A Faster Algorithm for Approximate String Matching

Ricardo Baeza-Yates
Department of Computer Science, University of Chile

July 8, 1996

[summary by Mireille Régnier]

Approximate string matching is one of the main problems in classical string algorithms. Given a text of length n, a pattern of length m, and a maximal number of errors allowed, k, we want to find all text positions where the pattern matches the text up to k errors. Errors can be substituting, deleting or inserting a character. The solutions to this problem differ if the algorithm has to be on-line (that is, the text is not known in advance) or off-line (the text can be preprocessed). In this paper the first case is studied, where the classical dynamic programming solution is $O(mn)$.

In the last years several algorithms have been presented that achieve $O(kn)$ comparisons in the worst-case [13, 6, 7] or in the average case [14, 6], by taking advantage of the properties of the dynamic programming matrix. In the same trend is [3], with average complexity $O(kn/\sqrt{c})$ (c is the alphabet size). The algorithms which are $O(kn)$ in the worst case tend to involve too much overhead, and are not competitive in practice. Other approaches attempt to filter the text, reducing the area in which dynamic programming needs to be used [12, 15, 11, 10, 4, 5]. These algorithms achieve sublinear expected time in many cases ($O(kn\log_c m/m)$ is a typical figure) for moderate k/m ratios, but the filtration is not effective for larger ratios. A simple and fast filtering technique is shown in [2], which yields an $O(n)$ algorithm for moderate k/m ratios. Yet other approaches use bit-parallelism [1] in a RAM machine of word length $O(\log n)$ to reduce the number of operations. [9] achieves $O(kmn/\log n)$, which is competitive for patterns of length $O(\log n)$. In [16], the cells are packed differently to achieve $O(mn\log c/\log n)$ complexity.

A new algorithm is presented which combines the ideas of taking advantage of the properties of the matrix, filtering the text and using bit-parallelism, being faster than previous work for moderate size patterns, as we are interested in text searching. One models the search with a non-deterministic finite automaton (NFA) built from the pattern and using the text as input. This automaton is simulated by an algorithm based on bit operations on a RAM machine of word length $O(\log n)$. The algorithm achieves running time $O(n)$, independently of k, for small patterns (i.e. $mk = O(\log n)$). This restricted algorithm is used to design two general algorithms.

The first one partitions the problem into subproblems, and has average time cost $O(mn/\log n)$ for small $\alpha = k/m$ (i.e. $\alpha < 1/\log n$), otherwise it is $O(\sqrt{mk/\log n}\,n)$ (i.e. $O(\sqrt{k}n)$ for $m = O(\log n)$, else $O(kn)$). It involves also a cost to verify potential matches, which is shown to be not significant for $\alpha < \alpha_1 \approx 1 - m^{1/\sqrt{\log n}}/\sqrt{c}$. This algorithm is a generalization of an earlier heuristic [8, 2], that reduces the problem to exact matching and is shown to be $O(n)$ for $\alpha < \alpha_0 = 1/(3\log_c m)$, and better than problem partitioning for $\alpha < \alpha_0' \approx 1/(2\log_c m)$.

The second one partitions the automaton into subautomata, being $O(k^2n/(\sqrt{c}\log n))$ on average. For $\alpha > 1 - 1/\sqrt{c}$ its worst case, $O((m-k)kn/\log n)$, dominates. This algorithm is shown to be better than dynamic programming for $k > \log(n)/(1-\alpha)$. One studies the optimal way to combine

Condition	Complexity	Method used
$mk = O(\log n)$	$O(n)$	the simple algorithm
$\alpha < \alpha_0$	$O(n)$	reducing to exact match
$\alpha_0 < \alpha < \alpha_1$	$O(\sqrt{mk/\log n}n)$	exact match if $\alpha < \alpha_0'$ else problem partitioning
$\alpha > \alpha_1 \wedge k < \log n/(1-\alpha)$	$O((m-k)kn/\log n)$	automaton partitioning
$\alpha > \alpha_1 \wedge k > \log n/(1-\alpha)$	$O(mn)$	plain dynamic programming

Table 1. Complexity of the hybrid algorithm.

the algorithms. It is shown experimentally that the hybrid algorithm is faster than previous ones, for moderate m. Table 1 shows the complexity.

As a corollary of the analysis, tight bounds are given for the probability of finding an occurrence of a pattern of length m with k errors starting at a fixed position in random text. We also show that the heuristic of [14] works $O(kn)$ on average, with a constant tighter than that of [3].

Bibliography

[1] Baeza-Yates (R.). – Text retrieval: Theory and practice. In *12-th IFIP World Computer Congress*. vol. I:Algorithms, Software, Architecture. – Elsevier Science, 1992.

[2] Baeza-Yates (R.) and Perleberg (C.). – Fast and practical approximate pattern matching. In *CPM'92. Lecture Notes in Computer Science*, vol. 644, pp. 185–192. – Springer-Verlag, 1992.

[3] Chang (W.) and Lampe (J.). – Theoretical and empirical comparisons of approximate pattern matching. In *CPM'92. Lecture Notes in Computer Science*, vol. 644, pp. 172–181. – Springer-Verlag, 1992.

[4] Chang (W.) and Lawler (E.). – Sublinear approximate string matching and biological applications. *Algorithmica*, vol. 12, 1994, pp. 327–344.

[5] Chang (W.) and Marr (T.). – Approximate string matching and local similarities. In *CPM'94. Lecture Notes in Computer Science*, vol. 807, pp. 259–274. – Springer-Verlag, 1994.

[6] Galil (Z.) and Park (K.). – An improved algorithm for approximate string matching. *SIAM Journal on Computing*, vol. 19, n° 6, 1990, pp. 989–999.

[7] Landau (G.) and Vishkin (U.). – Fast string matching with k differences. *Journal of Computer Systems Science*, vol. 37, 1988, pp. 63–78.

[8] Manber (U.) and Wu (S.). – Agrep—a fast approximate pattern matching tool. In *Usenix Technical Conference*, pp. 153–152. – 1992.

[9] Manber (U.) and Wu (S.). – Fast text searching allowing errors. *CACM*, vol. 35, n° 10, 1992, pp. 83–91.

[10] Suntinen (E.) and Tarhio (J.). – On using q-gram locations in approximate string matching. In *ESA'95. Lecture Notes in Computer Science*, vol. 834, pp. 234–242. – Springer Verlag, 1995.

[11] Takaoka (T.). – Approximate pattern matching with samples. In *ISAAC'94. Lecture Notes in Computer Science*, vol. 834, pp. 348–359. – Springer Verlag, 1994.

[12] Tarhio (J.) and Ukkonen (E.). – Boyer-Moore approach to approximate string matching. In *SWAT'90. Lecture Notes in Computer Science*, vol. 447, pp. 348–359. – Springer Verlag, 1990.

[13] Ukkonen (E.). – Algorithms for approximate string matching. *Information and Control*, vol. 64, 1985, pp. 100–118.

[14] Ukkonen (E.). – Finding approximate patterns in strings. *Journal of Algorithms*, vol. 6, 1985, pp. 132–137.

[15] Ukkonen (E.). – Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, vol. 1, 1992, pp. 191–211.

[16] Wright (A.). – Approximate string matching using within-words parallelism. *Software-Practice and Experience*, vol. 24, 1994, pp. 337–362.