International Journal of Innovative Computing, Information and Control
Volume **4**, Number **3**, March **2008**
 ISSN 1349-4198
pp. 761–770

ERROR RESILIENT LEMPEL-ZIV DATA COMPRESSION SCHEME WITH PERFECT HASHING

Chinchen Chang

Department of Information Engineering and Computer Science
Feng Chia University
No. 100 Wenhwa Rd., Seatwen, Taichung 40724, Taiwan
ccc@cs.ccu.edu.tw

Hsienwen Tseng

Department of Information Management
Chaoyang University of Technology
No. 168, Jifong E. Rd., Wufong Township, Taichung County 41349, Taiwan
hwtseng@cyut.edu.tw

Received January 2007; revised June 2007

Abstract. *Storer and Reif have studied the error resilient Lempel-Ziv code and shown that it gives protection against error propagation. They left an open problem in the error resilient Lempel-Ziv algorithm with k-error protocol. The algorithm uses hash function to eliminate the dependence between addresses in the dictionary. But hash conflicts affect the compression performance. In this paper, we have implemented a k-error protocol Lempel-Ziv algorithm with perfect hashing. By using a simple perfect hashing scheme, the hash conflicts can be easily removed and the compression performance can be improved. Experimental results also show that our algorithm can enhance the compression performance of the error resilient Lempel-Ziv code.*
Keywords: Lempel-Ziv code, Error resilient, k-error protocol, Perfect hashing

1. **Introduction.** The Lempel-Ziv (LZ) code is a dictionary-based data compression algorithm and is widely used in many entropy coders. In the late 1970s, Ziv and Lempel proposed the first LZ77 [1] and LZ78 [2] methods. Then LZW [3], a variant of LZ78, developed by Welch in 1984. The encoder/decoder maintain a dynamic dictionary when the data is encoded/decoded. When encoding, the LZ encoder breaks the input sequence into phrases of variable sizes while constructing a dictionary of phrases seen thus far. A new phrase is coded by the longest matching previously seen phrase plus the first character that didn't match. Finally, the encoder outputs a sequence of pointer pairs in the format (pointer, symbol). For example, an input sequence ABBABCABBBBBBC will be parsed as A, B, BA, BC, AB, BB, BBB, C and the output will be formatted as (0, A), (0, B), (2, A), (2, C), (1, B), (2, B), (6, B), (0, C). The corresponding dynamic dictionary is shown in Table 1. When decoding, the dynamic dictionary which keeps the decoder in synchronization with the encoder is constructed.

However, the LZ code is very vulnerable on an error-prone communication channel. Even a single error will cause synchronization loss at the decoder's dictionary, and possibly all decoded symbols after the error turn out faulty. This is called error propagation. For example, if we replace the second pair (0, B) with (1, B), the output sequence will be A<u>ABABAABCABABBA</u>BBBC (the incorrectly decoded symbols are underlined). Therefore, it is highly desirable to have the LZ code that can provide the capability of error