A Fast Constant-Space Substring Search Algorithm

Harri Hakonen and Timo Raita

A family of fast constant-space substring search algorithms is described. The central idea behind the new scheme is a generalization of the well-known Boyer–Moore string searching approach complemented with a technique called q-slicing, a form of probabilistic q-gram matching. The search procedure is independent of the alphabet size and results in efficient and practical on-line implementations. Experiments show that the new variants are comparable to the fastest known Boyer–Moore methods.

Introduction. The substring searching problem is to find all occurrences of a pattern $pat[1..m]$ in a given text $text[1..n]$. The strings are concatenations of symbols taken from the input alphabet Σ of size σ. String searching has been studied extensively (see e.g. [1] for an excellent survey). As a result, several efficient and elegant solutions to this problem have been given. The most practical implementations have been derived from the seminal ideas of Boyer and Moore [4].

The original BM algorithm (BM for short) aligns the pattern with a text position, compares the corresponding symbols of pat and $text$ starting from the last symbol of the pattern and advancing to the left. If a mismatch (if any) is found, the pattern is shifted forward with respect to the text and the process is repeated. Prior to the actual search, two tables are formed in $O(m)$ time. These tables determine the length of the shift when a (mis)match is found. The *match heuristic* table determines how much the pattern must be moved in order to align an identical part of the pattern with the matched part of the text. The *occurrence heuristic* table expresses the position of the rightmost occurrence of each symbol of Σ in the pattern. Thus, the heuristic determines how much we can shift the pattern in order to align the mismatched text symbol with an identical pattern symbol.

The actual search procedure of BM consists of three distinct phases which are repeated until the text is exhausted: (i) fast skipping over non-matching text regions, (ii) match checking when some evidence of a pattern occurrence has been found and (iii) shift to the next position. All these steps have been the subject of refinements [5, 6, 8, 11, 13, 14]. A detailed analysis of various BM substep combinations can be found in [8]. In what follows, we will mainly concentrate on steps (i) and (iii), by giving an intuitive analysis on the length of the shift on the basis of the information used for those steps.

The well-known and widely used variant of Horspool [6] (BMH) discards the match heuristic due to its small practical significance with non-periodic patterns. This results in $O(mn)$ worst case complexity. However, on the average it is only $O(n/m)$ and therefore very fast. It is clear that on the average, BMH makes shorter shifts than the original BM due to the omission of the match heuristic. This behaviour is emphasized when σ is small. On the other hand, the role of the match heuristic becomes insignificant when σ becomes large (this is studied in more detail in [12]). As suggested in [11], we should try to compensate the omission of the match heuristic in other ways during the search. One alternative is to extend the occurrence heuristic for bigrams, incorporating thus both heuristics (at least partly) into one. This idea was introduced in [14] and was shown to give improved running times, especially for small input alphabets. Moreover, if we follow the idea of BMH and choose always (independently of the position where the mismatch occurred) the bigram composed of the text symbols aligning with $pat[m-1]$ and $pat[m]$, we obtain a very close approximation to the match heuristic. In the special case where the mismatch occurs at $pat[m-2]$, they are identical. Thus, if we can generalize the approach (as suggested in [2, 3, 10]) and use q-grams ($q > 2$) of arbitrary length, we obtain a heuristic which is a combination of both original ones. However, the disadvantage of the approach is that the preprocessing time and the space demand both increase rapidly, being proportional to σ^q.

The q-slicing method. In order avoid the excessive time and space requirements during preprocessing and still get large shifts, we combine two ideas. First, the information, on the basis of which the shift is made, is gathered from a large text region. For this reason, we call the symbol group thus obtained, a *generalized q-gram*. A q-gram is normally defined as a substring of q consecutive symbols; we relax this definition and require that it is a subsequence of length q. The symbols of the generalized q-gram are picked from the text during the search process. The positions of the symbols are defined in a *template* at the start of the preprocessing stage. What is significant in the definition of the template is, that we do not require the symbols to reside in the text 'under' the pattern.

Second, we reduce the size of the alphabet at the cost of losing some accuracy in symbol comparison. The idea is to partition the symbols of the input alphabet into equivalence classes and tag each class with a symbol of a reduced alphabet Σ'. Now, instead of comparing individual symbols, we compare tags of the classes. Currently, our implementations form the tag by taking the k least significant bits from the original symbol encoding; hence the name *q-slice*.

The search procedure starts by preprocessing the mapped pattern symbols as follows: choose a template T of length q. Gather all q-slices of pat defined by T and store each of them into a convenient machine-dependent unit (i.e. parameters k and q are chosen so that the q-slice fits into a byte or a word, for example). For each q-slice, compute a shift value, that is, the number of positions we can move the pattern with respect to the text without losing any matches. After the preprocessing, the search proceeds by mapping the text symbols to the reduced alphabet on-the-fly and using template T. Because the tag representation is short, we can efficiently use information which is scattered into a wide region in the neighbourhood of the current context. This gives a basis to increase the average length of a shift using only a small amount of comparisons. A drawback of the approach is, that at each q-slice match we must confirm that an identical symbol pattern has been found. In this respect, the tag sequence can be regarded as a special Rabin–Karp type signature [9]: the equality of two slices is a necessary, but not sufficient condition for the equality of the corresponding patterns. However, theoretical analysis shows that false matches occur rarely.

Summary. A new family of fast substring searching algorithms is devised by combining the concept of a q-gram with the relaxation of symbol equivalence and introducing templates. The strategy is called q-slicing and effectively, it makes sampling of the text on-line to skip fast over regions where the pattern $pat[1..m]$ cannot occur. In spite of the fact that most machine architectures do not support the idea of q-slicing on the hardware level, the efficiency of the new methods is comparable to the fastest known substring searching algorithms. The algorithms in this family are highly parametric and can thus be adapted according to special needs. Tests have shown that this approach results in a constant space algorithm which has an average shift length even larger than m, the size of the pattern itself.

References

[1] **Baeza-Yates, R.A.:** Algorithms for String Searching: A Survey, *SIGIR Forum, Spring/Summer 1989, Vol. 23, No. 3,4, pp. 34-58*

[2] **Baeza-Yates, R.A.:** Improved String Searching, *Softw. Pract. Exp., Vol. 19, No. 3, March 1989, pp. 257-271*

[3] **Baeza-Yates, R., Krogh, F.T., Ziegler, B., Sibbald, P.R. & Sunday, D.M.:** Notes on a Very Fast Substring Search Algorithm, *Comm. ACM, Vol. 35, No. 4, April 1992, pp. 132-137*

[4] **Boyer, R.S. & Moore, J.S.:** A Fast String Searching Algorithm, *Comm. ACM, Vol. 20, No. 10, October 1977, pp. 762-772*

[5] **Galil, Z.:** On Improving the Worst Case Running Time of the Boyer-Moore String Matching Algorithm, *Comm. of the ACM, Vol. 22, No. 9, September 1979, pp. 505-508*

[6] **Horspool, R.N.:** Practical Fast Searching in Strings, *Softw. Pract. Exp., Vol. 10, 1980, pp. 501-506*

[7] **Galil, Z. & Seiferas, J.:** Time-Space-Optimal String Matching, *J. Comput. System Sci., Vol. 26, 1983, pp. 280-294*

[8] **Hume, A. & Sunday, D.M.:** Fast String Searching, *Softw. Pract. Exp., Vol. 21, No. 11, November 1991, pp. 1221-1248*

[9] **Karp, R.M. & Rabin, M.O.:** Efficient Randomized Pattern-matching Algorithms, *IBM J. Res. Develop., Vol. 31, No. 2, March 1987, pp. 249-260*

[10] **Knuth, D.E., Morris, J.H. & Pratt, V.R.:** Fast Pattern Matching in Strings, *Siam J. Comput., Vol. 6, No. 2, June 1977, pp. 323-350*

[11] **Raita, T.:** Tuning the Boyer-Moore-Horspool String Searching Algorithm, *Softw. Pract. Exp., Vol. 22, No. 10, October, 1992, pp. 879-884*

[12] **Tarvainen, H.:** A Theoretical Framework for the Substring Searching Algorithms, *M.Sc. Thesis (in Finnish), University of Turku, Finland, May 1995*

[13] **Sunday, D.M.:** A Very Fast Substring Search Algorithm, *Comm. of the ACM, Vol. 33, No. 8, August 1990, pp. 132-142*

[14] **Zhu, R.F. & Takaoka, T.:** On Improving the Average Case of the Boyer-Moore String Matching Algorithm, *J. of Inf. Proc., Vol. 10, No. 3, 1987, pp. 173-177*