

Approximate Pattern Matching Using the Burrows-Wheeler Transform

Nan Zhang¹, Amar Mukherjee¹, Don Adjero², Tim Bell³

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. In this paper, we consider approximate pattern matching on the text transformed by the Burrows-Wheeler Transform (BWT). This is an important first step towards developing compressed pattern matching algorithm for BWT based compression system. Algorithms are proposed that solve the k -mismatch problem in $O(\min\{m(m-k)|\Sigma|^k \log(u/|\Sigma|), mu \log(u/|\Sigma|)\})$ time worst case and the k -approximate matching problem in $O(u + |\Sigma| \log \Sigma + (m^2/k) \log(u/|\Sigma| + \alpha k))$ on average ($\alpha \leq u$), where u is the size of the text, m is the size of the pattern, and Σ is the symbol alphabet.

The motivation for our approach is 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 fast q -gram (context) generation and matching algorithms have been developed to exploit this with the help of auxiliary index arrays built in linear time. The algorithm first computes the index arrays for the correspondence between F , the first column of the sorted cyclic matrix in BWT and T . All exact matches are grouped in consecutive rows of the sorted matrix, which makes the binary searches on F and/or q -grams of the matrix very efficient. For k -mismatch problem, we record and keep only those groups for which the number of errors is less than or equal to k . Solutions for $k=0$ correspond to exact matches. There are two phases in k -approximate pattern matching algorithm: 1) locate areas in the text that contain potential matches by performing some filtering operations using *appropriately sized* q -grams, 2) verify the results that are hypothesized by the filtering operations. The verification method can be based on either deterministic finite automata (DFA), or arithmetic operation based methods such as Shift-And implemented in *agrep* and *nrgrep*. 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.

Tests were performed on different pattern lengths using 133 selected files from the Canterbury, Calgary, and TREC corpus. The results on k -mismatch pattern matching show that the running time and storage are superior to the fast suffix tree approach. The k -approximate matching algorithms are compared with *agrep* and *nrgrep* on raw text. The running times of our algorithm are slower. The main overhead is due to the BWT inverse transform and creating the index arrays. The pure search time is faster than *agrep* and *nrgrep* by orders of magnitude. Thus, once the index arrays are created, for repeated pattern search operations and for long patterns, our algorithms perform significantly better than *agrep* and *nrgrep*. Using DFA verification, the search time is almost constant. The amortized cost is lower for multiple patterns search operations.

This research is partially supported by NSF grant number IIS-0207819.

1. Department of Computer Science, University of Central Florida, Orlando, FL.32816, USA; {amar,nzhang}@cs.ucf.edu 2. Department of Computer Science & Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA; don@csee.wvu.edu 3. Department of Computer Science, University of Canterbury, New Zealand; tim@cosc.canterbury.ac.nz .