Parsing Strategies for BWT Compression

R. Yugo Kartono Isal & Alistair Moffat

Presented by Thorin Tabor

Burrows-Wheeler Transform: In Practice

- Encoding with the Burrows-Wheeler Transform typically has three steps:
  - Apply the BWT algorithm to the text to permute it.
  - Apply a Move-to-Front algorithm to the text.
  - Finally, do some sort of entropy encoding to the result, such as Huffman encoding or Arithmetic encoding.
- Sometimes run-length codes are introduced between any of these steps.

BWT in Practice: Part II

- The BWT algorithm permutes the text into a pattern that is more easily encoded by creating runs and areas of repeated characters in the string.
- The MTF algorithm rearranges the pattern such that more frequently used characters tend to be near the front of the string and less frequently used characters tend to be towards the end.
- Entropy encoding (like Huffman encoding) compresses the text with some form of lossless compression.

Move-to-Front Algorithm

- Transverses a string from start to end, looking the index of the encountered character in an array that starts as [1...n] in order.
- When a character is encountered in encoding the string, it moves that character to the front of the string, assigns new indexes and continues.

To Encode: 524700717

Initial List: (0,1,2,3,4,5,6,7)

Step 1: The first number in the sequence is 5, which appears at index 5. We add a 5 to the output.

New Output: 5
New List: (5,0,1,2,3,4,6,7)

Step 2: The next number is 2, which now appears at index 3. Add to output and permute list.

New Output: 53
New List: (2,5,0,1,3,4,6,7)

Etc...
Final Output: 535740151

The Unit of Transmission

- Most studies of BWT compression involve transmuting the text by rearranging blocks of bites that correspond to ASCII characters.
- There is no reason this unit of partition has to be so and a number of other parsing strategies exist that could be used.
- For example, we could partition the text a series of words instead of a series of ASCII characters.
- Using another parsing strategy may result in better compression.

The Aim of This Paper

- This paper examines an alternate way of parsing the text and permuting it other than by ASCII character.
- To do this it proposes prepending a fourth step onto the usual three for BWT compression. This fourth step is parsing, where the units to transform are determined.
- The paper looks at how to deal with non-character permutations at all three steps in the compression process.

Words as Tokens

- You could partition text into words and then permute the worlds instead of the characters.
- A benefit in compression of this is that the later characters in words are in effect assigned probabilities influences by the previous characters in a word. This means that the later characters are sometimes unnecessary.
- A drawback of this approach is that the tokens are no longer self-describing—that is a dictionary (whether explicit or implicit) has to be provided to look up what code relates to which word (having partitioning based on ASCII characters gets around this by using a standard dictionary: ASCII.)

Higher-Order Word-Based Models

- More complex word-based partitioning may also be considered.
- This, for example, may be useful because while the English word “the” may be the most common word in a text, but the sequence of words “the the” will usually be extremely uncommon.
- Also, some pairings of words will be more common than others, for example “block sorting” or “one ring.”

Storing the Dictionary

- Since partitioning into words requires the use of a dictionary of words, both encoder and decoder have to have access to said dictionary.
- One way to achieve this is to assume that there is a separate channel of communication between the encoder and decoder that can transmit the dictionary.
- The other way is to embed the dictionary in the symbol stream and through that, transmit it between encoder and decoder.

Dictionary Example

- Consider the following example on 2-grams:

String: spain.rain.mainly.plain.

Dictionary: sp, ai, n., ra, in, .m, nl, y., pl

Dictionary Sequence: spain.rain.mnly.pl

Encoded String: 0, 1, 2, 3, 4, 5, 1, 6, 7, 8, 1, 2

How Well Do Word-Based Models Perform?

- Variations used on Canterbury corpus' files
- Numbers are expressed in the unit “bits per character.”
- The actual compression showed little difference between individual character and words.

File	1-Gram	2-Gram	3-Gram	4-Gram	Words
asyoulik.txt	2.74	3.22	3.69	3.96	2.73
world192.txt	1.62	1.88	2.14	2.39	1.53
bible.txt	1.71	0.88	2.04	2.29	1.6
E.coli	1.99	1.98	1.98	1.99	2.05

Other Subsequent Alternations

- Assuming that an alternate (non-character) parsing method is introduced initially into BWT compression, it follows that it may be beneficial to question how this affects the other steps in the compression.
- These are the BWT algorithm, the MTF algorithm and the entropy encoder.
- Of these three, it is changes to the BWT algorithm that is the least likely to have a further impact, as it is the backbone of what we're working around anyway.

Entropy Encoding

- The earlier results were attained from using a large alphabet arithmetic encoder, treating all symbols uniformly.
- By allowing more recent tokens to exert a greater influence over the probabilities that earlier ones, we can predict the next token a greater portion of the time.
- By using something called a *structured knowledge* we can exploit this knowledge to get better compression.

Conclusion

- Breaking up a string by character and applying all the steps of BWT compression is now the only way to approach BWT compression
- Similar compression can be attained by breaking up the string into word tokens and applying the BWT compression.
- Then by tweaking the other elements of BWT compression (such as entropy encoding) we can get some improvement in the compression.