Making compression algorithms for Unicode text

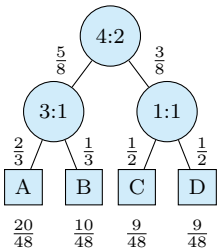
Adam Gleave
University of Cambridge
adam.gleave@cantab.net

Christian Steinruecken
University of Cambridge
tcs27@cam.ac.uk

Introduction. Traditional compressors operate on individual bytes. Often, this design choice makes them work poorly on texts in variable-length encodings such as UTF-8. We developed a method to modify single-byte compression algorithms so that they operate directly on characters. We used this method to create proof-of-concept variants of LZW and PPM that achieve substantially better compression effectiveness on a UTF-8 corpus than the unmodified algorithms. On ASCII and binary inputs, the modified and unmodified compressors perform comparably.

Method. To transform a byte-based compression algorithm, we propose the following two modifications. **(1)** Change the algorithm's alphabet from the *set of bytes* to a *set of tokens*. The token alphabet contains Unicode characters (used to represent UTF-8 inputs) and raw bytes (for error conditions and all other data). The input is mapped from bytes to tokens before compression, with the inverse operation taking place after decompression. **(2)** Change the algorithm's *base distribution* (typically a static uniform distribution over bytes) to an adaptive Pólya tree model over tokens.

Pólya tree model. Most texts are written in a single language, and use only symbols from a few small, contiguous regions of the Unicode code space. A Pólya tree model is well suited to learning distributions of this kind, as it tends to assign similar probabilities to nearby symbols. An example Pólya tree that defines a distribution over the four symbols $\{A, B, C, D\}$ is shown on the right.



The model consists of a balanced binary search tree, whose leaf nodes contain tokens. Each internal node (labelled L:R) maintains occurrence counts L and R for symbols in the left and right subtrees, which determine the branching probability (labelled on the edges). A token's probability is the product of the branching probabilities along the path from the root to the token. Learning a symbol therefore boosts the probability of its neighbours, as illustrated in the figure: '`B`' is assigned a higher probability than '`C`' and '`D`' (despite all three symbols occurring once each), because '`B`' is next to the frequently occurring symbol '`A`'.

Evaluation. We tested our compressors on the English-language Canterbury corpus and our own multilingual corpus of ten UTF-8 texts, with languages chosen based on their number of speakers. Operating over tokens substantially improves compression effectiveness on the UTF-8 corpus. Performance on the Canterbury corpus (ASCII and binary groups) is similar for the modified and unmodified compressors. Despite the radically different designs of LZW and PPM, both compressors enjoy an improvement in compression effectiveness from our method, suggesting this approach may be applicable to other compressors.

Group	LZW	LZW$^+$	PPM	PPM$^+$
UTF-8	2.839	2.492	1.980	1.860
ASCII	3.428	3.418	2.350	2.320
Binary	2.467	2.474	1.715	1.675
All	3.002	2.826	2.078	2.002

Mean compression effectiveness, in bits/byte. Column key: LZW$^+$ and PPM$^+$ are variants.

Full version at http://arxiv.org/abs/1701.04047