Pattern Matching in BWT-transformed Text

Don Adjeroh¹, Amar Mukherjee²,³, Tim Bell⁴,⁵, Matt Powell⁴, Nan Zhang²,³

The compressed pattern matching problem is to locate the occurrence(s) of a pattern \( P \) in a text string \( T \) using a compressed representation of \( T \), with minimal (or no) decompression. Algorithms are proposed that perform exact pattern matching in \( O(u + m \log(u/|\Sigma|)) \) time on average and in \( O(u + |\Sigma| \log |\Sigma| + (m^2/k) \log(u/|\Sigma| + \alpha k)) \) time for \( k \)-approximate matching on average (\( \alpha \leq u \)), where \( u \) is the size of the text, \( m \) is the size of the pattern, and \( \Sigma \) is the symbol alphabet.

The BWT performs a permutation of the characters in the text, such that characters in lexically similar contexts will be near to each other. The motivation for our approach is the observation that the BWT provides a lexicographic ordering of the input text as part of its inverse transformation process. The decoder only has limited information about the sorted context, but it may be possible to exploit this to perform an initial match on two symbols (a character and its context), and then decode only that part of the text to see if the pattern match continues. To search for potential matches, we consider only the bi-grams that are in the intersection of text bi-grams and pattern bi-grams. If the intersection is empty, it means that the pattern does not occur in the text; if not, we examine interaction set for higher \( q \)-grams, \( q > 2 \). We build a fast \( q \)-gram intersection algorithm by first computing the index arrays for the correspondence between the \( F \), the first column of the sorted cyclic matrix in BWT, and \( T \). Since \( F \) is sorted, we can then perform binary search on \( F \). There are two phases in approximate pattern matching approach: 1) locate areas in the text that contain potential matches by performing some filtering operations using \textit{appropriately sized} \( q \)-grams, 2) verify the results that are hypothesized by the filtering operations. The verification stage is generally slow, but usually, it will be performed on only a small proportion of the text. Thus, the overall performance depends critically on the number of hypothesis generated.

Preliminary results are given by comparing with that of \textit{agrep}, the standard pattern matching algorithm. Tests were performed using 19 selected files from the Canterbury and Calgary corpus. The results on exact pattern matching show that the number of comparisons of the proposed algorithms is much smaller than the \textit{agrep} with a factor of \( O(u/\log(u/|\Sigma|)) \). Although the running time is about 5 times longer, the main overhead is due to the BWT inverse transform and creating the index arrays. The amortized cost will be lower if multiple pattern matching is performed on the file. With 100 patterns picked randomly from Bible (4Mb), \textit{agrep} takes average 9.69 seconds for different sets, while our algorithm takes 17.63 seconds with an inverse BWT overhead of 6.85 seconds on Sun Ultra-5 workstation. There is still room to optimize the proposed algorithm.

1. Department of Computer Science & Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA; 2. Department of Computer Science, University of Central Florida, Orlando, FL 32816, USA; \{amar, nzhang\}@cs.ucf.edu 3. Partially supported by a grant from the National Science Foundation IIS-9977336.] 4. Department of Computer Science, University of Canterbury, New Zealand; \{tim, mjp86\}@cosc.canterbury.ac.nz 5. Supported by a grant from the Hitachinaka Techno Center Inc, Japan.