Investigation of the Error Performance of Tunstall Coding

Johann A. Briffa
University of Malta
Msida MSD 06, Malta
E-mail: jabrif@eng.um.edu.mt

Victor Buttigieg
University of Malta
Msida MSD 06, Malta
E-mail: vbutt@eng.um.edu.mt

Abstract — The error performance of the Tunstall code is analysed, and a relevant metric defined. Several parameters in the encoding process are considered, with the objective of minimising the effect of errors without sacrificing compression. The results obtained show that the error performance may be significantly improved without loss of compression.

I. INTRODUCTION

The Tunstall source coding algorithm [1] is a variable-to-fixed length encoding scheme, i.e. variable-length sequences of source symbols are mapped to fixed-length codewords. In general, over most practical channels, errors in the compressed stream are restricted to bit inversions only. Thus, since all codewords in a Tunstall code have a fixed length, an error in the compressed stream will only affect the codeword in which the error occurs; all subsequent codewords can be correctly decoded. This is in sharp contrast with fixed-to-variable length schemes, where the greatest adverse effect on error performance is due to loss of synchronisation and error propagation [2]. It is thus hypothesised that variable-to-fixed length schemes have a better performance in noise.

Furthermore, one or more errors in a codeword will cause this codeword to be interpreted as a different but valid one, since the Tunstall code is complete. This means that the sequence of source symbols represented by the original codeword will be replaced by the sequence corresponding to the incorrectly interpreted codeword. Now, the two sequences may not even be of the same length, leading to insertion, deletion as well as substitution errors in the decoded message. The Levenshtein distance [3], is used to calculate the number of symbol errors in the decoded message.

II. OPTIMISING THE ERROR PERFORMANCE

The Error Span ($E_s$) is defined as the average number of symbol errors in the decoded message for a random single bit error in the encoded message. Therefore, $E_s$ depends on the assignment of codewords to the sequences of source symbols. More importantly, this assignment will not affect the compression performance of the code. This feature has been exploited by seeking to assign codewords in such a way as to reduce $E_s$ (by designing a codebook such that random single-bit inversions in codewords result in another codeword mapping to a source symbol sequence very similar to the original one). Various code assignment algorithms have been investigated [4], representing different trade-offs between algorithm complexity and code performance.

To quantify the effect of $E_s$ on the code's performance in a noisy channel, the Symbol Error Rate (SER) of the decoded stream is plotted against the channel's Bit Error Rate (BER), as in Fig. 1. For comparison, the source was also encoded using a Huffman code. The performance of the randomly-assigned Tunstall code is marginally better than that of the Huffman code. This was expected because Tunstall coding does not suffer from loss of synchronisation of codewords. Furthermore, when the Tunstall code is optimised, its error performance approaches that of the uncompressed stream.

III. CONCLUSIONS

The algorithms which obtain the best $E_s$ tend to be very resource intensive. The fast algorithms, while performing significantly better than the random assignment, are much less effective in reducing $E_s$. In practice, the simple algorithms are the only ones that can be used in adaptive compression, and in any other case where the source statistics are not fixed. In certain cases, however, an approximation to the actual source statistics is known, and the complexity of the codeword assignment algorithm is of no consequence. This makes it feasible to seek the best error performance possible.

By increasing the costs of insertion and deletion errors, a code can be designed which trades off substitutional error performance for a better synchronisation. This may be useful in image compression applications, where insertion and deletion are typically more visible than substitution errors.

REFERENCES