Optimal Neural Network Algorithm for On-Line String Matching

Yiu-Wing Leung, Jiang-She Zhang, and Zong-Ben Xu

***Abstract*—We consider an on-line string matching problem in which we find all the occurrences of a pattern of m characters in a text of n characters, where all the characters of the pattern are available before processing while the characters of the text are input one after the other. We propose a space–time optimal parallel algorithm for this problem using a neural network approach. This algorithm uses m McCulloch–Pitts neurons connected as a linear array. It processes every input character of the text in one step and hence it requires at most n iteration steps.**

***Index Terms*—Algorithms, neural network, string matching.**

I. Introduction

The *string matching problem* is to find the first occurrence (or all occurrences) of a given *pattern* of length m in a given *text* of length n, where both the pattern and the text are available before processing. This problem arises in a wide variety of applications and many algorithms have been proposed for solving it. An excellent survey of these algorithms was reported in [1]. In particular, the Boyer–Moore algorithm [2] and the Knuth–Morris–Pratt algorithm [3] are the most well-known. Both of them require $O(m+n)$ time in the worst case and require $O(m)$ space. Several parallel algorithms have also been proposed for different types of parallel computers [4]–[11]. In particular, the Takefuji–Tanaka–Lee algorithm [4] is the fastest and it requires only two iteration steps using $n(n-m+1)$ simple processing units.

Takaoka [12] studied an on-line string matching problem in which all the n characters of the text are available before processing but the m characters of the pattern are input one after the other. For example, if a user is editing a file using an on-line editor and he wants to search a particular string pattern, he enters the characters of the pattern one after the other. Takaoka proposed an algorithm for this problem and it runs in $O(I+n)$ time where I is the time for inputting the pattern [12].

In this paper, we consider another on-line string matching problem in which the m characters of the pattern are available before processing but the n characters of the text are input one after the other. This problem arises in many practical scenarios where text retrieval is slower than processing. Two examples are given as follows: 1) if the text is stored in a CD-ROM, then the characters of the text are read and passed to the processor one after the other; 2) if a client wants to search a pattern in a text stored in a remote file server, the client receives the characters of the text one after the other through a computer network. We propose a space-time optimal parallel algorithm for this on-line string matching problem using a neural network approach. This algorithm uses m McCulloch–Pitts neurons connected as a linear array. It processes every input character of the text in one step and hence it requires at most n iteration steps.

Manuscript received December 15, 1995; revised May 25, 1997 and July 5, 1997.

Y.-W. Leung is with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong.

J.-S. Zhang and Z.-B. Xu are with the Research Center for Applied Mathematics, Institute for Information and Systems Sciences, College of Science, Xi'an Jiaotong University, 710049, Xi'an, China.

Publisher Item Identifier S 1083-4419(98)07308-7.

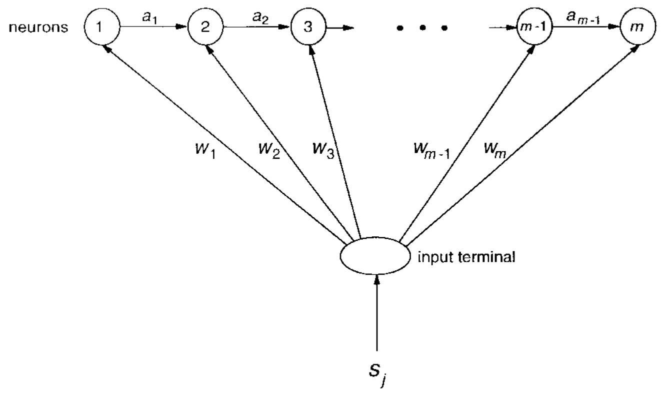

Fig. 1. The neural network for on-line string matching.

II. Neural Network Architecture and Algorithm

We let $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_q\}$ be a finite set of characters with size q. We assign a distinct integer value to every character in Σ such that $\sigma_i = 2i+1$. We denote the pattern by $p = p_1 p_2 \cdots p_m$ and denote the text by $s = s_1 s_2 \cdots s_n$, where $p_i \in \Sigma$ and $s_j \in \Sigma$ for $1 \le i \le m$ and $1 \le j \le n$. The pattern p is given before processing while the characters of the text s are input one after the other such that s_i is input before s_{i+1}.

The neural network consists of an input terminal and m neurons. The input terminal is connected to every neuron and the neurons are linearly interconnected (see Fig. 1). The text enters the network via the input terminal. The input terminal is connected to the ith neuron with synaptic weight w_i for all $1 \le i \le m$. The characters of the pattern are stored in these synaptic weights such that

$$w_i = \begin{cases} \dfrac{2}{p_1}, & \text{if } i = 1 \\ \dfrac{1}{p_i}, & \text{if } i > 1 \end{cases} \tag{1}$$

where we remind that the character p_i has been assigned an integer value. Based on (1), the synaptic weights can be changed for different patterns. The ith neuron is connected to the $(i+1)$th neuron with weight $a_i = 1$. Every neuron is a McCulloch–Pitts binary neuron [13] with threshold $\theta_i = 2$ for $i > 1$ and $\theta_1 = 1$, and activation function

$$g(x) = \begin{cases} 1, & \text{if } |x| < \dfrac{1}{(2q+1)} \\ 0, & \text{otherwise} \end{cases}$$

where q is the size of Σ. In other words, the state of every neuron is either 1 or 0. We denote the state of the ith neuron by O_i. If $O_i = 1$, we say that the ith neuron is fired; otherwise, it is inhibited.

The characters of the text enter the neural network one by one and we say that s_j enters at time j. After processing s_j, the state of the ith neuron is denoted by $O_i(j)$. When the character s_{j+1} enters at time $j+1$, the state of the ith neuron will be updated according to the following rule:

$$O_i(j+1) = g(s_{j+1} w_i + O_{i-1}(j) - \theta_i).$$

The details of the algorithm based on this neural network approach are given as follows.

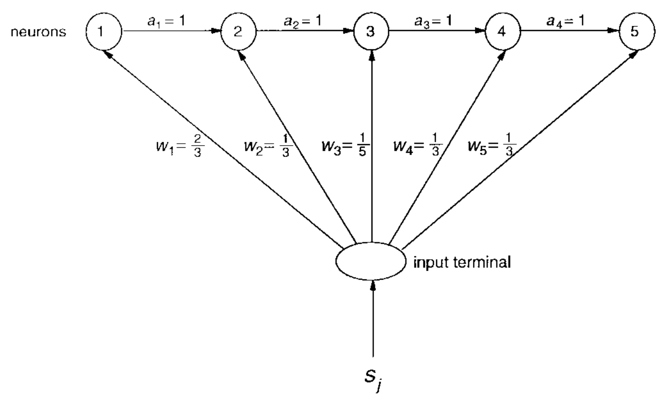

Fig. 2. The neural network used in Example 1.

TABLE II
THE STATES OF THE NEURONS IN EXAMPLE 2

Input Character	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9	O_{10}
	0	0	0	0	0	0	0	0	0	0
$s_1 = b$	0	0	0	0	0	0	0	0	0	0
$s_2 = a$	1	0	0	0	0	0	0	0	0	0
$s_3 = b$	0	1	0	0	0	0	0	0	0	0
$s_4 = c$	0	0	1	0	0	0	0	0	0	0
$s_5 = b$	0	0	0	0	0	0	0	0	0	0
$s_6 = a$	1	0	0	0	0	0	0	0	0	0
$s_7 = b$	0	1	0	0	0	0	0	0	0	0
$s_8 = c$	0	0	1	0	0	0	0	0	0	0
$s_9 = a$	1	0	0	1	0	0	0	0	0	0
$s_{10} = b$	0	1	0	0	1	0	0	0	0	0
$s_{11} = c$	0	0	1	0	0	1	0	0	0	0
$s_{12} = a$	1	0	0	1	0	0	1	0	0	0
$s_{13} = a$	1	0	0	0	0	0	0	0	0	0
$s_{14} = b$	0	1	0	0	0	0	0	0	0	0
$s_{15} = c$	0	0	1	0	0	0	0	0	0	0
$s_{16} = a$	1	0	0	1	0	0	0	0	0	0
$s_{17} = b$	0	1	0	0	1	0	0	0	0	0
$s_{18} = c$	0	0	1	0	0	1	0	0	0	0
$s_{19} = a$	1	0	0	1	0	0	1	0	0	0
$s_{20} = b$	0	1	0	0	1	0	0	0	0	0
$s_{21} = c$	0	0	1	0	0	1	0	0	0	0
$s_{22} = a$	1	0	0	1	0	0	1	0	0	0
$s_{23} = c$	0	0	0	0	0	0	0	1	0	0
$s_{24} = a$	1	0	0	0	0	0	0	0	1	0
$s_{25} = b$	0	1	0	0	0	0	0	0	0	1
$s_{26} = c$	0	0	1	0	0	0	0	0	0	0

ON-LINE ALGORITHM:

1) Given the pattern p, assign values to the synaptic weights w_i $(1 \le i \le m)$ based on (1). The initial states of all the neurons are set to zero, i.e., $O_i(0) = 0$ for all $1 \le i \le m$. Let $j = 1$.

2) The input terminal reads the character s_j of the input text.

3) Every neuron updates its state according to

$$O_i(j) = \begin{cases} g(s_j w_i + O_{i-1}(j-1) - \theta_i), & \text{if } 1 < i \le m \\ g(s_j w_1 - \theta_1), & \text{if } i = 1. \end{cases} \tag{2}$$

4) If the last neuron is fired (i.e., $O_m(j) = 1$), then a match is found. If the problem is to find the first occurrence of the pattern, the algorithm is terminated here. Otherwise, go to step 5).

5) $j := j + 1$. If $j > n$, terminate the algorithm; otherwise go to step 2).

Once all the n input characters have entered the neural network, the algorithm terminates since $j = n + 1 > n$. Therefore, the proposed algorithm requires at most n iteration steps.

In the following theorem, we prove that if the ith neuron is fired, this means that the first i characters of the pattern match with the latest i characters of the incoming text. As a result, the proposed algorithm can always find all the occurrences of the pattern in the input text.

Theorem: If and only if $O_i(j) = 1$ in the algorithm, then

$$p_1 p_2 \cdots p_i = s_{j-i+1} s_{j-i+2} \cdots s_j. \tag{3}$$

Therefore, if $O_m(j) = 1$, then $s_{j-m+1} s_{j-m+2} \cdots s_j = p$ and thus a match is found in the text.

Proof: We apply mathematical induction. For $j = 1$, according to (2), only $O_1(1)$ may be equal to 1 since $O_i(0) = 0$ for $1 \le i \le m$ and $s/p_1 \ne 2$ for any $s \in \Sigma$. If $s_1 \ne p_1$, then $s_1 w_1 = 2s_1/p_1 \ne 2$ and hence we have

$$|s_1 w_1 - 2| = \frac{|2s_1 - 2p_1|}{p_1} \ge \frac{2}{p_1} \ge \frac{1}{(2q+1)}.$$

By (2), this implies $O_1(1) = 0$. If $s_1 = p_1$, then $s_1 w_1 = 2$ and we have $O_i(1) = 1$. Therefore, the theorem holds for $j = 1$.

Suppose the theorem holds for $j = r$. If $O_i(r+1) = 1$, there are three cases: 1) $O_{i-1}(r) = 0$, 2) $O_{i-1}(r) = 1$ and $s_{r+1} \ne p_i$, and 3) $O_{i-1}(r) = 1$ and $s_{r+1} = p_i$. The first case is impossible because

$$|\frac{s_{r+1}}{p_i} - 2| = |\frac{s_{r+1} - 2p_i}{p_i}| \ge \frac{1}{p_i} \ge \frac{1}{2q+1}$$

which contradicts the fact that $O_i(r+1) = 1$. The second case is also impossible because

$$|\frac{s_{r+1}}{p_i} - 1| = |\frac{s_{r+1} - p_i}{p_i}| \ge \frac{1}{p_i} \ge \frac{1}{2q+1}$$

which also contradicts the fact that $O_i(r+1) = 1$. Therefore, if $O_i(r+1) = 1$, then $O_{i-1}(r) = 1$ and $s_{r+1} = p_i$. By the induction hypothesis that the theorem holds for $j = r$, $O_{i-1}(r) = 1$ implies

$$p_1 p_2 \cdots p_{i-1} = s_{(r+1)-i+1} s_{(r+1)-i+2} \cdots s_r.$$

Thus $O_i(r+1) = 1$ implies

$$p_1 p_2 \cdots p_i = s_{(r+1)-i+1} s_{(r+1)-i+2} \cdots s_{r+1}. \tag{4}$$

If $O_i(r+1) = 0$, then either $O_{i-1}(r) = 0$ or $s_{r+1} \ne p_i$, or both. In any of these cases, (4) does not hold. Thus, the theorem holds for $j = r + 1$. By the principle of mathematical induction, the theorem holds for any j. Q.E.D.

The proposed on-line algorithm is time-optimal because we only need one iteration step to process any input character. It is space-optimal because we only need to store m characters of the pattern and the states of m neurons.

III. EXAMPLES

In this section, we give two examples to illustrate the execution of the proposed algorithm.

Example 1: Consider the pattern $p = aabaa$, text $s = ababaabaabaab$ and $\Sigma = \{a, b\}$. The characters a and b are assigned the integer values 3 and 5, respectively. The neural network for the pattern p is shown in Fig. 2. The characters of the text are input one by one, and the state of the ith neuron at time j is given in the row corresponding to s_j in Table I. From Table I, we see that $O_5(9) = O_5(12) = 1$ and hence the algorithm finds two matches in

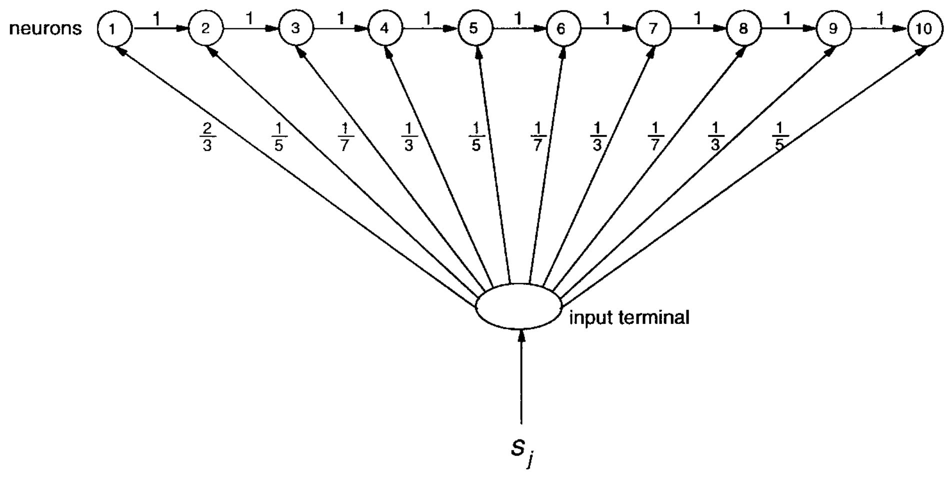

Fig. 3. The neural network used in Example 2.

the text at s_9 and s_{12} respectively, as shown below

Text	a	b	a	b	a	a	b	a	a	b	a	a	b
Pattern					a	a	b	a	a				
								a	a	b	a	a	

Example 2: Consider the pattern $p = abcabcacab$, text $s = babcbabcabcaabcabcabcacabc$ and $\Sigma = \{a, b, c\}$. The characters a, b and c are assigned the integer values 3, 5 and 7, respectively. The neural network for the pattern p is shown in Fig. 3. As the characters of the text are being input, the state of the ith neuron at time j is given in the row corresponding to s_j in Table II. From Table II, we see that $O_{10}(25) = 1$ and hence the proposed algorithm finds a match in the text at s_{25}, as shown below:

Text	$babcbabcabcaabc$	$abcabcacab$	c
Pattern		$abcabcacab$	

Before we close this section, we would like to point out an interesting remark. From Tables I and II, we can see that several neurons may be fired at the same time. This means that several suffixes $s_1 s_2 \cdots s_j$ are the prefixes of pattern p. For the example shown in Table II, when $j = 19$, $O_1(19) = O_4(19) = O_7(19) = 1$. It is easy to check that $s_{19} = a$, $s_{16}s_{17}s_{18}s_{19} = abca$, $s_{13}s_{14}s_{15}s_{16}s_{17}s_{18}s_{19} = abcabca$ are prefixes of $p = abcabcacab$, as shown below:

Pattern	a	b	c	a	b	c	a	c	a	b
$\mathbf{s_{19}}$	a									
$\mathbf{s_{16}s_{17}s_{18}s_{19}}$	a	b	c	a						
$\mathbf{s_{13}s_{14}s_{15}s_{16}s_{17}s_{18}s_{19}}$	a	b	c	a	b	c	a			

IV. Conclusions

In this paper, we proposed a space–time optimal parallel algorithm for an on-line string matching problem in which the m characters of the pattern are given and the n characters of the text are input one after the other. This algorithm requires m neurons connected as a linear array and hence it can easily be realized. In addition, it processes every input character of the text in exactly one step and hence it requires at most n iteration steps.

References

[1] A. V. Aho, “Algorithms for finding patterns in strings,” in *Handbook of Theoretical Computer Science*. Amsterdam, The Netherlands: Elsevier, 1990, ch. 5.

[2] R. S. Boyer and J. S. Moore, “A fast string searching algorithm,” *Commun. ACM,* vol. 20, pp. 762–772, 1977.

[3] D. E. Knuth, J. H. Morris, and V. R. Pratt, “Fast pattern matching in strings,” *SIAM J. Comput.,* vol. 6, pp. 323–350, 1977.

[4] Y. Takefuji, T. Tanaka, and K. C. Lee, “A parallel string search algorithm,” *IEEE Trans. Syst., Man, Cybern.,* vol. 22, pp. 332–336, 1992.

[5] G. H. Chen, “An $O(1)$ time algorithm for string matching,” *Int. J. Comput. Math.,* vol. 42, pp. 185–191, 1992.

[6] O. Berkman, D. Breslauer, G. Galil, B. Schieber, and U. Vishkin, “Highly parallelizable problems,” in *Proc. 21st ACM Symp. Theory Computing,* 1989.

[7] Z. Galil, “Optimal parallel algorithm for string matching,” in *Proc. 16th ACM Symp. Theory Computing,* 1984, pp. 240–248.

[8] ——, “A constant time optimal parallel string matching algorithm,” in *Proc. 23rd ACM Symp. Theory Computing,* 1992, pp. 69–76.

[9] T. Goldberg and U. Zwick, “Faster parallel string matching via larger deterministic samples,” *J. Algorithms,* vol. 16, pp. 295–308, 1994.

[10] U. Vishkin, “Optimal parallel pattern matching in strings,” in *Proc. 12th ICALP, Lecture Notes Computer Science,* 1985, vol. 194, pp. 497–507.

[11] ——, “Deterministic sampling—A new technique for fast pattern matching,” *SIAM J. Comput.,* vol. 20, pp. 22–40, 1990.

[12] T. Takaoka, “An on-line pattern matching algorithm,” *Inf. Process. Lett.,* vol. 22, pp. 329–330, 1986.

[13] W. S. McCulloch and W. H. Pitts, “A logical calculus of ideas immanent in nervous activity,” *Bull. Math. Biophys.,* vol. 5, pp. 115–133, 1943.