Increased Bit-Parallelism for Approximate String Matching

Heikki Hyyrö\textsuperscript{1,2*}, Kimmo Fredriksson\textsuperscript{3**}, and Gonzalo Navarro\textsuperscript{4***}

\textsuperscript{1} PRESTO, Japan Science and Technology Agency, Japan.
\textsuperscript{2} Department of Computer Sciences, University of Tampere, Finland.
\textsuperscript{3} Department of Computer Science, University of Joensuu, Finland.
\textsuperscript{4} Department of Computer Science, University of Chile, Chile.

Abstract. Bit-parallelism permits executing several operations simultaneously over a set of bits or numbers stored in a single computer word. This technique permits searching for the approximate occurrences of a pattern of length \(m\) in a text of length \(n\) in time \(O\left(\lceil \frac{m}{w} \rceil n\right)\), where \(w\) is the number of bits in the computer word. Although this is asymptotically the optimal speedup over the basic \(O(mn)\) time algorithm, it wastes bit-parallelism’s power in the common case where \(m\) is much smaller than \(w\), since \(w - m\) bits in the computer words get unused.

In this paper we explore different ways to increase the bit-parallelism when the search pattern is short. First, we show how multiple patterns can be packed in a single computer word so as to search for multiple patterns simultaneously. Instead of paying \(O(rn)\) time to search for \(r\) patterns of length \(m < w\), we obtain \(O(\lceil \frac{r}{\lfloor w/m \rfloor} \rceil n)\) time. Second, we show how the mechanism permits boosting the search for a single pattern of length \(m < w\), which can be searched for in time \(O(n/\lfloor w/m \rfloor)\) instead of \(O(n)\). Finally, we show how to extend these algorithms so that the time bounds essentially depend on \(k\) instead of \(m\), where \(k\) is the maximum number of differences permitted.

Our experimental results show that the algorithms work well in practice, and are the fastest alternatives for a wide range of search parameters.

1 Introduction

Approximate string matching is an old problem, with applications for example in spelling correction, bioinformatics and signal processing \[7\]. It refers in general to searching for substrings of a text that are within a predefined edit distance

\* Supported by the Academy of Finland and Tampere Graduate School in Information Science and Engineering.
\** Supported by the Academy of Finland.
\*** Supported in part by Fondecyt grant 1-020831.
threshold from a given pattern. Let \( T = T_1 \ldots n \) be a text of length \( n \) and \( P = P_1 \ldots m \) a pattern of length \( m \). Here \( A_{a \ldots b} \) denotes the substring of \( A \) that begins at its \( a \)th character and ends at its \( b \)th character, for \( a \leq b \). Let \( ed(A, B) \) denote the edit distance between the strings \( A \) and \( B \), and \( k \) be the maximum allowed distance. Then the task of approximate string matching is to find all text indices \( j \) for which \( ed(P, T_h \ldots j) \leq k \) for some \( h \leq j \).

The most common form of edit distance is Levenshtein distance \([5]\). It is defined as the minimum number of single-character insertions, deletions and substitutions needed in order to make \( A \) and \( B \) equal. In this paper \( ed(A, B) \) will denote Levenshtein distance. We also use \( w \) to denote the computer word size in bits, \( \sigma \) to denote the size of the alphabet \( \Sigma \) and \( |A| \) to denote the length of the string \( A \).

Bit-parallelism is the technique of packing several values in a single computer word and updating them all in a single operation. This technique has yielded the fastest approximate string matching algorithms if we exclude filtration algorithms (which need anyway to be coupled with a non-filtration one). In particular, the \( O(\lceil m/w \rceil kn) \) algorithm of Wu and Manber \([13]\), the \( O(\lceil km/w \rceil n) \) algorithm of Baeza-Yates and Navarro \([1]\), and the \( O(\lceil m/w \rceil n) \) algorithm of Myers \([6]\) dominate for almost every value of \( m, k \) and \( \sigma \).

In complexity terms, Myers’ algorithm is superior to the others. In practice, however, Wu & Manber’s algorithm is faster for \( k = 1 \) and Baeza-Yates and Navarro’s is faster when \((k + 2)(m - k) \leq w \) or \( k/m \) is low. The reason is that, despite that Myers’ algorithm packs better the state of the search (needing to update less computer words), it needs slightly more operations than its competitors. Except when \( m \) and \( k \) are small, the need to update less computer words makes Myers’ algorithm faster than the others. However, when \( m \) is much smaller than \( w \), Myers’ advantage disappears because all the three algorithms need to update just one (or very few) computer words. In this case, Myers’ representation wastes many bits of the computer word and is unable to take advantage of its more compact representation.

The case where \( m \) is much smaller than \( w \) is very common in several applications. Typically \( w \) is 32 or 64 in a modern computer, and for example the Pentium 4 processor allows one to use even words of size 128. Myers’ representation uses \( m \) bits out of those \( w \). In spelling, for example, it is usual to search for words, whose average length is 6. In computational biology one can search for short DNA or amino acid sequences, of length as small as 4. In signal processing applications one can search for sequences composed of a few audio, MIDI or video samples.

In this paper we concentrate on reducing the number of wasted bits in Myers’ algorithm, so as to take advantage of its better packing of the search state even when \( m \leq w \). This has been attempted previously \([2]\), where \( O(m[n/w]) \) time was obtained. Our technique is different. We first show how to search for several patterns simultaneously by packing them all in the same computer word. We can search for \( r \) patterns of length \( m \leq w \) in \( O(\lceil r/[m/w] \rceil n + occ) \) rather than \( O(rn) \) time, where \( occ \leq rn \) is the total number of occurrences of all the patterns. We