A Novel Text Processing for Better Compression and Security in Cloud

Ebru Celikel Cankaya and Hina Vinayak

***Abstract*—We introduce LG-encoding, a novel approach to text encoding that shuffles the position of letters anticipating an improved compression performance. Our technique brings together the repeating letters in a word, so as to inflate redundancy to be exploited by the compression algorithm to follow. The encoding process introduces no significant overhead: It is easily reversible as it only involves repositioning the letters in a text. We experiment LG-encoding on text from 4 different source languages: English, French, German, and Spanish with a set of well-known compression algorithms that follows the encoding: Arithmetic Coding, Huffman Coding, BWT and PPM. Our results yield promising outcomes as we achieve substantially better compression rates for Arithmetic Coding and Huffman Coding that follows LG-encoding. We also propose use of our method in large data repositories, such as cloud, as it also provides significant level of security by shuffling the letters of words in text.**

***Index Terms*—Text encoding, lossless text compression.**

I. Introduction

In an era of ultimate storage and transmission needs, it is getting more and more important to reduce the amount of data without losing the actual content. Lossless compression algorithm help accomplish this goal. Whether it is in the form of stored data (as in the case of a database, cloud, or any sort of storage media), or in the form of transmitted data (as in the case of a network communication), we need to optimize the media use to its upmost performance. So, research tries to push the theoretical upper limit more to compress text in a lossless manner. This study introduces a novel text-encoding scheme to help realize this goal: We group together the multiple occurrences of the same letter in a word in the hope to exploit redundancy better later, when we compress text afterwards. We call our method Letter Group (LG)-encoding.

We use the LG-encoding as a front end to prepare text for a lossless compression algorithm that is anticipated to yield improved performance. When an unprocessed text is input to a compression algorithm, the degree to which the text could be compressed is bound by the original position of the letters in the text. The LG-encoding we introduce repositions those letters in text -if it is part of a repeating group of letters-, causes an inflated redundancy in text, and therefore helps it compress better. Our goal is to see the practical tool that is generated as a result of this work being used for increased storage and transmission performance such as large databases, clouds, search engine retrievals and even ordinary communication on a daily basis.

The purpose of this paper is twofold: First we inquire when LG-encoding is used as a front end, how much improvement is attained in benchmark compression tools, such as Arithmetic Coding, Huffman Coding, Burrows Wheeler transform (BWT), and Prediction by Partial Matching (PPM). The BWT and PPM are not only the better performing algorithms in our compression set, but also the best tools among lossless text compression algorithms. So, this work also investigates how much, if we can, improve these better performing compression tools when preceded by LG-encoding. Second, we measure how LG-encoding performs under different source languages, so as to investigate if source language is a determinant in the performance of the compression algorithms we use.

The rest of the paper is organized as follows: Section II summarizes related work, Section III describes our scheme, Section IV presents results and comparative work, and Section V concludes and gives future insights about our design.

II. Related Work

Several scholar works on the field of focus on the same goal as ours: Preprocess text to help it compress better. One such study applies Edge-guided text compression that is based on graphs, ordered pairs and sets [1] to transform text into a word net; the adjacencies of the word have a direct relationship with the unique graph, which is the result of the word net. Our approach has less complexity as it only involves letter repositioning, rather than complex data structures as graphs.

Another text encoding introduced by [2] is to form string dictionaries by dividing text in blocks. Once the dictionaries are formed, strings are stored in a tree structure.

In Ref. [3], authors keep track of the word frequency while both comparing and separating the words with different frequency. They adaptively change the position for words to compress it better. This idea is an expansion of our approach as it extends the letter repositioning we apply to word repositioning.

The work in [4] implements Star-encoding to text, where a separate dictionary of words based on word length is used to encode actual words of the input text therefore an abundance of stars are created to help compress it better. The major overhead of this work is the requirement to store a shared dictionary on sender and receiver, which does not exist in our approach.

Manuscript received September 7, 2014; revised November 18, 2014. This work was supported by the NanoTech Institute, at the University of Texas at Dallas.

The authors are with the University of Texas at Dallas, Department of Computer Science, Richardson, TX USA (e-mail: {exc067000, hxv121530}@utdallas.edu).

DOI: 10.7763/IJCTE.2016.V8.1011

The authors in [5] exploit word based byte-oriented compression, and then transit text through character positioned compression. They use end-tagged dense code, as it is easier to build than a Huffman code.

Obviously, our scheme can be further implemented on emerging areas, such as mobile devices, where the best rates for storage and compression performances become the ultimate need [6].

III. Implementation and Results

The fundamental technique we use for LG-Encoding is illustrated in Fig. 1 below. During LG-encoding (step 2), our scheme applies a one pass scan to input text to explore multiple occurrences of the same character. For each such finding, a key is stored for the character that has multiple occurrences, together with its corresponding positions. This key and position tracks are particularly important to keep as we want our method to be reversible without any loss, i.e. the input text of step 1 should be the same as the text generated after step 8 in Fig. 1.

After LG-encoding, a compression algorithm from a repository of algorithms is selected (step 3) to compress the input text (step 4) and the output text is generated. It is this output text that we store and or transmit, instead of the original input text. To reveal the original input, first decompression is applied to output text (step 6), then LG-decoding decoding is implemented (step 7) to yield the input text.

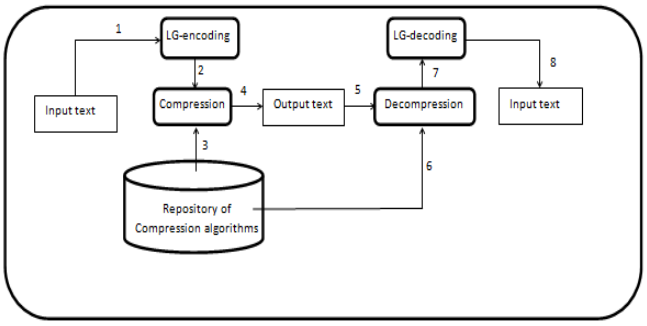

Fig. 1. Overall LG-encoding followed by compression scheme.

A. Corpora

TABLE I: English Corpora with Selected Text from Calgary Corpus and Canterbury Corpus [7], [8]

File Name	Size(Byte)	Description
Calgary		
bib	111261	Bibliography
book1	768771	Fiction book
book2	610856	non-fiction book
news	377109	USENET batch file
paper1	53161	technical paper
paper2	82199	technical paper
trans	93695	Transcript terminal session

File Name	Size(Byte)	Description
Canterbury		
asyoulike	152089	Shakespeare
alice29	125179	English text
fields	11150	C source
ice10	426754	Technical writing
plrabn12	481861	Poetry

This study presents the results of applying LG-encoding to 4 different source languages: English, French, German, and Spanish. The detail of each corpus is given in Table I through Table IV. We use the Standard English corpora: Calgary Corpus and Canterbury Corpus for English (Table I). For the rest of the source languages, we compile and use text from available corpora, whose details are presented in Table II, III, and IV. We also apply a pre-processing to each text to filter punctuation marks, non-alphabet characters and multiple occurrences of space character before LG-encoding and compression.

TABLE II: Dereko German Corpus with Selected Text from Cosmas II Database [9]

File Name	Size(Byte)	Description
doc1.txt	17720	Binnenhandel Der DDR
doc2.txt	17106	Pressemitte ilung - 132/4/70 - NRW
history.txt	16487	Bemerkungen Zur Modernen Darstellung Natinaler
horror.txt	15223	Der Scgrecken Von Takera
scope.txt	15545	Brunte Horoskop

TABLE III: French Corpus with Selected Text from Corpus of Spoken French [10]

File Name	Size(Byte)	Description
anthology	97059	Anthologle du journalisme
darwinOrigns	1391304	L'origine des espèces
dominique	442548	Dominique
football	20456	Notes sur le foot-ball (1897)
meditations	176650	Les Meditations

TABLE IV: Spanish Corpus with Selected Text from [11]

File Name	Size(Byte)	Description
pachecho.txt	78708	Pachecosy palomeques
palabras.txt	43610	Palabras y plumas
palau.txt	69133	El palau de vidre
palomares.txt	15318	Palomares(Palomares)
ramilletes.text	8307	Los ramilletes

B. Text Processing via LG-Encoding

The idea behind LG-Encoding is to bring together multiple occurrences of the same letter in a word. The algorithm traverses the text to find such occurrences and starts combining them as the traversal progresses. To assure that the algorithm is reversible, we also keep track of position placeholders for each letter moved. We use word end (space character) as the deli meter to stop encoding, which is to be followed by the next word encoding.

C. Compressing the LG-Encoded Text

We employ four widely used conventional compression algorithms to test the performance improvement of our star encoding front end on each. These algorithms are Arithmetic Coding, Huffman Coding [12], a combination of the algorithms Burrows Wheeler Transform (BWT) + Run Length Encoding (RLE) + Move to Front (MTF) + Arithmetic Coding (ARI) – we call this set BWT -, and PPM* [12]-[16].

The reason why we use these compression algorithms is that we want to compare the performance of our scheme with what is considered as benchmark compression tools. Also, we would like to see how much we could improve the performance of these tools. Among them, particularly Arithmetic Coding and Huffman Coding are more promising for improvement, as other compression algorithms in our package outperform them. The BWT and PPM are not only the better performing algorithms in our compression set, but also the best tools among lossless text compression algorithms. This work investigates how much, if we can, improve these better performing compression tools when

preceded by LG-encoding.

IV. Results

One goal of this work is to reveal how LG-encoding affects the compression performance of Arithmetic Coding, Huffman Coding, BWT, and PPM on 4 different source languages as English, French, German, and Spanish. In other words, the effect of LG-encoding to compression algorithms is measured. To reveal this, we first apply our LG-encoding scheme as a front end to text from each source language, and then run each compression algorithm to see how the compression rate is affected. All compression rates measured in bits per character (bpc).

In Fig. 2, we see that for Arithmetic Coding and Huffman Coding, LG-encoding improves the compression performance for each text, with average improvement rate of 13.9% for Arithmetic Coding and 14.24% for Huffman Coding. For already better performing compression tools BWT and PPM, though, for almost all text compression performance deteriorates when preceded by LG-encoding. Except for the file "news", the compression performance deteriorates 5.3% and 3.23% on average, for BWT and PPM, respectively.

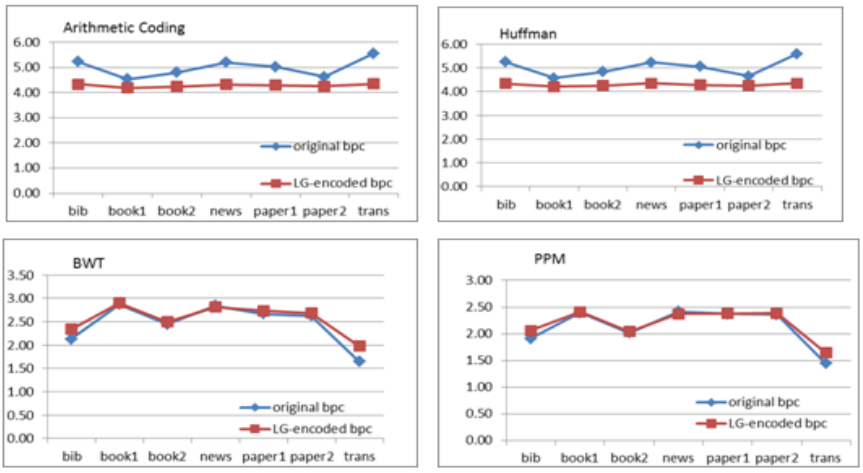

Fig. 2. LG-encoding on English Calgary Corpus.

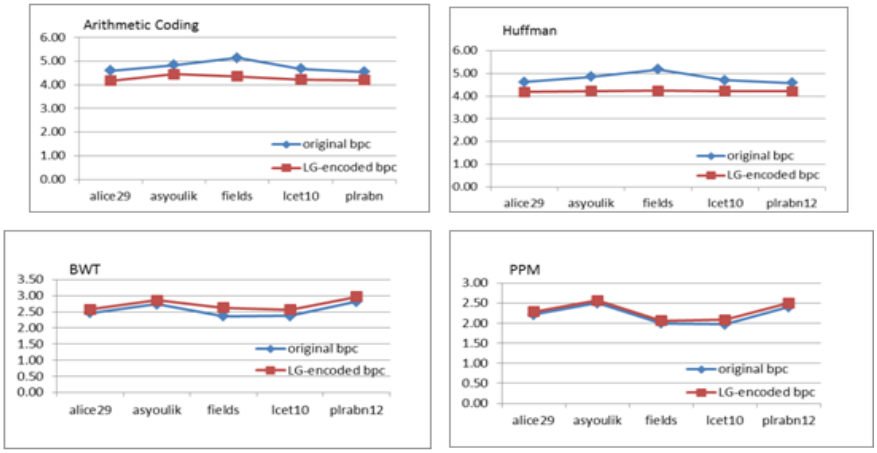

Fig. 3. LG-encoding on English Canterbury Corpus.

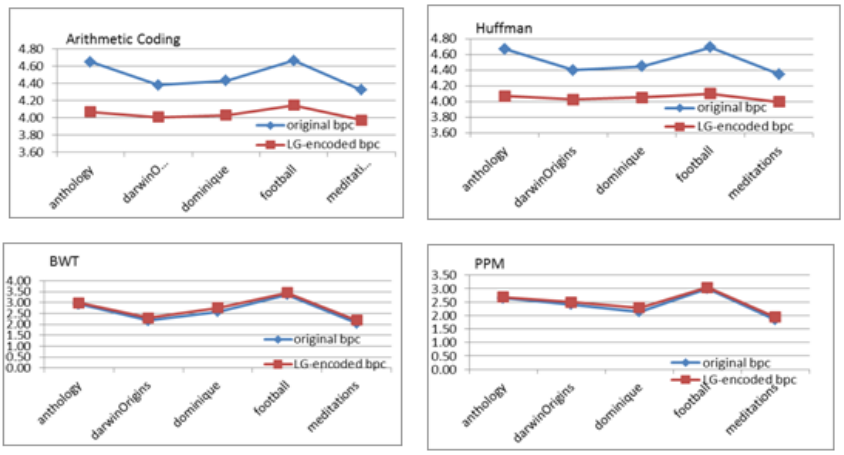

Fig. 4. LG-encoding on French Corpus.

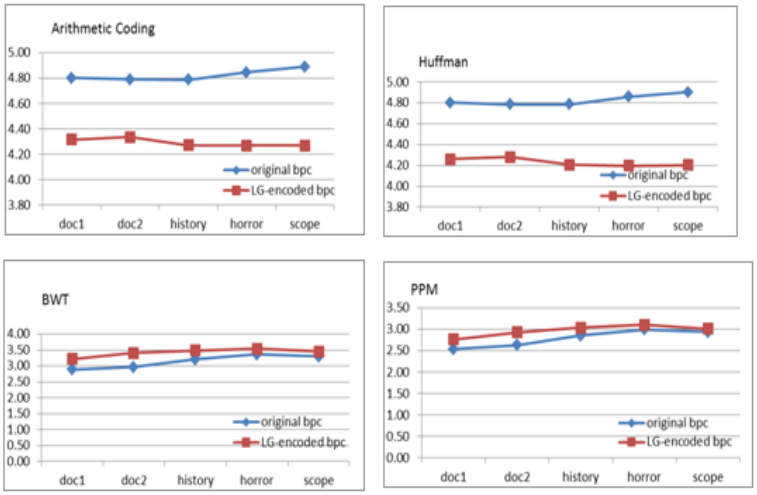

Fig. 5. LG-encoding on German Corpus.

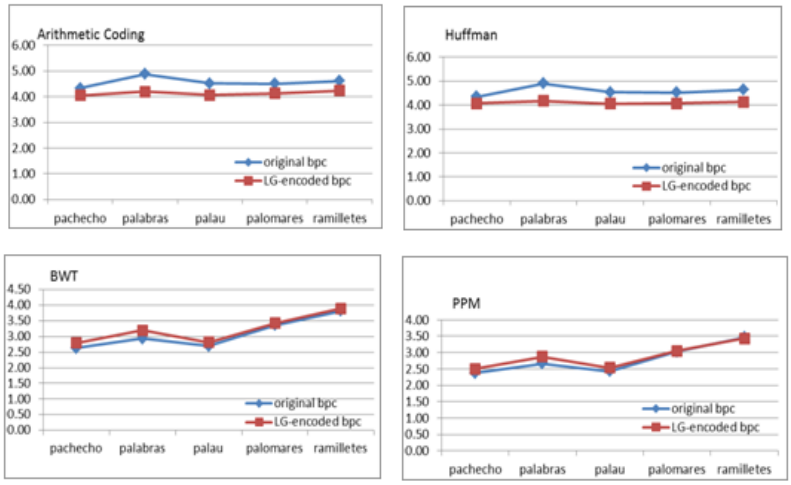

Fig. 6. LG-encoding on Spanish Corpus.

We repeat experiments on the other English Corpus: Canterbury Corpus, whose results are presented in Fig. 3.

Fig. 3 shows a similar trend in terms of compression performance for each compression algorithm: For Arithmetic Coding LG-encoding improves the compression rate by 9.90% on average, while for Huffman Coding improvement rate becomes 11.59% on average. These rates are lower than that of Calgary Corpus, simply because the text files are different now. For BWT and PPM, LG-encoding deteriorates compression rate by 6.55% and 3.70%, respectively.

For French corpus, compression performance change per algorithm is illustrated in Fig. 4.

As seen in Fig. 4, the LG-encoding improves the performance of Arithmetic Coding with an average rate of 9.84%, and the performance of Huffman Coding with an average rate of 10.18%. The LG-encoding worsens the other two compression algorithms with average rates of 4.82%, and 3.76%.

When the source language is German, the compression rate plots with and without LG-encoding are illustrated in Fig. 5.

LG-encoding affects performance of compression algorithms similarly: Arithmetic Coding and Huffman Coding improves with average rates of 10.98% and 12.36% (the highest so far), and BWT and PPM worsens with average rates of 9.09% and 6.56% (the lowest so far).

For Spanish being the source language, LG-encoding exhibits a similar result as seen in Fig. 6.

The change in compression performance is consistently positive for Arithmetic Coding and Huffman Coding, as we observe an improvement after LG-encoding in these algorithms for each source language (Fig. 2 through 6). However, the consistent deterioration in BWT and PPM performances could be an indication that for these algorithms the theoretical upper limit is already achieved and LG-encoding cannot make their compression rates any better. To further investigate the effect of LG-encoding per algorithm, we compare the average rate of change in compression performance for each source language. Table V through 9 present the resulting compression rates measured in bits per character (bpc).

TABLE V: THE EFFECT OF LG-ENCODING ON ARITHMETIC CODING

Language	plain	LG-encoding	% change
ENG-Calgary	5.00	4.29	14.22%
ENG- Canterbury	4.76	4.28	10.01%
French	4.49	4.05	9.89%
German	4.82	4.29	10.99%
Spanish	4.57	4.13	9.47%

Table V indicates that for Arithmetic Coding, the LG-encoding improves compression performance for each source language. The improvement rates are very close, where English leads with an average of 12.12%

improvement, and is followed by German with 10.99%, then French with 9.89%, and Spanish with 9.47% improvement achieved after employing LG-encoding. The results show that for Arithmetic coding, the most sensitive language to LG-encoding is English, and the least sensitive is Spanish.

The effect of LG-encoding on Huffman Coding has a similar behavior as seen in Table VI.

TABLE VI: THE EFFECT OF LG-ENCODING ON HUFFMAN CODING

Language	plain	LG-encoding	% change
ENG-Calgary	5.02	4.29	14.56%
ENG- Canterbury	4.79	4.22	11.76%
French	4.51	4.05	10.25%
German	4.83	4.23	12.37%
Spanish	4.59	4.10	10.60%

As seen in Table VI, for Huffman coding English continues to be the language that is most improved by LG-encoding with an average rate of 13.16%. German follows with 12.37%, then Spanish and French follows with rates 10.60% and 10.25%, respectively.

Both Table VII and Table VIII show that the LG-encoding deteriorates the compression performance for BWT and PPM in all source languages. When the algorithm is BWT, the worsening happens with the rates for German being the highest, then for English (when we take the average of Calgary and Canterbury corpora), then for French, and for Spanish (Table VII). When the algorithm is PPM, the rates of deterioration in compression performance are not as much as that of BWT: The highest rate of deterioration is recorded for German with 6.38%, then for French with 3.44%, then for English with an average rate (for Calgary and Canterbury corpora) and for Spanish both with 3.06%. The difference in LG-encoded compression rate for the same language attributes to the linguistic characteristics of the text being encoded and then compressed.

TABLE VII: THE EFFECT OF LG-ENCODING ON BWT

Language	plain	LG-encoding	% change
ENG-Calgary	2.46	2.56	-4.21%
ENG- Canterbury	2.55	2.71	-6.42%
French	2.62	2.74	-4.53%
German	3.14	3.42	-8.88%
Spanish	3.09	3.23	-4.44%

TABLE VIII: THE EFFECT OF LG-ENCODING ON PPM

Language	plain	LG-encoding	% change
ENG-Calgary	2.13	2.18	-2.46%
ENG- Canterbury	2.22	2.30	-3.66%
French	2.41	2.49	-3.44%
German	2.79	2.97	-6.38%
Spanish	2.79	2.88	-3.06%

This result could be an indication that BWT as well as PPM already perform very well on plain text and any further encoding does not help improve this rate.

V. CONCLUSIONS AND FUTURE WORK

We introduce a new encoding scheme called LG-encoding that exploits the idea that if multiple occurrences of the same letter are brought together, redundancy will increase to help improve compression performance. To investigate the effect of LG-encoding on text, we employ 4 different compression algorithms (Arithmetic Coding, Huffman Coding, BWT, and PPM) on 4 source languages (English, French, German, and Spanish). We propose use of our method in large data repositories, such as cloud, as it also provides significant security by shuffling the letters of words. We measure the rate of change in compression performance for 4 algorithms and the results show that LG-encoding improves the performance of Arithmetic Coding in the range between 14.22% to 9.47% for each source language. It also improves the performance of Huffman Coding within a range 14.56% to 10.25% for each source language. For BWT and PPM, LG-encoding deteriorates, rather than improving compression performance at rates between 4.44% to 8.88% for BWT and between 3.06% to 6.38% for PPM.

We also observe that for each source language, the behavior of the scheme (compression preceded by LG-encoding) is similar: A significant improvement for Arithmetic Coding and Huffman Coding, and deterioration (with less significance than the improvement rates) for BWT and PPM.

As for future work, we plan on extending the LG-encoding to include all letters of the text, rather than words only, in the hope to get even better compression. We also consider employing other compression algorithms to see how their performance will be affected by LG-encoding.

Moreover, we consider encrypting the LG-encoded text further with a symmetric encryption algorithm (e.g. AES) to offer even better security in cloud. With a cloud based encryption on LG-encoded text followed by a hash algorithm, we expect to speed up the encryption process. Furthermore, we plan on a client based LG-encoding so as to improve network utilization.

REFERENCES

[1] M. A. Martinez-Prieto, J. Adiego, and P. Fuente, "Natural language compression on edge-guided text processing," *Journal of Information Sciences*, vol. 181, no. 24, Santiago, Chile, 2011.

[2] M. A. Martinez-Prieto, R. B. Nieves, N. Gonzalo, and C. Rodrigo, *Compressed String Dictionaries*, Mideplan, Chile, 2011.

[3] N. Brisaboa, A. Fariña, N. Gonzalo, and J. Paramá, "Dynamic lightweight text compression," *ACM Transactions on Information Systems*, pp. 1-32, 2010.

[4] C. E. Cankaya and O. Darwish, "Improving compression performance with a star encoding front end: A linguistic comparison," in *Proc. 2013 World Comp.*, Las Vegas, NV, 22-25 July, 2013.

[5] A. Fariña, G. Navarro, and J. Paramá, *Boosting Text Compression with Word-Based Statistical Encoding*, Oxford University Press, Oxford, UK, 2011, pp. 112-118.

[6] R. Pizzolante, B. Carpentieri, A. Castiglione, and F. Palmieri, "Text compression and encryption through smart devices for mobile communication," *IMIS*, pp. 672-677, 3-5 July, 2013.

[7] English Calgary Corpus. [Online]. Available: http://corpus.canterbury.ac.nz/descriptions/#calgary

[8] English Canterbury Corpus. [Online]. Available: http://corpus.canterbury.ac.nz/descriptions/#cantrbry

[9] German Corpus URL: COSMAS II, Institute for Deutsche Sprache. [Online]. Available: http://www.ids-mannheim.de/cosmas2/projekt/registrierung/

[10] French Corpus URL: Corpus of Spoken French, Centre for Languages, Linguistics, & Area Studies. University of the West England. [Online]. Available: http://www.llas.ac.uk/resources/mb/80

[11] Spanish Corpus. [Online]. Available: http://www.cervantesvirtual.com/bib/seccion/literatura/psegundonivel d6b2.html?conten=catalogo

[12] R. Ahlswede, A. Ahlswede, I. Althöfer, C. Deppe, and U. Tamm, "Data compression," *Storing and Transmitting Data Foundations in Signal Processing, Communications and Networking*, vol. 10, pp. 9-38, 2014.
[13] J. Kelley and R. Tamassia, "secure compression: Theory & practice," *IACR Cryptology*, 2014.
[14] D. Zhang, Q. Liu, Y. Wu, Y. Li, and L. Xiao, "Compression and indexing based on BWT: A survey," in *Proc. Web Information System and Application Conference*, Yangzhou, China, November 2013.
[15] W. Oliveira, E. Justinoa, and L. S. Oliveira, "Comparing compression models for authorship attribution," *Journal of Forensic Science International*, vol. 228, pp. 100-104, 2013.
[16] H. Kruse and A. Mukherjee, "Preprocessing text to improve compression ratios," in *Proc. DCC '98*, 1998, p. 556.

Ebru Celikel Cankaya is a senior lecturer in the Department of Computer Science at the University of Texas at Dallas (UTD) where she has been a faculty member since 2012. Dr. Cankaya has completed her postdoctoral research at the same institute in the area of database security. She has received her Ph.D. degree from Ege University, Turkey. Before joining to UTD, she has worked as a faculty member at University of North Texas, Earlham College, and Ege University. Her research interests lie in the area of computer and database security, lossless compression, and cryptology.

Hina Vinayak is doing her bachelors in telecommunications engineering at The University of Texas at Dallas in Texas, USA. She is interested in research related to computer technology, compression algorithms, and networking. She plans to pursue her career in the telecommunications industry and in academia with MBA with concentration in information technology.