Conjugation-Based Compression for Hebrew Texts

YAIR WISEMAN and IRIT GEFNER
The Hebrew University of Jerusalem

Traditional compression techniques do not look deeply into the morphology of languages. This can be less critical in languages like English where most of the sequences are illegal according to the grammatical rules of the language, for example, zx, bv or qe; hence the morphology can add a little information that can be beneficial for the compression algorithm. However, this negligence can be a significant flaw in languages like Hebrew where the grammatical rules allow much more freedom in the sequences of letters and, except tet after gimel, any pair is legal; hence compressing without taking the morphological rules into account can yield a poorer compression ratio. This article suggests a tool that optimizes the Burrows-Wheeler algorithm which is an unaware morphological rules compression method. It first preprocesses a Hebrew text file according to the Hebrew conjugation rules, and, after that, it provides the Burrows-Wheeler algorithm with this preprocessed file so that can be compressed better. Experimental results show a significant improvement.

Categories and Subject Descriptors: E.4 [**Coding and Information Theory**]: Data Completion and Compression

General Terms: Algorithms, Languages

Additional Key Words and Phrases: Text compression, Burrows-Wheeler algorithm, Hebrew text analysis, Root conjugations, semitic languages

ACM Reference Format:

Wiseman, Y. and Gefner, I. 2007. Conjugation-based compression for Hebrew texts. ACM Trans. Asian Lang. Inform. Process. 6, 1, Article 4 (April 2007), 10 pages. DOI 10.1145/1227850.1227854 http://doi.acm.org/10.1145/1227850.1227854

1. INTRODUCTION

Since 1948, many text compression techniques have been developed. Some of these methods are statistical, that is, they look at each item in the text and its frequency like Shanon-Fano Coding [Shannon 1948], Huffman Coding [Huffman 1952; Bookstein and Klein 1993] and Arithmetic Coding [Rubin 1979; Witten et al. 1987], whereas other techniques look at strings in the text and put

Authors' address: School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904 Israel; email: wiseman@cs.huji.ac.il.

pointers to strings or substrings that have been already appeared, for example, Lempel-Ziv Codes [Ziv and Lempel 1977, 1978]. There are also techniques like PPM [Moffat 1990] which look at the frequency of the item, and also look at the item in its neighborhood when they decide how to encode an item.

Compression methods are also often partitioned into static and dynamic methods. The static methods assume that the file to be compressed has been generated according to a certain model which is fixed in advance and known to both compressor and decompressor. The model could be based on the probability distribution of the different characters or, more generally, of certain variable length substrings that appear in the file, combined with a procedure to parse the file into a well-determined sequence of such elements. The encoded file can then be obtained by applying some statistical encoding function, such as Huffman or Arithmetic Coding. Information about the model is either assumed to be known (such as the distribution of characters in an English text), or it can be gathered in a first pass over the file so that the compression process can be performed in a second pass. Many popular compression methods, however, are adaptive in nature. The underlying model is not assumed to be known but rather discovered during the sequential processing of the file. The encoding and decoding of the element is based on the distribution of the i-1 preceding ones so that compressor and decompressor can work in synchronization without requiring the transmittal of the model itself. Examples of adaptive methods are the Lempel-Ziv (LZ) methods and their variants, but there are also adaptive versions of Huffman and arithmetic coding. We would like to focus in this article on the Burrows-Wheeler Algorithm [Burrows and Wheeler 1994; Nelson 1996] which is an adaptive method in most of its steps.

According to [Wirth and Moffat 2001], the best-known text compression technique is Burrows-Wheeler Algorithm, hence we have chosen to employ it. This technique is very sensitive to the word structure of any language so the compression efficiency achieved by this technique can be dissimilar when using different languages. However, this technique is not sensitive to word conjugations that are stemmed from the same root, hence it lacks the ability to utilize the morphology of the language. This is very important in Hebrew where the rules for root conjugation are very complex and a Hebrew word affixation can be done in many ways [HaCohen-Kerner et al. 2003].

Some works for compression enhancement have been suggested. Some of them are based on syntax aware techniques, for example, Changsong et al. [1998] and some are based on morphological rules like Diri [2000, 2001] and Hakkani-Tur et al. [2002] in agglutinative languages like Turkish. Our work suggests a conjugation-based technique which is a kind of a morphological method. It should be noted that this suggested technique fits only Semitic languages like Hebrew or Arabic. On Arabic, there is a work of Yaghi at el. [2003] that suggests a dictionary of Arabic roots. We suggest another way to handle this issue. Our method saves the patterns of the words instead of the roots. In addition, we use the Burrows-Wheeler algorithm, unlike Yaghi et al. [2003].

The rest of this article is organized as follow. The next section explains how the Burrows-Wheeler algorithm works and why it does not produce good results

for Hebrew text. Section 3 surveys some known researches about the Hebrew language. Section 4 introduces the conjugation-based compression for Hebrew texts and describes the fundamental themes of this algorithm and explains why this algorithm produces better results. This section also discusses the pros and cons of the proposed algorithm. Finally, the Section 5 presents the results and the Section 6 summarizes the conclusions of this article.

2. BURROWS-WHEELER COMPRESSION

2.1 Burrows-Wheeler Algorithm

The Burrows-Wheeler algorithm is a dictionary compression method. This method utilizes repetitions of words' sequences in order to improve the compression ratio. No information is lost in the compression procedure. This method usually outperforms Lempel-Ziv coding, hence it is widely use in a variety of compression utilities. The main imperfection of this method is its high execution time, but this imperfection is not critical to many applications. The popular bzip/bzip2 utility [Seward 2005] is based on this algorithm.

The Burrows-Wheeler algorithm has several steps. In the first step, the algorithm creates pointers to all characters of the file that is compressed. The pointers are sorted according to the characters that they point to. The preceding characters of each pointed character are sent to the next step according to the order of the sorted pointers. Actually, this newly-generated sequence of characters in the output has the same characters as the original file has, but the order of the characters is different.

In the second step, the algorithm performs move-to-front. It keeps all the 256 possible characters in a list. Each character in the sequence is read and its position in the list is sent. After the character is sent, it will be moved from its current position in the list to the front of the list (i.e., to position 0).

In the next step, the algorithm applies a run-length coding to the output of the previous step. The output of the run-length coding is then compressed using either Huffman or Arithmetic Coding.

The main disadvantage of the Burrows Wheeler transform is its slow execution time due to the file sorting. In order to reduce this time, files are split into blocks. This may harm the compression ratio because shorter files are less effectively compressed, but usually this harm is very small. This article uses the SGI [2003] version of the Burrows-Wheeler algorithm.

2.2 Burrows-Wheeler Algorithm and the Hebrew Language

As mentioned previously, the Burrows-Wheeler Algorithm is currently the best among the known methods for text compression [Wirth and Moffat 2001]. However, this method will produce much better results if the data is an English text, whereas if the data is a Hebrew text, the results will not be as good. As an example, the Burrows-Wheeler Algorithm compresses the King James Version of the English Bible (only Old Testament) into 20.89% of its original size, while the Hebrew Bible (only Old Testament) is compressed by the Burrows-Wheeler Algorithm into just 34.04% of its original size.

The reason for the difference between the compression efficiency is the dissimilarity of the root conjugation rules of English and Hebrew. English leave roots unchanged in the words, for instance, *clear*, *clear*ly, un*clear,* whereas Hebrew allows many letters to be inserted within a root and still make it a valid Hebrew word, for example, the root {Shin, Mem, Reish} means in Hebrew to keep and the letter "Vav" can be inserted before the root, after the root, and between any two letters in the root, and all of these combinations are valid Hebrew words.

—Inserting the letter before the root means "and kept".
—Inserting the letter after the first letter and before the second letter means "keeper".
—Inserting the letter after the second letter and before the third letter is the imperative form of the verb "keep".
—Inserting the letter after the root means "kept" in plural.

This feature of the Hebrew language causes a wide diversification of preceding letters for each string in the language and the 'move-to-front' algorithm yields unsuitable data for an efficient operation of the run-length function.

3. THE HEBREW LANGUAGE

This section briefly surveys the structure of the Hebrew language and some research that has been conducted on the formation of the language.

3.1 The Hebrew Conjugations

The Hebrew language is based on roots of two, three, or four letters. These roots are the basis of the verbs, nouns, and adjectives. The roots are conjugated by inserting letters before, after, or inside the roots. Hebrew has four tenses, namely, past, present, future, and imperative, and they are conjugated in three persons (first, second and third), in singular or plural and in masculine or feminine. The later should be noted because English verbs and adjectives in singular and plural are identical and also verbs and adjectives in masculine and feminine are identical, whereas Hebrew verbs and adjectives are differently affixed in singular and plural and also differently affixed in masculine and feminine.

In addition, Hebrew has seven conjugations for passive, active, and semipassive verbs, and twenty-seven patterns for nouns and adjectives. There are also several prefixes and suffixes which can mean connection, relation, or ownership.

3.2 Diacritics

The Hebrew diacritic is a sign over, under, inside, or to the left of the letter. These signs replace the role of the vowels in English, that is, the diacritic lets us know how to pronounce the Hebrew words correctly.

Usually Hebrew text does not contain any diacritic and the readers are expected to deduce them themselves. A good example of text with diacritic is children books where diacritics typically appear in order to help the children to

read the text correctly. We will not deal with diacritics in this article because it is not common in Hebrew texts.

3.3 Homographs

There are Hebrew words that are written in the same way, but have different meanings. For instance, the word {Aleph, Mem} in Hebrew means "if" and also means "mother". This feature can help the compression because, if theoretically all the words in Hebrew were written in the same way, we could indicate just how many words there are and write this number as the compressed file.

3.4 Research and Tools

Much research has been conducted over the years concerning natural language processing in Hebrew. We took some ideas from this body of work. However, the aims of these research projects were different from ours as we detail in this section.

The project[1] [Choueka et al. 1971; Choueka 1980] searches for words in a huge database including texts in ancient Hebrew, conversational Hebrew from different periods, modern Hebrew, and Aramaic. All the texts lack diacritics, and many of them lack punctuation such as commas and dots as well.

The project has been active since its inception in 1964, and it contains a grammatical synthesis that creates all the possible affixation for any given word [Attar et al. 1978]. In addition, this tool lets the user see the root and all the conjugations of the searched word before the searching starts. This feature helps the user to choose the desired affixations. The project has gained an accuracy of 34%, but the research and the improvements have been continuing.

A similar tool for finding roots was developed at 1984 by Daniel Cohen [1984]. This tool extracts the possible roots and finds for each of these roots all the possible conjugations. Then it chooses the best interpretation of the word according to the adjacent context. The tool has an accuracy of 50%.

The possible affixations for a word is also an issue that has been addressed by other research, for example, HUHU [Nirenburg and Ben-Asher 1984] has a dictionary that contains the potential affixations for any possible word and the rules for adding prefixes and suffixes.

In 1985, Choueka and Serge suggested a method of finding the meaning of a word. The method creates a dictionary of problematic words along with the contexts that these words can appear in and the meaning of the word in each of the contexts. In 1990, Choueka introduced Rav-Milim. This tool automatically generates a morphological analysis of the words in the text and accordingly automatically inserts diacritics into the text. The morphological analysis also improves the search of words within the analyzed text.

The syntax of the sentence can also be of help and, in 1991, Wintner and Oman suggested an improved technique to find the context of words which was expanded by Wintner in 2004. This technique analyzes the syntax of a sentence and concludes, according to this analysis, what the context of each

[1]Judaic Responsa, at http://www.biu.ac.il/JH/Responsa

word in the text is. The main disadvantage of this technique is its long execution time.

In 1992, IBM developed a new tool that has a dictionary containing the conjugations of almost all the roots of Hebrew [Bentur et al. 1992]. In 1995, Levinger et. al. suggested reducing the dictionary and having only a partial dictionary by using some statistics instead of the missing words.

As shown in this brief survey, the Hebrew language has been researched for some decades. We would like to take some ideas from these research projects in order to build a new compression technique.

4. CONJUGATION-BASED COMPRESSION

As mentioned, most of the words in Hebrew consist of roots affixed in some know patterns. The new compression method described in this article will be based on this feature.

4.1 The Compression Algorithm

The compression is done in two steps. The first step consists of reading of the source file and splitting it into two separate files. One of the files contains the patterns of all the words, and the other file contains the roots of the words. The second step consists of compressing both of the files with the Burrows-Wheeler algorithm. In this way, we address the complex pattern mechanism of the Hebrew language and are able to obtain a better compression ratio. The letters of the patterns that interfere with the roots are extracted, and the traditional Burrows-Wheeler compression can be better utilized as in less complex languages. The structure of the pattern vector and the root file is different, so it is better to compress them separately in order to obtain a better compression ratio.

It should be noted that the analysis of the words does not have to be correct as it has to be in some works mentioned in the previous section. Our aim is a better compression; thus even if the analysis states that one word is a noun while actually the word is an adjective, the compression will not be harmed by that, and the reconstruction of the word in the decompression stage will be completely correct. Words that do not fit any of the patterns are written into the roots file.

4.2 Pattern Selection

There are some rare patterns in Hebrew which we do not like to have in our pattern database. If we use all the Hebrew patterns, the pattern vector will be too large and such a large vector can damage the compression efficiency. We actually used a dynamically built pattern vector that is different for any kind of text.

4.3 Word Definition

Unlike many other compression techniques, the technique in this article is based on words. So, the decision of where a word starts and where a word ends is very essential.

A word is any sequence of Hebrew letters. The other characters are the delimiters. Another approach would be to treat comma, brackets, dots etc., as a part of the patterns. This would have increased the number of patterns, so in order to avoid this increase, we do not include these characters in the patterns.

4.4 Terminal Letters

There are five letters in Hebrew that are written differently when they appear at the end of a word. If we wrote these letters as they are, they could slightly harm the compression efficiency. Since we analyze the word boundaries, we can know when the letter is at the end of a word and when the letter is not at the end of a word and write the letter accordingly.

The flaw in this approach occurs when someone decides to break the Hebrew rules for any reason and decides to write a regular letter at the end of a word instead of writing the special letter. In such a case, we put a special indication in the file stating that this letter is an exception. This indication consumes more space and we count on its rareness.

5. RESULTS

In order to check the feasibility of this idea, we have implemented this compression algorithm. We have used the beginning of the Hebrew bible for the compression tests. The size of this file was 260kB. The algorithm split the file into two files. The pattern file was 68kB, and the root file was 146kB. The use of a pattern vector gives an initial compression. We write just the index of the pattern instead of writing the entire pattern. This reduced the size of the data to 82.58% of the original size.

The second compression step reduced the size of the pattern file to 531 bytes, just 0.776% of the original size, and the size of the root file into 74kB, 51.11% of the original size.

The overall target/source file size ratio using the technique suggested in this article is 28.97%, whereas the traditional Burrows-Wheeler technique compresses this file to 40.13%. The block size of Burrows-Wheeler was set to be bigger than the actual size of both the pattern file and the root file, that is, both of the files were compressed as one block.

The Hebrew language contains a lot of patterns; some of them are very rare. Our implementation chooses only the most common patterns. Sometimes there are words that can fit more than one pattern so, in order not make an NP-Hard algorithm, we have used a greedy algorithm that has chosen the pattern that seems to contribute the maximum space-saving at selection time.

The patterns were recorded so that we could indicate where the letters of the root had been and where the additional letters had been. In this way, when we reconstructed the word, we could know where the letters of the root should be put, and where the letters of the pattern should be put.

Figure 1 shows the cost effect of the patterns. All the numbers in this figure are the logarithm (base 2) of the original number, that is, the X-axis is the logarithm of the number of the patterns that have been chosen (64, 128, . . . , 2048). This figure contains two kinds of information, hence we added two titles for

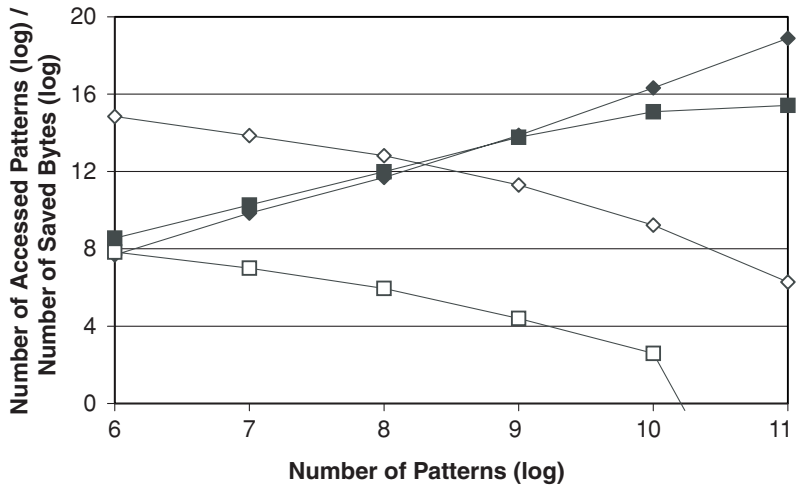

Fig. 1. Cost effect of patterns.

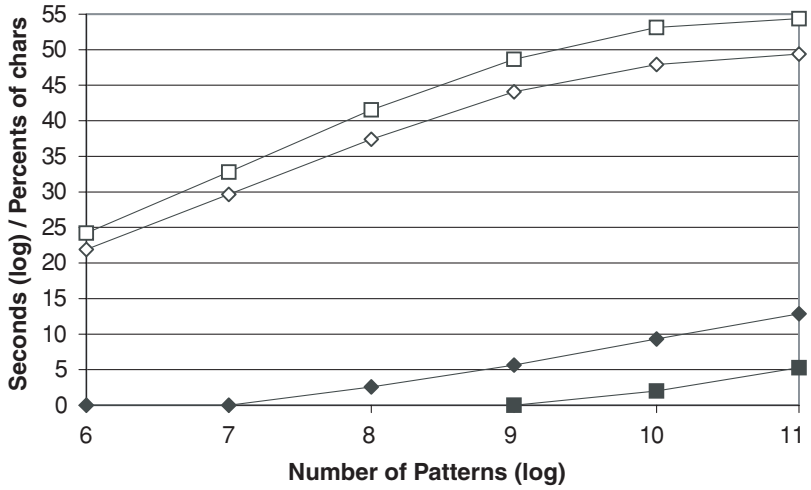

Fig. 2. Time efficiency.

the Y-axis separated with "/". The filled shapes represent the logarithm of the number of the patterns that have been checked. The greedy algorithm checks several potential patterns for each word. The numbers in the chart (in the Y-axis) are the logarithm of the sum of the numbers of the checked patterns for all the words in the compressed text. Since adding some patterns can contribute nothing to the compression or even damage it, we do not include the noncontributory patterns in the pattern list. The hollow shapes represent the logarithm of the smallest contribution that the last pattern added. The contribution is specified in logarithm of the bytes that were saved because of the last pattern. It can be easily seen that when adding many patterns, the contribution will be diminished.

The diamonds in Figure 1 represent a modern Hebrew literature of 4 millions words written by the popular Israeli author Ram Oren, while the squares represent the text of several sessions of the Israeli parliament which contains almost 30,000 words.

Figure 2 shows the time efficiency of the proposed technique. The X-axis is the logarithm of the number of the patterns as in Figure 1. This figure also contains two kinds of information, hence we added again two titles for the Y-axis separated with "/". The filled shapes represent the logarithm of the execution time in seconds. The tests were executed on a Pentium 4 machine with 768MB. The hollow shapes represent the percent of letters that were not put in the root file, that is, the letters that comprise the patterns. This information is not logarithmic and is specified as the ratio between the characters that were not put in the root file and all the characters. The diamonds and the squares represent the same databases as in Figure 1. It can be noted that increasing the number of the patterns will exponentially increase the execution time, while the compression efficiency will be just slightly improved.

6. CONCLUSIONS

A novel technique for Hebrew text compression optimization has been shown. The algorithm tries to understand the text instead of just unintelligently compressing it. This technique is good for any language like Hebrew that has letters that interfere within the root's letters. Such languages like Hebrew or Arabic can optimize their compression efficiency using this morphological analyzing technique.

REFERENCES

ATTAR, R., CHOUEKA, Y., DERSHOWITZ, N., AND FRAENKEL, A. S. 1978. KEDMA—Linguistic tools for retrieval systems. *J. ACM, 25*, 1, 52–66.

BENTUR, E., ANGEL, A., SEGEV, D., AND LAVIE, A. 1992. Analysis and generation of the nouns inflection in Hebrew. In *Hebrew Computational Linguistics*, Ornan, Arieli and Doron, Eds., 3, 36–38.

BOOKSTEIN, A. AND KLEIN, S. T. 1993. Is Huffman coding dead?, *Comput. 5*, 279–296.

BURROWS, M. AND WHEELER, D. J. 1994. A block-sorting lossless data compression algorithm. Tech. rep. 124, Digital System Research Center, Palo Alto, CA.

CHANGSONG, X., ECK, P., AND MATZNER, R. 1998. Syntax-oriented Coding (SoC): A new algorithm for the compression of messages constrained by syntax rules. In *Proceedings of the International Symposium on Information Theory*. Cambridge, MA. 16–21, 317.

COHEN, D. 1984. Mechanical syntactic analysis of a Hebrew sentence. Ph.D. thesis, Hebrew University of Jerusalem.

CHOUEKA, Y. 1980. Computerized full-text retrieval systems and research in the humanities: The Responsa project. *Comput. Humanities 14*, 153–169.

CHOUEKA, Y. 1990. MILIM—A system for full, exact, on-line grammatical analysis of modern Hebrew. In *Proceedings of the Annual Conference on Computers in Education*. Y. Eizenberg Ed., Tel Aviv, April, 63.

CHOUEKA, Y., COHEN M., DUECK J., FRAENKEL A. S., AND SLAE M. 1971. Full text document retrieval: Hebrew legal texts (report on the first phase of the Responsa retrieval project). *Annual ACM Conference on Research and Development in Information Retrieval*. College Park, MD.

CHOUEKA, Y. AND SERGE, L. 1985. Disambiguation by short context. *Comput. Humanities 19*, 147–157.

DIRI, B. 2000. A text compression system based on the morphology of turkish language. The *15th International Symposium on Computer and Information Sciences*. Istanbul, Turkey.

DIRI, B. 2001. Content-Based compression of Turkish documents. *Pakistan J. Appl. Sci. 1*, 4, 446–451.

HACOHEN-KERNER, Y., MALIN, E., AND CHASSON, I. 2003. Summarization of Jewish law articles in Hebrew. In *Proceedings of the 16th International Conference on Computer Applications in Industry and Engineering (ISCA)*. Las Vegas, NV. 172–177.

HAKKANI-TUR, D. Z., OFLAZER, K., AND TUR, G. 2002. Statistical morphological disambiguation for agglutinative languages. *Comput. Humanities 36*, 381–410.

HUFFMAN, D. A. 1952. A method for construction of minimum-redundancy codes. In *Proceedings of the Institute of Radio Engineers (IRE) 40*, 9, 1098–1101.

LEVINGER, M., ORNAN, U., AND ITAI, A. 1995. Morphological disambiguation in Hebrew using Apriori Probabilities. *Computat. Linguist. 21*, 383–404.

MOFFAT, A. 1990. Implementing the PPM data compression scheme. *IEEE Trans. Commun. 38*, 11, 1917–1921.

NIRENBURG, S. AND BEN-ASHER, Y. 1984. HUHU—the Hebrew University Hebrew Understander. *Comput. Lang. 9*, 3/4.

NELSON, M. 1996. Data compression with the Burrows-Wheeler transform. *Dr. Dobb's J.*, (Sept.) 46–50.

SEWARD, J. 2005. bzip2 and libbzip2—A program and library for data compression. http://www.bzip.org/1.0.3/bzip2-manual-1.0.3.html.

SGI® IRIX® FREEWARE DISTRIBUTION. 2003. 1600 Amphitheatre Pkwy. Mountain View, CA, USA, Feb. Ed.

SHANNON, C. E. 1948. A mathematical theory of communication. *Bell Sys. Tech. J. 27*, 379–423; 623–656.

RUBIN, F. 1979. Arithmetic stream coding using fixed precision registers. *IEEE Trans. Infor. Theory 25*, 6, 672–675.

WINTNER, S. 2004. Hebrew computational linguistics: Past and future. *Artif. Intell. Rev. 21*, 2, 113–138.

WINTNER, S. 1991. Syntactic analysis of Hebrew sentences. Master's thesis. Technion, Israel Institute of Technology, Haifa, Israel.

WINTNER, S. AND ORNAN, U. 1992. Syntactic analysis of Hebrew sentences. *In Proceedings of the 8th Israeli Symposium on Artificial Intelligence and Computer Vision*. Information Processing Association of Israel, 201–230.

WIRTH, A. AND MOFFAT, A. 2001. Can we do without ranks in Burrows Wheeler transform compression? *IEEE Data Compression Conference*. 419–428.

WITTEN, I. H., NEAL, R. M., AND CLEARY, J. G. 1987. Arithmetic coding for data compression. *Comm. ACM* 30, 6 (June), 520–540.

YAGHI, J. 2004. Computational Arabic verb morphology: Analysis and generation, MS thesis. University of Auckland, Auckland, New Zealand.

YAGHI, J., M. R., TITCHENER, AND S. M., YAGI. 2003. T-Code compression for Arabic computational morphology. In *Proceedings of the Australasian Language Technology Workshop*. Melbourne, Australia.

ZIV, J. AND LEMPEL, A. 1977. A universal algorithm for sequential data compression. *IEEE Trans. Inform. Theory* 23, 3 (May), 337–343.

ZIV, J. AND LEMPEL, A. 1978. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory* 24, 5 (Sept.), 530–536.

Received February 2006; revised January 2007; accepted February 2007