for c is $|\mathcal{H}(c)| \approx -\log_2 f(c)$.

Assume that a word w appears with probability $p(w) = \frac{C(w)}{C_T}$, where $C(w)$ is the count of w and C_T is the total count of words (i.e., a normalization factor). Let $d_H(\cdot,\cdot)$ denote the Hamming distance between two vectors. Now consider a communication model. A sender chooses a word w with

(a) The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point. Frequently the messages have meaning; that is they refer to or are correlated according to some system with certain physical or conceptual entities. These semantic aspects of communication are irrelevant to the engineering problem. The significant aspect is that the actual message is one selected from a set of possible messages. The system must be designed to operate for each possible selection, not just the one which will actually be chosen since this is unknown at the time of design.

(b) 10110110000111110010110100000111001011110100000100111100101010100011000010111011100101000001000110011 1.....000

(c) 10110110000111110010110100000111000011110100000100111100101010100011000010111011100101000000000110011 1.....000

(d) The fuidamental proc nra of communication is that of reproducing at one point eithea exactly or approximatelorgaaessage selected at another point. Frecduently the messages have meaning; that is they epoer to y(nri,rrelated according to some system with certain pnsesica.sl c aih0ohal entities. These semantic aspects of communication are irrelevast to the engineering problem. The tignificant atpecs is that thloc a o iaaessa nrrffnledlected from a set of possible ne0nles. The system muszae designed to operquttidach possible selection, 0t just the one which wil)cohally be chosen since t, olfa rig iwn at the time of design.

(e) The <fuidamental proc nra >of communication is that of reproducing at one point <eithea> exactly or <approximatelorgaaessage> selected at another point. <Frecduently> the messages have meaning; that is they <epoer to y(nri,rrelated> according to some system with certain <pnsesica.sl c aih0ohal >entities. These semantic aspects of communication are <irrelevast> to the engineering problem. The <tignificant atpecs> is that <thloc a o iaaessa nrrffnledlected> from a set of possible < ne0nles>. The system <muszae> designed to <operquttidach> possible selection, <0t> just the one which <wil)cohally> be chosen since <t, olfa rig iwn> at the time of design.

(f)

Segment index	Number of bits	Number of errors	Number of all solutions	Number of decodable solutions	Number of "valid" solutions	"Valid" solutions
1	92	2	4279	3547	6	(1) fundamental proc ira (2) fundamental proc-m (3) fundamental problem (4) fundamental pro w ra (5) fundamental pr ch ra (6) fundamental t9ch ra
2	23	1	24	8	1	(1) either
3	105	2	5566	4564	2	(1) approximately a message (2) approximate tna message
.....	.....	.....	.....	.....	.....	.....
9	138	3	438128	373033	4	(1) th actual message is on selected (2) th actual message is one selected (3) the actual message is on selected (4) the actual message is one selected
.....	.....	.....	.....	.....	.....	.....
15	68	3	52463	27689	21	(1) t, ol8 ri il; (2) t, ol8 ri i.un (3) t, e is un,uc1 (4) t, e is un g own (5) t, is unknown (6) t, is un,uc1 (7) t, is un g own (8) t,p is unknown (9) t,p is un,uc1 (10) t,p is un g own (11) this is unknown (12) this is un,uc1 (13) t, old unknown (14) this is un g own (15) t, old un,uc1 (16) t, old un g own (17) t, ol ai9ti.un (18) t, ol ai"ti.un (19) t, o"nun,uc1 (20) t, o"nun g own (21) t, e is unknown

Fig. 2. Illustration of the potential of error correction through language processing (namely, language-based decoding).

probability $p(w)$, compresses it into a Huffman codeword $\mathcal{X} = (x_1, x_2, \cdots, x_n) \in \{0,1\}^n$, and transmits $\mathcal{X}$ through a binary symmetric channel (BSC) with error probability p_b. A receiver receives a noisy codeword $\mathcal{Y} = (y_1, y_2, \cdots, y_n)$. If there happens to be a word v such that $d_H(\mathcal{Y}, \mathcal{H}(v)) \leq \delta$ for some small integer δ (namely, the receiver recognizes a word v from the received binary codeword by flipping at most δ bits), what is the likelihood that $w = v$ (namely, the word recognition is correct)? Let us denote this likelihood by $P_{w=v}$, and call it the *confidence* of word recognition.

The exact computation of $P_{w=v}$ is complex because it depends on the distribution of letter compositions in words (namely, the letters in words are not i.i.d. in practice) and numerous other factors. For practical decoding, we are more interested in how different parameters, – including word length, word count and δ – impact the value of $P_{w=v}$, which is important for optimizing decoding algorithms. We estimate the impact of these parameters as follows.

1) Impact of Word Length: For simplicity, assume all letters have the same frequency of appearance, and their Huffman codewords have the same length. Then the receiver can determine the number of letters in word w, which we denote by m. (In practice, a reasonably small range for m may be determined for long words.) Since the character

set is fixed, $n = \Theta(m)$. Let $W_1 = v, W_2, \cdots, W_{L_m}$ denote the words of m letters. For $i = 2, 3, \cdots, L_m$, we have $\frac{Pr\{w=W_i|\mathcal{Y}\}}{Pr\{w=W_1|\mathcal{Y}\}} = \frac{Pr\{w=W_i\}Pr\{\mathcal{Y}|w=W_i\}}{Pr\{w=W_1\}Pr\{\mathcal{Y}|w=W_1\}} = \frac{C(W_i)p_b^{d_H(\mathcal{Y},\mathcal{H}(W_i))}(1-p_b)^{n-d_H(\mathcal{Y},\mathcal{H}(W_i))}}{C(W_1)p_b^{d_H(\mathcal{Y},\mathcal{H}(W_1))}(1-p_b)^{n-d_H(\mathcal{Y},\mathcal{H}(W_1))}}$. Let us denote $d_H(\mathcal{H}(W_1), \mathcal{H}(W_i))$ by d_i. Given that $p_b < 0.5$, $d_H(\mathcal{Y}, \mathcal{H}(W_1)) \leq \delta$ and the triangle inequality $d_H(\mathcal{Y}, \mathcal{H}(W_i)) \geq d_H(\mathcal{H}(W_1), \mathcal{H}(W_i)) - d_H(\mathcal{Y}, \mathcal{H}(W_1)) \geq d_i - \delta$, we get $\frac{Pr\{w=W_i|\mathcal{Y}\}}{Pr\{w=W_1|\mathcal{Y}\}} \leq \frac{C(W_i)p_b^{d_i-\delta}(1-p_b)^{n-d_i+\delta}}{C(W_1)p_b^{\delta}(1-p_b)^{n-\delta}}$. Notice that for large word length $m = \Theta(n)$, L_m (the number of words of length m) decreases exponentially fast. If we assume the Huffman codewords $\mathcal{H}(W_1), \cdots, \mathcal{H}(W_{L_m})$ are uniformly distributed in the binary vector space $\{0,1\}^n$, for large m, the minimum distance between words (including d_i) grows at the order of $\Theta(n)$. So the ratio $\frac{Pr\{w=W_i|\mathcal{Y}\}}{Pr\{w=W_1|\mathcal{Y}\}} \triangleq \alpha_i$ decreases exponentially in m. And the *confidence* of word recognition $P_{w=v} = Pr\{w = W_1|\mathcal{Y}\} = \frac{Pr\{w=W_1|\mathcal{Y}\}}{\sum_{i=1}^{L_m} Pr\{w=W_i|\mathcal{Y}\}} = \frac{1}{1+\sum_{i=2}^{L_m}\alpha_i}$ approaches one exponentially fast in the word length m.

2) Impact of Word Count: By the above analysis, we see that $\frac{Pr\{w=W_i|\mathcal{Y}\}}{Pr\{w=W_1|\mathcal{Y}\}}$ is proportional to $C(W_i)$ and inversely proportional to $C(W_1)$. So when the word count of the recognized word v increases, the confidence of word recognition increases steadily, but not exponentially fast.

3) Impact of Number of Flipped Bits: To recognize a word v given the noisy Huffman codeword $\mathcal{Y}$, we need to flip up to δ bits. By the above analysis, we see that the upper bound to $\frac{Pr\{w=W_i|\mathcal{Y}\}}{Pr\{w=W_1|\mathcal{Y}\}}$ increases exponentially in δ. So as the number of bits we need to flip in the received noisy codeword $\mathcal{Y}$ increases in order to recognize a word, the confidence of word recognition decreases exponentially fast.

The above observations through analysis is consistent with our experimental observations. Therefore, in our language-based decoding algorithm, we assign great weights to long recognized words, moderate weights to recognized words of large counts, and very small weights to words that require flipping multiple bits before they can be recognized.

B. Greedy Huffman Decoding and Its Synchronization

A fast speed of word recognition in a noisy bit sequence can be obtained for a reason we call *character synchronization*. As we have observed in Fig. 2, after a character string has been compressed by a Huffman code and then errors are added to its bits, when we decompress it, we still recognize many words. This means error propagation in recovered characters is not serious. More generally, given a noisy binary substring of the Huffman codeword for a character string, if it is not known where the error-free words begin in the substring, we can start decompressing at any position and still may recognize words. This enables us to use a greedy algorithm to recognize words, and quickly partition the binary string into "segments with recognized words" and the remaining segments, and consequently, quickly reduce the decoding problem with a large input to subproblems with small inputs.

Given a Huffman code for a character set $\mathcal{C} = \{c_1, c_2, \cdots, c_m\}$, assuming the characters in a character string are i.i.d. with probabilities $f(c)$ for $c \in \mathcal{C}$, the rate of synchronization can be analyzed. Let us illustrate it with a basic case. Assume there are $m = 3$ characters, with $\mathcal{H}(c_1) = (0)$, $\mathcal{H}(c_2) = (10)$ and $\mathcal{H}(c_3) = (11)$. Consider a binary string $B_1 = (b_0, b_1, b_2, b_3, \cdots)$ that is the Huffman codeword for a character string, and suppose an error changes its first bit, therefore changing the binary string to $B_2 = (b_0' = 1 - b_0, b_1, b_2, b_3, \cdots)$. We define the *synchronization delay* $D_{sync} \in \{1, 2, 3 \cdots\}$ to be bit index such that starting at $b_{D_{sync}}$, the Huffman code decodes B_1 and B_2 to the same characters. (For example, if $B_1 = (\underline{0}, \underline{1, 0}, \underline{1, 1}, \underline{0}, \underline{1, 1}, \cdots)$ and $B_2 = (\underline{1, 1}, \underline{0}, \underline{1, 1}, \underline{0}, \underline{1, 1}, \cdots)$, we have $D_{sync} = 3$, because $\mathcal{H}^{-1}(B_1) = (c_1, c_2, \mathbf{c_3}, \mathbf{c_1}, \mathbf{c_3}, \cdots)$ and $\mathcal{H}^{-1}(B_2) = (c_3, c_1, \mathbf{c_3}, \mathbf{c_1}, \mathbf{c_3}, \cdots)$, which become synchronized at bit b_3.)

Proposition 2 *When $b_0 = 0$, for $i = 0, 1, 2 \cdots$, $Pr\{D_{sync} = 2i+1\} = p_3^i p_1$, and $Pr\{D_{sync} = 2i+2\} = p_3^i p_2$; the expected synchronization delay is $\mathbf{E}(D_{sync}) = \frac{2-p_1}{1-p_3}$.*

When $b_0 = 1$, we have $Pr\{D_{sync} = 1\} = 0$, $Pr\{D_{sync} = 2\} = \frac{p_2}{p_2+p_3}$, and for $i = 1, 2, 3 \cdots$, $Pr\{D_{sync} = 2i+1\} = \frac{p_3^i p_1}{p_2+p_3}$ and $Pr\{D_{sync} = 2i+2\} = \frac{p_3^i p_2}{p_2+p_3}$; the expected synchronization delay is $\mathbf{E}(D_{sync}) = \frac{2}{1-p_3} + \frac{p_1 p_3}{(1-p_1)(1-p_3)}$.

We see that $Pr\{D_{sync} = i\}$ decreases exponentially in i. This means synchronization is fast and is helpful for decoding.

C. Combined Language-ECC Decoding Algorithm

We now present our error correction scheme. It has two steps: *language-based decoding*, and *ECC decoding*. (See Fig. 1.) The language-based decoding algorithm takes the noisy Huffman codeword for a text as input, and keeps finding more substrings that consist of recognized words as follows: (1) Decompress the codeword into characters, and greedily find substrings that consist of words of sufficiently great total length and count (let us call such a substring a "good n-gram"); (2) For a substring that is not a good n-gram, try decompressing it starting at a few different bit positions (to further reduce the effect of error propagation), to recognize more good n-grams; or flip one bit in it and then recognize more good n-grams in the same way. The algorithm is greedy and efficient. Each substring of the noisy Huffman codeword that corresponds to a recognized good n-gram is called a *stable region*. The remaining substrings are called *unstable regions*. In the end, the language-based decoding algorithm partitions the noisy Huffman codeword into *stable* and *unstable regions*. (Note that some bits may have been flipped.) Since stable/unstable regions reflect our observation on where errors concentrate, stable regions tend to have a much lower error probability than unstable regions.

The ECC decoder then decodes the noisy ECC codeword (which contains the updated noisy Huffman codeword as its information bits) with soft decoding as usual, with one modification: stable (resp., unstable) regions are assumed to have error probability p_{st} (resp., p_{un}), while the parity-check bits are still assumed to have the original error probability. (p_{st} and p_{un} are parameters set empirically beforehand.) It can

be shown that for LDPC codes, with this improved intrinsic information, a better decoding threshold can be achieved. We skip its details due to space limitation.

III. Experimental Performance

We now show by experiments that the language-based decoding algorithm can significantly improve error correction performance. The data source for experiments is Wikipedia, a very large and commonly used corpus for text analysis. We use around 2/3 of its English texts as training data and around 1/3 for tests. The training texts contain over 1 million distinct words, which are recorded along with their counts (numbers of appearances). The size of such a dictionary is negligible for most storage systems. We also design a Huffman code for the 117 characters in the used articles, which include letters, numbers, punctuation marks and other special characters, based on the characters' frequencies.

The ECC we use is an $(4376, 4095)$ LDPC code designed by MacKay [3]. It has a rate of 0.936 and is designed for BSC of error probability 0.2%, a typical parameter setting in storage systems. In experiments, randomly chosen texts are compressed by the Huffman code, partitioned into 4095-bit segments, and encoded by the LDPC code. Then random bit errors are added to codewords, and decoding is performed. We show the estimated bit error probabilities for *stable regions* (resp., *unstable regions*) as p_{st} (resp., p_{un}) in Fig. 3, which were set empirically before the experiments.

BER	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%
p_{st}	0.05%	0.065%	0.085%	0.095%	0.11%	0.13%
p_{un}	0.65%	0.95%	1.3%	1.6%	1.9%	2.1%
BER	0.8%	0.9%	1.0%	1.1%	1.2%	1.3%
P_{st}	0.14%	0.15%	0.16%	0.18%	0.19%	0.2%
P_{un}	2.3%	2.6%	2.8%	2.9%	3.1%	3.2%

Fig. 3. Estimated bit error probability for stable regions (p_{st}) and unstable regions (p_{un}) for different BER.

We let the error probability (BER) range from 0.2% to 1.3%, and randomly generate 1000 codewords for each BER. We measure performance by the percentage of codewords decoded successfully (called *success rate of decoding*). The results are shown in the table of Fig. 4, which also illustrates them in a figure. Here P_{ldpc} is the success rate of using the LDPC code alone, and P_{soft} is the success rate of combining language-based decoding with LDPC decoding. For comparison, we also show P_{hard}, defined as hard-decision language-based decoding combined with LDPC decoding, where the LDPC decoder uses the BER (instead of the soft information p_{st} and p_{un}) as intrinsic information for codeword bits.

It is easy to see that language-based decoding combined with LDPC decoding corrects errors significantly better than the LDPC code alone. While P_{ldpc} drops toward 0 quickly after BER exceeds 0.4%, the combined language-LDPC decoding can still correct a significant fraction of codewords. And the difference between P_{soft} and P_{hard} clearly shows the importance of soft information p_{st} and p_{un}.

BER	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%
P_{ldpc}	100%	98.2%	77.5%	27.4%	2.9%	0
P_{hard}	100%	99.8%	98.6%	93.9%	78.1%	51.5%
P_{soft}	100%	99.9%	99.5%	97.9%	94.2%	84.6%
BER	0.8%	0.9%	1.0%	1.1%	1.2%	1.3%
P_{ldpc}	0	0	0	0	0	0
P_{hard}	27.6%	9.8%	2.2%	0.3%	0.1%	0
P_{soft}	67.1%	47.8%	26.7%	12.4%	3.9%	1.4%

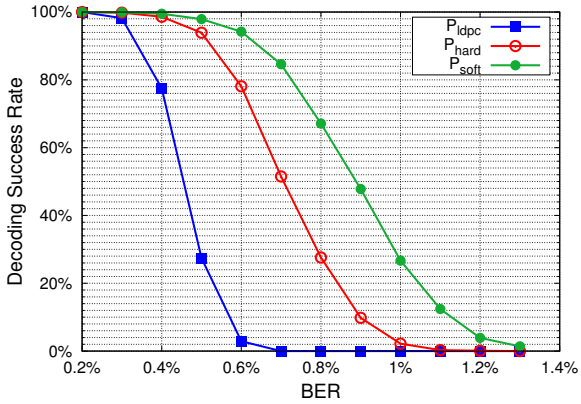

Fig. 4. The success rate of decoding with LDPC code alone (P_{ldpc}) and hard or soft language-based decoding combined with LDPC decoding (P_{hard} and P_{soft}, respectively), when the bit error probability (BER of a binary-symmetric channel) increases from 0.2% to 1.3%.

We can also roughly estimate the reduction in storage redundancy the language-based decoding achieves. For any BER $= p \geq 0.2\%$, if an ECC alone is used, to achieve the same success rate as P_{soft}, a fraction of P_{soft} codewords would need to use an ECC of rate at least $1 - H(p)$ instead of $1 - H(0.002)$. For $i = 1, 2, \cdots, 12$, let $p_i = 0.2\%, 0.3\%, \cdots, 1.3\%$ as in Fig. 4, and let $P_{soft,i}$ be the P_{soft} corresponding to $p = p_i$. So given $k \to \infty$ information bits, the "ECC alone" approach will assign at least $R = \sum_{i=1}^{11} \frac{(P_{soft,i} - P_{soft,i+1})k}{1-H(p_i)} + \frac{P_{soft,12}k}{1-H(p_{12})} - k$ parity-check bits, while the language-ECC approach will assign approximately $r = \frac{k}{1-H(0.002)} - k$ parity-check bits. The "ECC alone" approach needs R/r times the redundancy as the language-ECC approach does. We get $R/r = 3.52$, a very significant improvement in redundancy.

References

[1] E. Akyol, K. Rose and T. Ramstad, "Optimal mappings for joint source channel coding," in *Proc. ITW*, pp. 150–154, 2010.

[2] Y. Li, Y. Wang, A. Jiang and J. Bruck, "Content-assisted file decoding for nonvolatile memories," in *Proc. 46th Asilomar Conference on Signals, Systems and Computers*, pp. 937–941, 2012.

[3] D. MacKay, *Encyclopedia of Sparse Graph Codes*, *http* : *//www.inference.phy.cam.ac.uk/mackay/codes/data.html#l*132.

[4] A. Orlitsky and N. P. Santhanam, "Performance of universal codes over infinite alphabets," in *Proc. Data Compression Conference*, pp. 402–410, 2003.

[5] C. Shannon, "Prediction and entropy of printed English," in *Bell System Technical Journal*, vol. 30, no. 1, pp. 50–64, 1951.

[6] R. Smith, "Distinct word length frequencies: distributions and symbol entropies," in *Glottometrics*, vol. 23, pp. 7-22, 2012.

[7] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu and M. J. Weinberger, "Universal discrete denoising: Known channel," in *IEEE Trans. Information Theory*, vol. 51, no. 1, pp. 5-28, 2005.