ELSEVIER

Neurocomputing 14 (1997) 289–299

NEUROCOMPUTING

A low-cost neural sorting network with $O(1)$ time complexity [1]

Shun-Shii Lin [*], Shen-Hsuan Hsu

Department of Information and Computer Education, National Taiwan Normal University, Taipei, Taiwan, ROC

Received 14 July 1995; accepted 31 March 1996

Abstract

In this paper, we present an $O(1)$ time neural network with $O(n^{1+\varepsilon})$ neurons and links to sort n data, $\varepsilon > 0$. For large-size problems, it is desirable to have low-cost hardware solutions. In order to solve the sorting problem in constant time and with less hardware-cost, we adopt Leighton's column sort [5] as the main architecture. Then we use Chen and Hsieh's neural network [3] with $O(n^3)$ complexity as the lowest-level sub-networks. By using recursive techniques properly, we are able to explore constant-time, low-complexity neural sorting networks.

Keywords: Complexity; Neural network; Parallel processing; Sorting

1. Introduction

The primary objectives of parallel sorting algorithm are to minimize both sorting time and processors while arranging given items in a desired order. A neural network is a suitable architecture in some parallel algorithms because it can process the data simultaneously [4].

The problem of sorting n numbers with a fixed connected network has a long and rich history [1]. There are a number of neural networks proposed for related purposes. Chen and Hsieh [3] have constructed a neural sorting network with $O(1)$ time complexity. Their network needs $O(n^3)$ neurons and $O(n^3)$ links to sort n numbers. Tseng and Wu [6] have derived a constant-time WTA (winner-take-all) neural network with

[*] Corresponding author. Email: linss@ice.ntnu.edu.tw

[1] This work was supported by the National Science Council of R.O.C. under grant NSC-84-2511-S-003-089.

$O(n^{1+\varepsilon})$ neurons and $O(n^{1+\varepsilon})$ links, $\varepsilon > 0$. The WTA network identifies the neuron with the maximum (or minimum) activation among a set of n neurons. Zwietering et al. [8] show that the minimal number of layers needed are 3 for sorting when using classical multilayered perceptrons with the hard-limiting response function. Wang [7] presents an analog sorting network to be capable of monotonic and bitonic sorting and suitable for hardware implementation. These investigations have showed light on the neural network approach to sorting.

In [3], Chen and Hsieh's sorting network needs $O(n^3)$ neurons and links. In [6], Tseng and Wu's WTA network is nearly cost-optimal, but it can only find the maximum of n numbers. That is, it can not sort the n numbers. In this paper, we improve the results of both [3] and [6]. We derive an $O(1)$ time neural sorting network which needs only $O(n^{1+\varepsilon})$ neurons and $O(n^{1+\varepsilon})$ links, $\varepsilon > 0$.

The paper is organized as follows. In Section 2, the model of neural networks, Chen and Hsieh's sorting network, and Leighton's column sort are introduced. In Section 3, a constant-time, low-cost sorting network is proposed. The implementation issues for our network are discussed in Section 4. Finally, Section 5 concludes this paper.

2. Basic terminologies and definitions

2.1. The model of neural networks

A neural network consists of a number of neurons and links. Neurons can be considered as the processing elements (PEs) in the network. A link is a data path between two neurons. Each neuron sends impulses to other neurons and receives impulses from other neurons. The basic operation of a neuron is depicted in Fig. 1, where $x_1, x_2, \ldots, x_n$, are the values of input data that pass through the links, $w_1, w_2, \ldots, w_n$ are weights on these links, f is the activation function in each neuron, y is the value of output data, and θ is a constant, which represents a threshold value. All links are

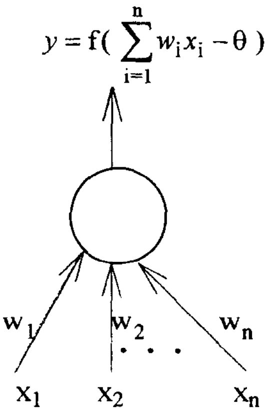


Fig. 1. The operation of a neuron.

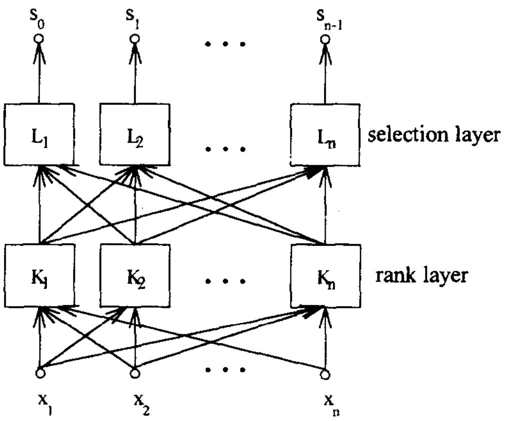


Fig. 2. The block diagram of the Chen and Hsieh's sorting network.

weighted, so that the values of data are multiplied by the weight on the link it passed through.

2.2. Chen and Hsieh's sorting network

Our network uses Chen and Hsieh's sorting network [3] as the lowest-level network. Therefore, we briefly introduce their network in this subsection.

Chen and Hsieh's neural sorting network consists of two layers: rank layer and selection layer. Its block diagram is shown in Fig. 2, where $(x_1, x_2, \ldots, x_n)$ is the input list and $(s_0, s_1, \ldots, s_{n-1})$ is the output list. The rank layer is composed of n modules $K_1, K_2, \ldots, K_n$. Module K_i computes the rank of input x_i. We know the positions of each x_i in the sorted list after their ranks are derived. The selection layer is composed of n modules $L_1, L_2, \ldots, L_n$. Module L_i selects one x_j such that the correct data are selected as the i-th output. The modules L_i are also connected to the inputs. By passing these two layers, the outputs nodes $s_0, s_1, \ldots, s_{n-1}$ constitute the sorted list.

Chen and Hsieh's neural network can sort n numbers in $O(1)$ time, but it has $O(n^3)$ links and neurons. In this paper, we will improve the result to $O(n^{1+\varepsilon})$ neurons and links, where $\varepsilon > 0$.

$$\begin{bmatrix} 6 & 15 & 12 \\ 14 & 4 & 7 \\ 10 & 1 & 13 \\ 3 & 16 & 9 \\ 17 & 8 & 2 \\ 5 & 11 & 0 \end{bmatrix} \qquad \qquad \begin{bmatrix} 0 & 6 & 12 \\ 1 & 7 & 13 \\ 2 & 8 & 14 \\ 3 & 9 & 15 \\ 4 & 10 & 16 \\ 5 & 11 & 17 \end{bmatrix}$$

(a) (b)

Fig. 3. A 6×3 matrix (a) before, and (b) after column sorting.

$$\begin{bmatrix} a & g & m \\ b & h & n \\ c & i & o \\ d & j & p \\ e & k & q \\ f & l & r \end{bmatrix} \begin{array}{c} \xrightarrow{\text{Step 2 (transpose)}} \\[2em] \xleftarrow{\text{Step 4 (untranspose)}} \end{array} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \\ m & n & o \\ p & q & r \end{bmatrix}$$

Fig. 4. The transpose and untranspose permutations in step 2 and step 4.

2.3. *Leighton's column sort*

In order to derive a low-cost neural sorting network, we adopt a novel scheme – Leighton's column sort [5]. Column sort is a generalization of odd-even merge sort. Let Q be an $r \cdot s$ matrix of numbers, where $r \cdot s = n$, $s \mid r$, and $r \geq 2(s-1)^2$. Initially, each entry of the matrix contains one of the n numbers to be sorted. After completion of the algorithm, the i,j entry $(0 \leq i \leq r-1, 0 \leq j \leq s-1)$ of Q will contain the p-th sorted number $(0 \leq p \leq n-1)$, where $p = i + j \cdot r$. That is, the sorted elements are put in a column-major-ordering fashion. An example is shown in Fig. 3.

Column sort has eight steps. In steps 1, 3, 5, and 7, the numbers within each column are sorted. In steps 2, 4, 6, and 8, the entries of the matrix are permuted. The permutation in step 2 corresponds to a "transpose" operation of the matrix. The permutation in step 4 is the inverse of that in step 2. Fig. 4 shows the operations of step 2 and step 4. The permutation in step 6 corresponds to an $(r/2)$-shift of the matrix. The permutation in step 8 is the inverse of that in step 6. The shift and unshift permutations in step 6 and step 8 are shown in Fig. 5. The $-\infty$'s denote arbitrarily small dummy elements, and the ∞'s denote arbitrarily large dummy elements.

3. A new low-cost neural sorting network

In this section, we present a low-cost $O(1)$-time neural sorting network that is based upon Leighton's eight-step column sort [5] and Chen and Hsieh's neural sorting network [3]. By using the recursion technique, we are able to reduce the number of links and neurons to $O(n^{1+\varepsilon})$, where $\varepsilon > 0$.

We show the block diagram of our network in Fig. 6, where $(x_1, x_2, \ldots, x_n)$ is the input list and $(v_1, v_2, \ldots, v_n)$ is the sorted output result. That is, $v_1 \leq v_2 \leq \ldots \leq v_n$. We let $r \cdot s = n$, $s \mid r$, and $r \geq 2(s-1)^2$, $s \geq 1$.

$$\begin{bmatrix} a & g & m \\ b & h & n \\ c & i & o \\ d & j & p \\ e & k & q \\ f & l & r \end{bmatrix} \begin{array}{c} \xrightarrow{\text{Step 6 (shift)}} \\[2em] \xleftarrow{\text{Step 8 (unshift)}} \end{array} \begin{bmatrix} -\infty & \mathrm{d} & \mathrm{j} & \mathrm{p} \\ -\infty & \mathrm{e} & \mathrm{k} & \mathrm{q} \\ -\infty & \mathrm{f} & \mathrm{l} & \mathrm{r} \\ \mathrm{a} & \mathrm{g} & \mathrm{m} & \infty \\ \mathrm{b} & \mathrm{h} & \mathrm{n} & \infty \\ \mathrm{c} & \mathrm{i} & \mathrm{o} & \infty \end{bmatrix}$$

Fig. 5. The shift and unshift permutations in step 6 and step 8.

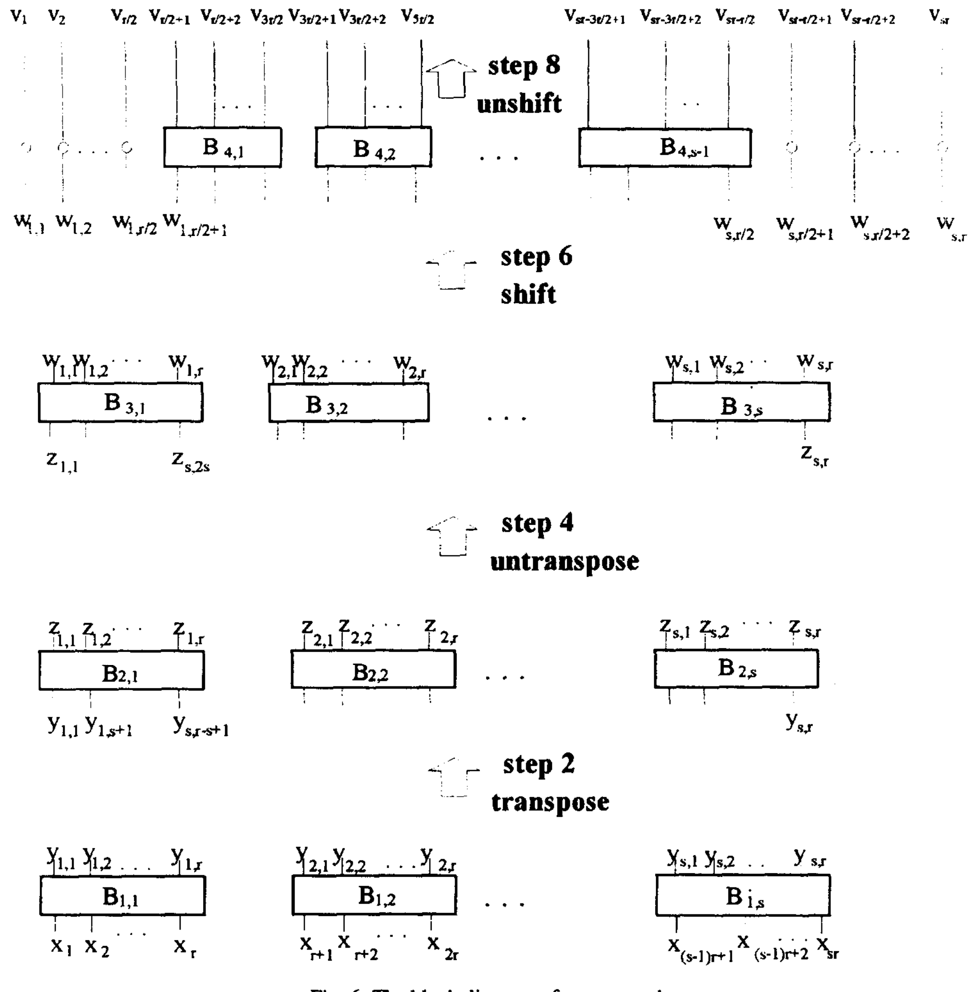


Fig. 6. The block diagram of our network.

We implement the eight-step column sort [5] by using these sub-networks B_{ij}, and links between them, $1 \leq i \leq 4$, $1 \leq j \leq s$. Each sub-network B_{ij} sorts its input-list and outputs the sorted values. The links between these sub-networks perform the column-sort operations which include transpose, untranspose, shift and unshift. Fig. 7 shows these operations in detail. It is clear that if all the sub-networks sort their inputs in constant time, the whole sorting network will take $O(1)$ time. Our network is a recursive scheme. We use Chen and Hsieh's network [3] as the lowest-level sub-networks. Now let us take into account the depth of the recursive construction.

Case 1. No recursion.

Here we use Chen and Hsieh's $O(1)$-time neural sorting network as all B_{ij}, $1 \leq i \leq 4$, $1 \leq j \leq s$. Chen and Hsieh's sorting network can sort r data in $O(1)$ time by using

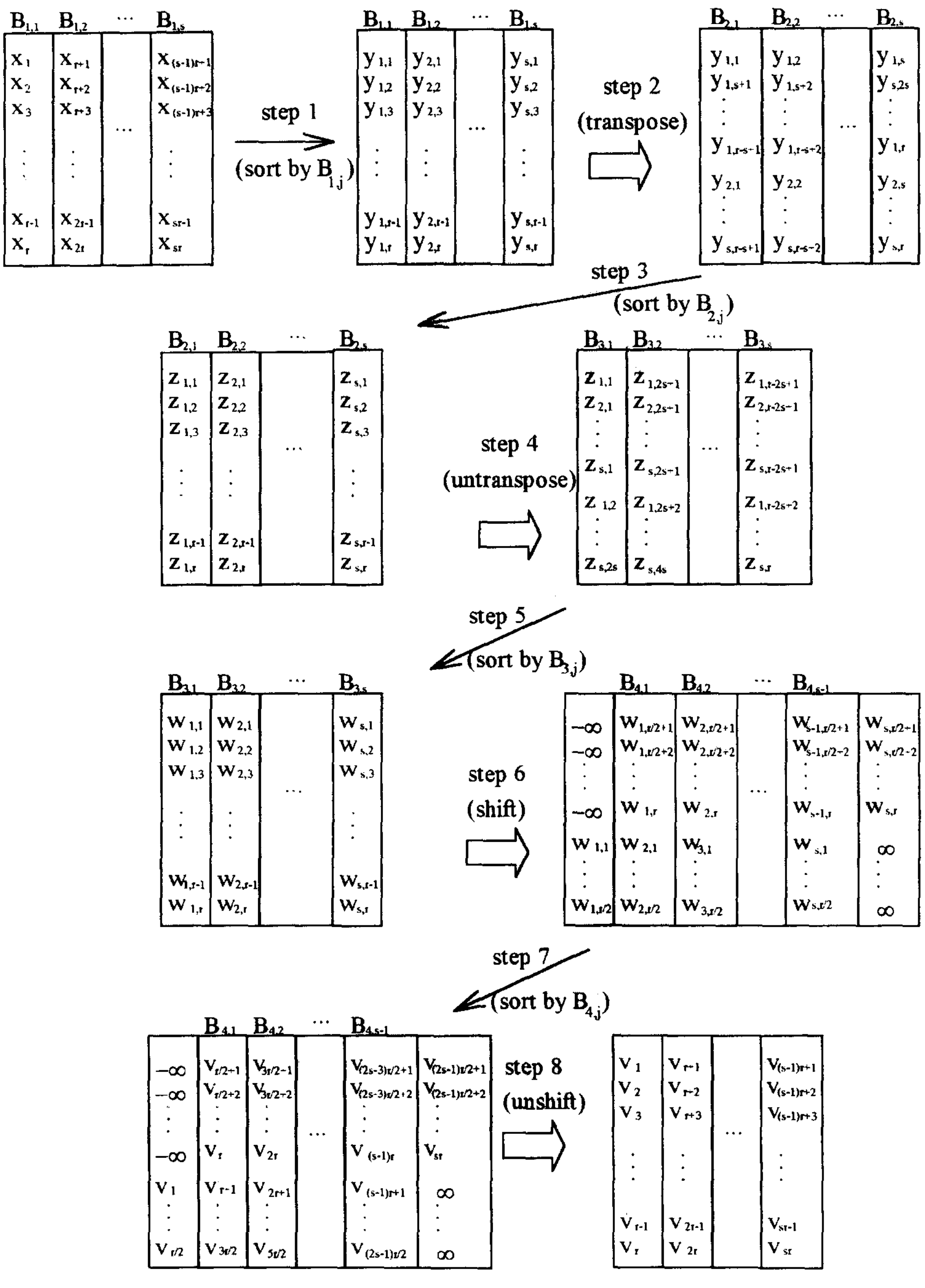


Fig. 7. The eight-step column sort.

$O(r^3)$ neurons and links. The input list $(x_1, x_2, \ldots, x_n)$ is divided into s sublists. Each sublist has r elements. In order to satisfy the condition of column sort: $s \mid r$ and $r \geq 2(s-1)^2$, we let $r = 2s^2$. In this case, $n = r \cdot s = 2s^3$. That is, $s = (n/2)^{(1/3)}$ and $r = 2(n/2)^{(2/3)}$. Each B_{ij} sorts its sublist individually and outputs the sorted values. After the operations of B_{ij}, these sorted values become the inputs of B_{i+1j}. As the block diagram of Fig. 6 shows, the outputs $v_1, v_2, \ldots, v_n$, are sorted after four sort stages and four permutation stages have been done. Since Chen and Hsieh's sorting network takes $O(1)$ time, so does our network. The number of neurons and links in each B_{ij} is $O(r^3)$. So the total neurons and links in our network are $O(4 \cdot s \cdot r^3) = O(n^{7/3}) \approx O(n^{2.33}) < O(n^3)$. This $O(n^{2.33})$ result is better than that of Chen and Hsieh's network which needs $O(n^3)$ neurons and links to sort n numbers. We call this network "Case 1" network.

Case 2. Recursively construct once.

The "Case 1" network derived above can be used recursively as B_{ij} shown in Fig. 6. Now each B_{ij} has $r = 2s^2$ inputs and contains $O(r^{7/3})$ neurons and links. The total number of neurons and links in the whole network is $O(4 \cdot s \cdot r^{7/3}) = O(n^{(7/3)\cdot(2/3)} \cdot n^{1/3}) = O(n^{17/9}) \approx O(n^{1.89})$. We call this network "Case 2" network.

Case 3. Recursively construct k times.

As we showed, the "Case 2" network needs $O(n^{1.89})$ neurons and links. We can further apply the recursion in our network. Assume the "recursively construct k times" network needs $O(n^{a_k})$ neurons and links. We have the following lemma:

Lemma 1. The "recursively construct k times" network needs $O(n^{a_k})$ neurons and links, where $a_k = 1 + 2(2/3)^{k+1}$, $k = 0,1,2,\ldots$

Proof. We can use the "recursively construct $k-1$ times" network as all B_{ij} in the "recursively construct k times" network. Now each B_{ij} has $r = 2s^2$ inputs and contains $O(r^{a_{k-1}})$ neurons and links. The total number of neurons and links in the whole network are $O(4 \cdot s \cdot r^{a_{k-1}}) = O(4 \cdot (n/2)^{(1/3)} \cdot 2 \cdot (n/2)^{(2/3)a_{k-1}}) = O(n^{(2/3)a_{k-1}+(1/3)}) = O(n^{a_k})$. From this, we know that $a_k = (2/3)a_{k-1} + 1/3$ and $a_0 = 7/3$. For $k = 0$, $a_0 = 1 + 2(2/3)^{0+1} = 1 + 4/3 = 7/3$, the statement is true. Now assume the result is true for $k = m-1$, i.e., $a_{m-1} = 1 + 2(2/3)^{(m-1)+1} = 1 + 2(2/3)^m$. When $k = m$, $a_m = (2/3)a_{m-1} + 1/3 = (2/3)(1 + 2(2/3)^m) + 1/3 = 1 + 2(2/3)^{m+1}$. By using the mathematic induction, we have proved Lemma 1. $\square$

Table 1
Connection complexities of our network for small k

Recursion times k	Connection complexity
0	$O(n^{7/3}) \approx O(n^{2.33})$
1	$O(n^{17/9}) \approx O(n^{1.89})$
2	$O(n^{43/27}) \approx O(n^{1.59})$
3	$O(n^{113.81}) \approx O(n^{1.40})$
4	$O(n^{306/243}) \approx O(n^{1.26})$
5	$O(n^{857/729}) \approx O(n^{1.18})$

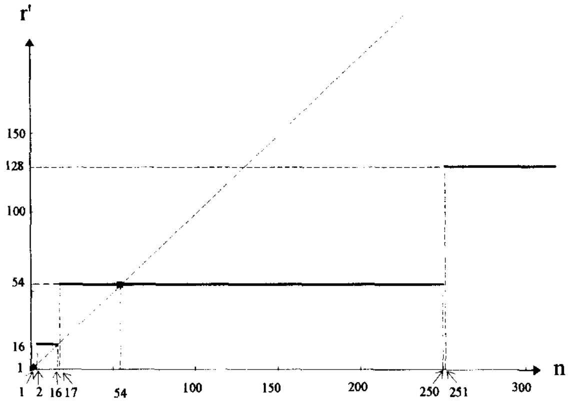


Fig. 8. The relation between r' and n.

The result $a_k = 1 + 2 \cdot (2/3)^{k+1}$ means that if we recursively construct the network k times, we need $O(n^{a_k}) = O(n^{1+2(2/3)^{k+1}})$ neurons and links. Table 1 shows the connection complexities of this scheme for small k. At this moment, an $O(1)$ time neural sorting network with $O(n^{1+\varepsilon})$ neurons is derived, where $\varepsilon = 2 \cdot (2/3)^{k+1} > 0$.

4. Implementation of our network

In the previous section, we assume that the problem size n is of the form: $2s^3$ for some integer s, where $s \geq 1$. In most situations, n does not fit this form. That is, there does not exist an integer s such that $2s^3 = n$ justly. We let $n' = 2 \cdot \lceil (n/2)^{(1/3)} \rceil^3$ be a proper value that is the smallest number which is greater than or equal to n and has the form.

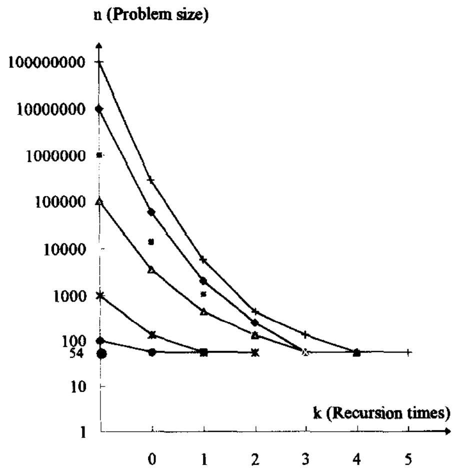


Fig. 9. The decreasing effect of problem size $n = 10^i$, where $i = 1 \ldots 8$.

Table 2
Practical implementation of our network

n	$n' = 2\cdot\lceil(n/2)^{(1/3)}\rceil^3$	$s = (n'/2)^{(1/3)}$	$r = 2s^2$	$r' = 2\cdot\lceil(r/2)^{(1/3)}\rceil^3$	k	Connection complexity	Depth
1 ~ 2	2	1	2	2		$O(n^3)$	5
3 ~ 16	16	2	8	16		$O(n^3)$	5
17 ~ 54	54	3	18	54		$O(n^3)$	5
55 ~ 250	250	5	50	54	0	$O(n^{7/3}) \approx O(n^{2.33})$	20
251 ~ 2662	2662	11	242	250	1	$O(n^{17/9}) \approx O(n^{1.89})$	80
2662 ~ 93312	93312	36	2592	2662	2	$O(n^{43/27}) \approx O(n^{1.59})$	320
93312 ~ 20155392	20155392	216	93312	93312	3	$O(n^{113/81}) \approx O(n^{1.40})$	1280
20155393 ~ 6.39515E + 10	6.39515E + 10	3174	20148552	20155392	4	$O(n^{307/243}) \approx O(n^{1.26})$	5120
6.39515E + 10 ~ 1.14355E + 16	1.14355E + 16	178817	6.39510E + 10	6.39515E + 10	5	$O(n^{857/729}) \approx O(n^{1.18})$	20480

It is easy to show that the connection complexity of sorting n numbers is equal to the connection complexity of sorting n' numbers in our network. However, we should stop applying the recursion, if the recursion will not take advantage any more. This occurs when $n \le 54$. That is, $r' = 2 \cdot \lceil (r/2)^{(1/3)} \rceil^3 \ge n$ when $n \le 54$, where $r = 2s^2 = 2 \cdot \lceil (n/2)^{(1/3)} \rceil^2$. See Table 2 for a depiction. Fig. 8 shows the relation between r' and n. Therefore, we use Chen and Hsieh's sorting network [3] directly when $n \le 54$.

It is clear that the recursion times k get a restriction with a specific n. The practical implementation of our network is show in Table 2. For example, if we want to sort $n = 10^8$ numbers, we can recursively construct the network k $(= 3)$ times, and its depth is 1280 layers. The depth of our network can be derived as follows. In Case 1, our network uses Chen and Hsieh's networks as B_{ij}. Since the depth of Chen and Hsieh's network is 5, the depth of "Case 1" network's depth is $4 \cdot 5 = 20$. "Case 2" network uses "Case 1" networks in B_{ij}, so it's depth is $4 \cdot 20 = 80$. A "recursively construct k times" network uses b_k layers, where $b_k = b_{k-1} \cdot 4$, $b_0 = 20$. Therefore, $b_k = 20 \cdot 4^k = 5 \cdot 4^{k+1}$. Fig. 9 shows the decreasing effect of problem size n as the recursion times increase, where $n = 10^i$, $i = 1 \ldots 8$.

5. Conclusion

In the past, neural networks were used to solve problems about pattern recognition, artificial intelligence, and so on. Today, many researchers use neural networks to solve general problems, to name just a few, the WTA problem [6], sorting [3] or NP-complete problems [2]. To successfully apply neural networks to solve these problems, we must reduce the hardware-cost of the networks used. Our study goes toward this goal by presenting a low-cost neural sorting network. Since the sequential time complexity for sorting is $O(n\log n)$, it is interest to know whether there exists an $O(1)$-time neural network with $O(n\log n)$ neurons and links. This problem remains open. It is well known that sorting is of fundamental importance in computer science. Sorting also finds applications in the solution of a huge number of complex problems. We hope that this paper will prompt researchers to study related problems such as the selection problem, the minimum(maximum)-finding problem, the integer sorting problem.

Acknowledgements

We thank the anonymous reviewers for their helpful comments on an earlier version of the paper.

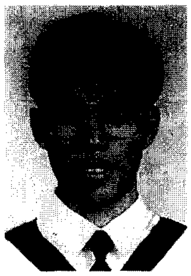
References

[1] S.G. Akl, *The Design and Analysis of Parallel Algorithms* (Prentice-Hall International Editions, 1989).

[2] S. Bhide, N. John, and M.R. Kabuka, A Boolean neural network approach for the traveling salesman problem, *IEEE Trans. Comput.* 42(10) (1993) 1271–1278.

[3] W. Chen and K. Hsieh, A neural sorting network with $O(1)$ time complexity, *Information Processing Letters* 45(6) (1993) 309–313.
[4] M. Chester, *Neural Networks: A Tutorial* (Prentice-Hall, 1993).
[5] T. Leighton, Tight bounds on the complexity of parallel sorting, *IEEE Trans. Comput.* C-34(4) (1985) 344–354.
[6] Y. Tseng and J. Wu, On a constant-time, low-complexity winner-take-all neural network, *IEEE Trans. Comput.* 44(4) (1995) 1–3.
[7] J. Wang, Analysis and design of an analog sorting network, *IEEE Trans. Neural Networks* 6(4) (1995) 962–971.
[8] P.J. Zwietering, E.H.L. Aarts, and J. Wessels, The minimal number of layers of a perceptron that sorts, *J. Parallel Distributed Comput* 20 (1994) 380–387.

Shun-Shii Lin received the B.S. degree in Computer Engineering from the National Chiao-Tung University in 1981, the M.S. degree and the Ph.D. in Computer Science and Information Engineering from the National Taiwan University, Taiwan, ROC, in 1985 and 1990, respectively. In 1986, He joined the National Taiwan Normal University, Taiwan, ROC, where he is now the department head of information and computer education. His current research interests include VLSI routing, real-time scheduling, parallel algorithms and neural computing.



Shen-Hsuan Hsu received the B.S. degree in Applied Mathematics from the National Sun-Yet-Sen University, Taiwan, ROC in 1994. He is now a graduate student at the department of information and computer education, National Taiwan Normal University, Taiwan, ROC. His research interests include neural computing and parallel algorithms.