Approximate String Matching Based on Bit Operations

E, Hanmei
Faculty of Information Science and Electrical Engineering, Kyushu University

Yu, Yunqing
Faculty of Information Science and Electrical Engineering, Kyushu University

Baba, Kensuke
Faculty of Information Science and Electrical Engineering, Kyushu University

Murakami, Kazuaki
Faculty of Information Science and Electrical Engineering, Kyushu University

https://hdl.handle.net/2324/15561

出版情報：Lecture Series on Computer and Computational Sciences. 7 (A), pp.195-198, 2006-10. Brill
バージョン：
権利関係：

Approximate String Matching Based on Bit Operations[*]

Hanmei E   Yunqing Yu   Kensuke Baba[†]   Kazuaki Murakami

Abstract

The bit-parallelism is a speedup method for solving problems of string matching. The speedup by the bit-parallelism depends on the performance of a computer and it is very significant in practice. In terms of time complexity based on a standard computational model, however, the performance cannot be represented explicitly. This paper introduces a parameter in a computational model to measure the performance of a computer, and explicitly analyses the time complexity of a bit-parallel algorithm for the match-count problem. The implementation of the algorithm and some test calculations are presented.

Keywords: Approximate string matching, bit-parallelism, match-count problem, score vector.

1 Introduction

The problem of string matching [3, 4] is to find all occurrences of a string (called a "pattern") in another string (called a "text"). The approximate string matching is defined as the string matching with some errors allowed. The approximate string matching is more useful in a wide area of applications, and its most general form (e.g., the problem of weighted edit distance [9] and its extension [8]) is the essence of some interesting systems [7] for homology search in biology.

One of the most active areas for the approximate string matching is bit-parallelism [6]. The main idea of this approach is to represent strings as numbers (or bit sequences) and perform plural comparisons of characters simultaneously by arithmetic (or bit operations). Therefore, a practical run-time depends on the performance of a computer, and this idea can be found essentially in the Rabin-Karp algorithm [2]. As for the approximate string matching, we consider the match-count problem [5] in this paper. For this problem, a simple and efficient method based on bit-parallelism is introduced by Baeza-Yate and Gonnet [1], and it is called the "Shift-Add" method. While a naive algorithm (based on character comparison) requires $O(mn)$ comparisons for input strings of lengths m and n, an algorithm based on the Shift-Add method requires $O(mn \log m/w)$ bit operations, for the word size w of a computer.

The speedup by bit-parallelism is usually significant in practice, however, in a strict sense, it cannot be represented explicitly in terms of time complexity based on a standard computational model. In order to solve this problem, in this paper, we introduce a parameter for a computational model to measure the performance of a computer. We define a set of bit operations running in a unit time with a parameter, which restricts the lengths of strings for the operations. Then, we analyze an algorithm based on the Shift-Add method in terms of our computational model, and modify the algorithm in two aspects. One modification is that we convert each character in a given string into a single bit character rather than a bit sequence. By this modification, a straightforward parallelism can be applied to the Shift-Add algorithm. The other modification is that we prepare a table for the computation of the Hamming distance of two bit-sequences. The algorithms are implemented and some preliminary tests are presented.

2 Implementation of Algorithm

We modify the Shift-Add algorithm [1] as mentioned in the previous subsection. For the modification, we prepare a table for the computation of the Hamming distance of two bit sequences.

We first convert input strings $t = t_1 t_2 \cdots t_n$ and $p = p_1 p_2 \cdots p_m$ on an alphabet Σ respectively into the bit-sequences $T \in \{0,1\}^n$ and $P \in \{0,1\}^m$ by the functions f_u for $u \in \Sigma$ such that $f_u(v)$ is 1 if $u = v$, and 0 otherwise. After the conversion, the ith element c_i of the score vector is obtained by

$$c_i = \sum_{u \in \Sigma} \sum_{j=1}^{m} f_u(t_{i+j-1}) \cdot f_u(p_j).$$

The outline of Shift-Add algorithm is shown in Figure 1, where $x \ll b$, $x \gg b$, and $x \wedge y$ for $x, y \in \{0,1\}^*$ are b shift-left operations to x, b shift-right left operations to x, and an and operation to x and y. The main idea of our modified algorithm is to apply the

[*] An edited version of this report was published in: *Selected Papers from the International Conference of Computational Methods in Sciences and Engineering 2006 (ICCMSE 2006), Lecture Series on Computer and Computational Sciences, vol.7(A), pp.195–198, VSP/Brill, Oct, 2006.*

[†] Faculty of Information Science and Electrical Engineering, Kyushu University, *baba@i.kyushu-u.ac.jp*

```
**Procedure** Shift-Add
Input: $t = t_1 t_2 \cdots t_n$, $p = p_1 p_2 \cdots p_m$
Output: $C(t,p) = (c_1, c_2, \ldots, c_{n-m+1})$

$b := \lceil \log(m+1) \rceil$ ;

for $1 \le i \le n$ do $E[t_i] := 0$ ;
for $1 \le i \le m$ do {
    $E[p_i] := 0$ ;
    for $1 \le j \le m$ do
        $E[p_i] := (E[p_i] \ll b) + f_{p_j}(p_i)$ ;
}

$D := 0$ ;
for $1 \le i \le n$ do {
    $D := (D \gg b) + E[t_i]$ ;
    $c_{i-m+1} := D \wedge 1^b$ ;
}
```

Figure 1: The Shift-Add algorithm for the match-count problem, where 1^b is the string constructed by b unities.

```
**Procedure** Parallel Shift-Add
Input: $t = t_1 t_2 \cdots t_n$, $p = p_1 p_2 \cdots p_m$
Output: $C(t,p) = (c_1, c_2, \ldots, c_{n-m+1})$

for $1 \le i \le n-m+1$ do $c_i := 0$ ;
for $a \in \Sigma$ do {
    $ft := 0$, $fp := 0$ ;
    for $1 \le i \le m$ do {
        $ft := (ft \ll 1) + f_a(t_i))$ ;
        $fp := (fp \ll 1) + f_a(p_i))$ ;
    }

    for $1 \le i \le n-m+1$ do {
        $c_i := c_i + Match(ft, fp)$ ;
        $ft := S_l(ft) + f_a(t_{m+i})$ ;
    }
}
```

Figure 2: The Parallel Shift-Add algorithm for the match-count problem.

speedup by the parallelism with respect to the word size w of a computer into the computation of the converted strings rather than the given strings. Moreover, we consider the function $Match$ from $\Sigma_m \times \Sigma_m$ to $\{0, 1, \ldots, m\}$ such that, for $u, v \in \Sigma^m$, $Match(u, v)$ is the number of 1 in $u \wedge v$. This function cannot be computed in a unit time by a straightforward combination of some of the operations in the computational model.

The outline of the modified algorithm is shown in Figure 2. The computation of the initial conversion from strings to bit sequences needs $O(\sigma(n+m))$ comparisons of characters, where σ is the cardinality of Σ. The computation of the score vector takes $O(mn)$ time and w elements can be computed simultaneously if a suitable table of size $O(2^m)$ is prepared. In a same way as the Shift-Add algorithm, we assume that the score vector is obtained by a straightforward iteration. Therefore, the time complexity is

$$(O(\sigma(n+m)) + O(\sigma mn))/w = O(\sigma mn/w).$$

3 Experimental Results

We have worked out a new algorithm called Parallel Shift-Add. It is a modification of the algorithm Shift-Add with bit-parallelism. We have implemented these three algorithms: Comparison (the naive comparison-based algorithm), Shift-Add, and Parallel Shift-Add and tested for a text of length $n = 100$, $1,000$, and $10,000$ and a pattern of length $m = 4$, 8, 32, and 64 over $\Sigma = \{0,1\}^8$. The word size of the computer which we used is 32. In the algorithm Parallel Shift-Add, the size of the table for the function $Match$ is $2^8 = 256$. The results are shown in Table 3, where each value is the average of $1,000$ times iteration.

4 Discussions

We introduce a parameter in a computational model in order to measure the performance of a computer. By the analysis of the algorithm based on the Shift-Add method, we extend the algorithm to be adaptable to a simple parallelism one. Moreover, we implemented the three algorithms and presented the results.

Although the estimation of the computational time of our modified algorithm should be less than that of the simple Shift-Add algorithm, the practical run time is not so as shown in Table 3. This implies that the word size w of the computer should be more than about 100 to receive a benefit of our algorithm. The main reason for the unexpected result of our program is the fact that some descriptions of the bit operations in C-language do not perform as efficiently as we expected. The other reason is that our program is not well optimized. As the performance also depends on the computer processor, hence to refine the computational model by using other computer processors is our future work.

References

[1] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, 1992.

n	100				1,000				10,000			
m	4	8	32	64	4	8	32	64	4	8	32	64
Comparison	0.02	0.04	0.13	0.24	0.07	0.13	0.49	0.96	0.67	1.31	4.87	9.59
Shift-Add	0.02	0.03	0.12	0.23	0.03	0.04	0.18	0.35	0.18	0.20	0.72	1.44
Parallel Shift-Add	0.52	0.52	2.06	4.13	4.44	4.50	18.11	35.54	-	-	-	-

Table 1: CPU time comparison between different string matching algorithms.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, Second Edition.* MIT Press, 2001.

[3] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[4] M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific, 2003.

[5] D. Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[6] G. Navarro. A guided tuor to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.

[7] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proc. Natl. Acad. Sci. USA*, volume 85, pages 2444–2448, 1988.

[8] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[9] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.