Enhanced Error Correction via 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

I. Introduction

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.

There are good practical reasons to study the second approach. In data storage, including flash-memory systems, 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.

This work focuses on error correction of texts in English as a case study. (For an illustrative example, see the full paper of this work [2].) It proposes a scheme that combines language-based decoding with ECC decoding. Both analysis and experimental results are presented, and in particular, we have tested its experimental performance in a flash memory testbed under highly practical settings. The tested ECCs are LDPC codes, a family of capacity-approaching codes important for memories, and our results show that language-based decoding can very significantly enhance the error correction performance of LDPC codes by providing useful soft information.

This work is a continuation of Shannon's well known 1951 paper on the entropy of the English language. It is also related to our prior work [3], 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. Due to space limitation, we skip many details here. Interested readers may refer to the full paper [2]. And part of the results here are also presented in our work [1].

This work was supported in part by the NSF Grant CCF-1217944, a grant from Intellectual Ventures, and the BSF Grant 2010075.

II. Language-based Error Correction

Let us first introduce the ideas and practice of language-based decoding. Consider a text file compressed by a Huffman code into a binary string $B = (b_1, b_2, \cdots, b_n)$, where the Huffman code is optimized for English texts with characters (letters, punctation marks, etc.) as its alphabet. Add i.i.d. errors to the binary string with BER (bit-error rate) p, and we get a noisy binary string $B' = (b'_1, b'_2, \cdots, b'_n)$. When we greedily decompress B' back into a text file, as long as p is not too large (e.g., $p < 2\%$), it can be experimentally found that typically many segments of it are correct and easily recognizable, because the other segments are meaningless strings of random characters. (See [2] for a concrete example.) In other words, the greedily recovered text file typically consists of alternating "good" and "bad" segments, where good segments consist of valid words that can be easily collected from any large text corpus, while bad segments are just the opposite. This phase-transition phenomenon enables efficient language-based decoding: first, we can greedily decompress a file, and recognize the good segments (which we call *stable regions*) using a dictionary of words; second, for the bad segments (which we call *unstable regions*), we can search for nearby solutions by flipping at most a few bits, and recognize more good segments; the process can be repeated recursively until we reach a satisfactory solution or a time bound. Data structures can be used to further reduce time complexity.

The binary string B corresponds to the information bits of a systematic ECC. Our decoding consists of two steps: *language-based decoding* and *ECC decoding*. The language-based decoding algorithm partitions information bits into *stable* and *unstable* regions, where bits in stable (resp., unstable) regions can be assigned an updated BER $p_{st} < p$ (resp., $p_{un} > p$). (That is, the decoding helps polarize the distribution of BERs across bits.) This updated soft information is given to the ECC (which is an LDPC code in our work) as the update intrinsic information for information bits. The ECC performs its regular soft decoding algorithm and outputs the final result.

Two important properties make language-based decoding effective. First, *the recognized words (in stable regions) are correct solutions with high probability*. This is mainly due to the high sparsity of words. Second, *errors in a compressed*

file do not cause substantial error propagation in the decompressed file. This is mainly due to fast synchronization of characters with prefix-free codes. We present theoretical analysis of the two properties in the full paper [2], which considers a number of parameters including word length distribution, word frequencies, BERs, etc.

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 more than 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 [4]. 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. 1, 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. 1. 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. 2, 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.

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%

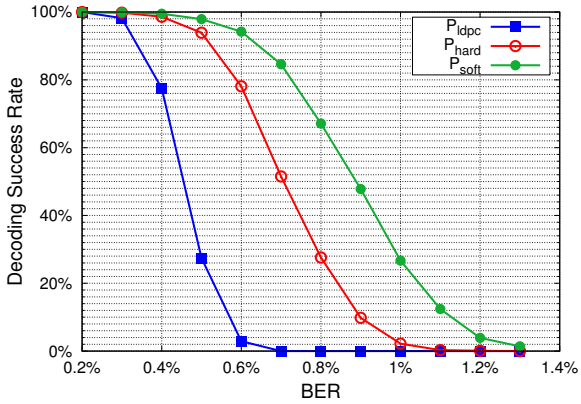

Fig. 2. The success rate of decoding with LDPC code alone (P_{ldpc}) and hard or soft language-based decoding combined with LDPC decoding (P_{hard} and P_{soft}, respectively), when the bit error probability (BER of a binary-symmetric channel) increases from 0.2% to 1.3%.

And the difference between P_{soft} and P_{hard} clearly shows the importance of soft information p_{st} and p_{un}.

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. 2, 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.

Our scheme has also been tested in a flash memory testbed using 16nm MLC NAND flash under practical settings. The results again show significant performance improvement in error correction. Interested readers can refer to [2] for details.

References

[1] A. Jiang, Y. Li and J. Bruck, "Error correction through language processing," to appear in *Proc. IEEE Information Theory Workshop*, 2015.

[2] A. Jiang, Y. Li and J. Bruck, "Enhanced error correction via language processing," full paper, 2014. Available online at *http* : *//faculty.cs.tamu.edu/ajiang/enhanced.pdf*.

[3] 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.

[4] D. MacKay, *Encyclopedia of Sparse Graph Codes*, *http* : *// www.inference.phy.cam.ac.uk/mackay/codes/data.html#l132*.