Regular Paper

Parallel Algorithms for a Class of Graph Theoretic Problems

MA JUN,† TADAO TAKAOKA†† and MA SHAOHAN†

Different from the known algorithms to compute the all pair shortest paths for a weighted, directed graph $G=(V, E, \text{COST})$, an $O(|V|^3/p)$ parallel algorithm running on the CREW PRAMs with p, $1 \leq p \leq |V|^2$, processors is presented, which not only computes the distance from vertex i to vertex j in G, but also records the forward and backward shortest path trees rooted at i, $i \in V$, of G. For any pair $i, j \in V$, the shortest path P from i to j can be found in $O(|P|)$ time, where $|P|$ is the number of edges in P. It is pointed out that the parallel algorithm can be updated properly to calculate the transitive closure of G and some graph algorithms can be derived from above computations. The ways to parallelize these derived graph algorithms in known parallelizing techniques are also given.

1. Introduction

Parallel algorithms are of two types, that is, unbounded parallelism and bounded parallelism. In unbounded parallelism, parallel algorithms are developed assuming that arbitrarily many processors are available in order to yield insight into the maximum amount of parallelism inherent in a particular problem. On the more practical side, in bounded parallelism, the number of processors used in parallel algorithms is limited to be independent of the size of the problem to be solved.

Both unbounded and bounded parallel algorithms for graph problems have received considerable attention in the past.[1)-3),8)-12),14)-17)] For the all pair shortest path problem (APSP) in a weighted directed graph $G=(V, E, \text{COST})$, $|V|=n$, Reif and Spirakis[17)] proposed an $O(\log n \log\log n)$ time complexity with $O(n^3)$ processors parallel algorithm on CREW PRAMs. Frieze et al.[2)] presented an $O(\log n)$ with $O(n^3)$ processors algorithm on CREW PRAMs. On bounded parallelism, Deo et al.[1)] gave an $O(n^3/p)$ with $O(p)$ processors algorithm on CREW PRAMs.

Until now, algorithms for APSP only means to compute the shortest distance matrix D, but even if D is known, we can not find the shortest path P from vertex i to j in $O(|P|)$ time, where $|P|$ is the number of edges in path P.

In this paper, we reconsider the algorithms for APSP, and propose an $O(n^3/p)$ parallel algorithm for APSP running on CREW PRAMs with p, $1 \leq p \leq n^2$, processors, which not only computes the shortest distance matrix D, but also records all pair shortest paths in a matrix S. For any pair $i, j \in V$, we can find the shortest path P from i to j in $O(|P|)$ time, where $|P|$ is the number of edges in P. Moreover, we prove that the matrix S also records the forward and backward shortest path trees rooted at every vertex of G. We show some graph algorithms can be derived from D and S and give the ways to parallelize these derived algorithms on CREW PRAMs with p processors in known parallelizing techniques.

2. Preliminaries

A weighted directed graph $G=(V, E, \text{COST})$ is an ordered triple of the set V of n vertices numbered from 0 to $n-1$, the set E of edges and a function COST that maps into real numbers. The function COST is usually given by a matrix COST $(0 \cdot\cdot n-1, 0 \cdot\cdot n-1)$, where COST (i, j) is the weight of the edge from vertex i to j. Here let COST $(i, i)=0$ and COST $(i, j)=\infty$ if there is no edge from i to j, $0 \leq i, j \leq n-1$. A vertex

† Department of Computer Science, Shandong University, Jinan City, P. R. China
†† Department of Computer Science, Ibaraki University, Hitachi, Ibaraki 316, Japan

j is said to be reachable from i if there is a directed path from i to j. The distance $D(i, j)$ from vertex i to j in G is the minimum of the sums of the weights of the edges over the paths from i to j, and the path corresponding to the minimum sum is called the shortest path from i to j. For a $v \in V$, $E(v) = \max\{D(i, v) | i \in V\}$ is called the centrifugal rate of v, and the vertex v with the minimum centrifugal rate is called the center of G. If G is an undirected graph, the diameter of G is the maximum distance between two vertices of G.

The matrix A with the property that $A(i, j) = 1$ if pair $(i, j) \in E$, 0 otherwise and $A(i, i) = 1$, $0 \leq i, j \leq n-1$, is called the adjacency matrix of G. The matrix A^* with the property that $A^*(i, j) = 1$ if there is a path of length ≥ 0 from i to j and 0 otherwise is the transitive closure of G.

A forward shortest path tree rooted at i of G[4] is a subtree $T_f(i) = (X, S)$ of G, such that:

(1). $x \in X$ iff x is reachable from i in G and one of the shortest path from i to x in G is kept in $T_f(i)$.

(2). S is the edge set of $T_f(i)$, $S \subseteq E$.

A backward shortest path tree rooted at i of G[4] is a subtree $T_b(i) = (X, S')$ of G, such that:

(1). $x \in X$ iff i is reachable from x in G and one of the shortest path from x to i in G is kept in $T_b(i)$ but the directions of edges are reversed.

(2). S' is the edge set of $T_b(i)$, $S' \subseteq E^R$, $E^R = \{(y, x) | (x, y) \in E\}$.

An example of $T_f(0)$ and $T_b(0)$ in a weighted directed graph G is shown in **Fig. 1**.

The applications of the shortest path trees are given in Ref. 4)-7).

A PRAM (parallel random access machine) consists of a finite number p of processors operating synchronously on common, shared memory cells. We assume that the processors are numbered $1 \cdot\cdot p$ and that each processor is able to implement some sequential subroutines independently; One among various types of PRAMs is an EREW (exclusive read exclusive write) PRAM that allows no memory cell to be accessed simultaneously by more than one processors. In contrast, CRCW (concurrent read concurrent write) PRAMs allow simultaneous reading as well as simultaneous writing of each cell by an arbitrary set of processors. CREW (concurrent read exclusive write) PRAMs allow simultaneous reading but not simultaneous writing.

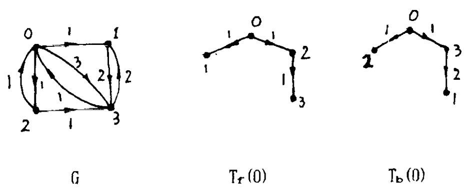

Fig. 1 An example of $T_f(0)$ and $T_b(0)$ in a weighted directed graph G.

The speed up of a parallel algorithm over a sequential one is $S_p = T_1/T_p$, where T_1, T_p are the running time of the sequential algorithm and the parallel one with p processors for the same problem respectively. A parallel algorithm is said to be efficient when $S_p/p = O(1)$. The cost of a parallel algorithm is the product of the parallel running time and the number of processors used.

3. Algorithms for APSP and the Transitive Closure

A well known sequential algorithm for APSP given by Floyd[13] can be described as follows.

Input: D^{-1}, the COST of a directed graph G without negative cycles.

Output: D^{n-1}, $D^{n-1}(i, j)$ is the distance from i to j.

Algorithm 1 (FLOYD)

```
FOR K:=0 TO n-1 DO
  FOR I:=0 TO n-1 DO
    FOR J:=0 TO n-1 DO
      D(i, j):=min{D(i, j), D(i, k)
                  +D(k, j)}.
```

The principle of the Floyd algorithm is to generate $D^0, D^1, \cdots, D^{n-1}$ successively by the following formulas.

$$D^{-1}(i, j) = \text{COST}; \quad (1)$$

$$D^k(i, j) = \min\{D^{k-1}(i, j), D^{k-1}(i, k) + D^{k-1}(k, j)\} \quad (2)$$

$$0 \leq k \leq n-1.$$

It is obvious that the time complexity of Floyd algorithm is $O(n^3)$. Floyd algorithm only computes the all pair shortest distance matrix D^{n-1}. Now we add a new function to Floyd algorithm, recording the shortest paths corresponding to D^{n-1}. We use an array S, where $S(i, j)$ is the successor of vertex i in the shortest path from i to j, $0 \leq i, j \leq n-1$. The new sequential algorithm is as follows.

Algorithm 2.

Step 1.

1.1 $D := \mathrm{COST}$;

1.2 $S(i, j) := j$; $0 \leq i, j \leq n-1$.

Step 2.

```
2.1  FOR k := 0 TO n-1 DO
2.2    FOR i := 0 TO n-1 DO
2.3      FOR j := 0 TO n-1 DO
2.4        IF D(i, j) > (D(i, k) + D(k, j))
             THEN
2.5          D(i, j) := D(i, k) + D(k, j);
2.6          S(i, j) := S(i, k)
2.7        ENDIF
2.8      ENDFOR
2.9    ENDEOR
2.10 ENDFOR
```

Theorem 1. When algorithm 2 terminates, the following two propositions are true.

(1) Matrix S records the shortest path from i to j, $0 \leq i, j \leq n-1$ corresponding to $D^{n-1}(i, j)$, and the shortest path from i to j can be found in $O(|P|)$time.

(2) Matrix S records both the forward shortest path tree rooted at i and the backward shortest path tree rooted at i, $0 \leq i \leq n-1$.

Proof. (1) Clearly, the computation of matrix D in algorithm 2 is the same as that in algorithm 1, at step 1 of algorithm 2. We let $S(i, j) := j$, $0 \leq i, j \leq n-1$, that is, we suppose there is an edge for pair i, j of G, $0 \leq i, j \leq n-1$, because when pair (i, j) is not in E, we can imagine there is an edge (i, j) with the weight ∞ from i and j. In the step 2 when a shorter path from i to j via k is found, we update the shortest path from i to j by the assignment $S(i, j) := S(i, k)$, that is, we change the successor of i in the path from i to j by the successor of i in the path from i to k. It is clear that when algorithm 2 terminates, if $D(i, j) = \infty$, there is no path from i to j; otherwise, by the definition of $S(i, j)$, the shortest path P from i to j is the sequence of $(i, S(i, j), S(S(i, j), j), \cdots, j)$. It is easy to output P in $O(|P|)$ time.

(2) Based on (1), for any i, j, $0 \leq i, j \leq n-1$, if j is reachable from i, one of the shortest path P from i to j is in S. Suppose for a vertex w in G, there are two shortest path P_1 and P_2 from i to w recorded in S, as shown in **Fig. 2**.

$$P_1 = i v_{i_1} v_{i_2} \cdots v_{i_k} v_{i_{k+1}} \cdots w,$$

$$P_2 = i v_{i_1} v_{i_2} \cdots v_{i_k} v'_{i_{k+1}} \cdots w,$$

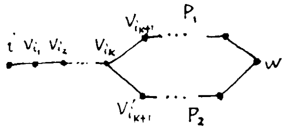

Fig. 2 The sketch for proving theorem 1.

where $v_{i_{k+1}} \neq v'_{i_{k+1}}$ and $k \geq 0$, that is, v_{i_k} is the first vertex whose successor in P_1 is different from whose successor in P_2. Because we suppose both P_1 and P_2 are in S, if we consider P_1, we have $S(v_{i_k}, w) = v_{i_{k+1}}$. If we consider P_2, we have $S(v_{i_k}, w) = v'_{i_{k+1}}$. Because of $v_{i_{k+1}} \neq v'_{i_{k+1}}$ we have $S(v_{i_k}, w) \neq S(v_{i_k}, w)$, which is a contradiction. So we prove that for any i, j, $0 \leq i, j \leq n-1$, if j is reachable from i, one and only one of the shortest path from i to j in G is recorded in S. We can infer further that there is no intersect vertex w for any two paths starting from i in S, which ensures that there is no loops in the connected subtree T consisting of the shortest paths from i to j, $0 \leq j \leq n-1$, in S. By the definition of $T_f(i)$, clearly T is one of the forward shortest path tree rooted at i of G.

For the same reason we can prove that the connected subgraph T' consisting of the shortest path from j to i, $0 \leq j \leq n-1$, in S is one of the backward shortest path tree rooted at i of G.□

Now let us consider the parallelization of algorithm 2 on CREW PRAMs with p processors. A parallel version of Algorithm 2 is as follows.

Algorithm 3 (Parallel Version of algorithm 2)

Step 1

1.1 FOR $i := 1$ TO p DO IN PARALLEL
P_i calls procedure Comp1 $(i-1)$;

Step 2

2.1 FOR $k := 0$ TO $n-1$ DO

2.2 FOR $i := 1$ TO p DO IN PARALLEL
P_i calls the procedure Comp2 $(i-1, k)$.

The declarations of procedure Comp1 and Comp2 are:

```
Procedure Comp1(x);
i, j, t, bound: INTEGER;
BEGIN
  1  t := x; bound := n * n;
  2  WHILE t < bound DO
  3  j := t mod n; i := t/n
```

```
  4  D(i, j) := COST(i, j);
  5  S(i, j) := j;
  6  t := t + p
  7  ENDWHILE;
END Comp1;
Procedure Comp2(x, k);
i, j, t, bound: INTEGER;
BEGIN
  1  t := x; bound := n * n;
  2  WHILE t < bound DO
  3    j := t mod n; i := t/n
  4    IF D(i, j) > (D(i, k) + D(k, j)) THEN
  5      D(i, j) := D(i, k) + D(k, j);
  6      S(i, j) := S(i, k);
  7    ENDIF
  8    t := t + p
  9  ENDWHILE;
END Comp2;
```

Theorem 2. Algorithm 3 calculates arrays D^{n-1} and S correctly on the CREW PRAMs with p, $1 \leq p \leq n^2$, processors in $O(n^3/p)$ time.

Proof. Suppose we have p, $1 \leq p \leq n^2$, processors, the procedure Comp1 and Comp2 are in the local memory of every processor and *D and S* are in the common, shared memory. We want to use p processors to update the elements of *D and S* concurrently and let every processor calculate $O(n^2/p)$ elements of D and S in one call to procedures Comp1 and Comp2. Because the function $F(i, j) = in + j$ is an 1-1 function from pair (i, j) to one dimension array index x, we can let processor P_i call the procedure Comp1 $(i-1)$ and Comp2 $(i-1, k)$ to update the elements of $D(m, l)$ and $S(m, l)$ whose array indices satisfy $(mn+1) \bmod p = (i-1) \bmod p$, $0 \leq i \leq p-1$. Because it is clear that every element of D can be calculated once and only once in one call to Comp1 and Comp2, the correctness of step 1 is obvious.

In step 2 although on a CREW PRAM model, $D(i, j)$ is updated based on one of the following four randomly chosen formulas.

$$D^k(i, j) = \min\{D^{k-1}(i, j),\ D^{k-1}(i, k) + D^{k-1}(k, j)\} \quad (3)$$

$$D^k(i, j) = \min\{D^{k-1}(i, j),\ D^{k-1}(i, k) + D^{k}(k, j)\} \quad (4)$$

$$D^k(i, j) = \min\{D^{k-1}(i, j),\ D^{k}(i, k) + D^{k-1}(k, j)\} \quad (5)$$

$$D^k(i, j) = \min\{D^{k-1}(i, j),\ D^{k}(i, k) + D^{k}(k, j)\} \quad (6)$$

$0 \leq k \leq n-1$.

if we note that $D^k(k, k) = 0$, $0 \leq k \leq n-1$, it is easy to see

$$\begin{aligned} D^k(i, k) &= \min\{D^{k-1}(i, k),\ D^{k-1}(i, k) + D^{k-1}(k, k)\} \\ &= \min\{D^{k-1}(i, k),\ D^{k-1}(i, k) + 0\} \\ &= D^{k-1}(i, k), \end{aligned}$$

and $D^k(k, j) = D^{k-1}(k, j)$ for the same reason. So in the procedure of Comp2, whichever operation of the above four is used, the result is the same. It also ensures the correctness of the computation to array S, so algorithm 3 is a correct parallel version of algorithm 2. Because the time complexity of two procedures is $O(n^2/p)$, on CREW PRAMs simultaneous reading by more than one processors is allowed and there is no writing conflict in algorithm 3, the time complexity of algorithm 3 is $O(n^3/p)$. □

Corollary 1. Algorithm 3 is of the linear speedup to Floyd sequential algorithm for APSP.

If we change the line 4 of algorithm 1 as:

$$D(i, j) := \max\{D(i, j),\ D(i, k) * D(k, j)\}; \quad (7)$$

and D^{-1} is the adjacent array A of a graph G, the changed algorithm 1 becomes the Warshall's[18] algorithm to compute the transitive closure A^*. Clearly, if we replace the lines 4-5 of Comp1(x) with $D(i, j) := A(i, j)$ and the lines 4-7 of Comp2 (x, k) with formula 3.7, the changed algorithm 3 become a parallel version of Warshall algorithm, so we have theorem 3.

Theorem 3. On the CREW PRAMs with p, $1 \leq p \leq n^2$, processors, the transitive closure A^* of a graph can be calculated in $O(n^3/p)$ time.

4. Graph Algorithms Derived from *D* and *S* and Their Parallelization

Because we record the all pair shortest paths in a matrix S when we compute the all pair shortest distance matrix D, some graph algorithms can be derived from D and S. Since an undirected graph can be considered as a special case of a directed graph, so algorithm 3 is valid for an undirected graph. Let us use D and S to represent the distance array and the successor array for both directed graphs and undirected graphs, the following give the graph algorithms derived from D and S.

(a). to determine the center of a directed graph G.

Step 1. $E(j) = \max_{0 \leq i \leq n-1}\{D(i, j)\}$; $0 \leq j \leq n-1$.

Step 2. $E(k) = \min_{0 \leq j \leq n-1}\{E(j)\}$.

The vertex k is the center of G.

(b). to calculate the diameter d of an undirected graph G and the corresponding path.

Step 1. Let $D(i', j') = \max_{0 \leq i,j \leq n-1}\{D(i, j)\}$

Step 2. $P = (i', S(i', j'), S(S(i', j'), j'), \cdots, j')$ is the diameter of G and $D(i', j')$ is the length.

(c). to search for a directed cycle with the minimum (maximum) length in a directed graph.

Step 1. $D(i, i) := \infty(-\infty)$; $0 \leq i \leq n-1$.

Step 2. $D(i', j') = \min(\max)\ \{D(i, j) + D(j, i) | D(i, j) \neq \infty$ and $D(j, i) \neq \infty, 0 \leq i, j \leq n-1\}$.

Step 3. $(i', S(i', j'), S(S(i', j'), j'), \cdots, j', S(j', i'), S(S(j', i'), i'), \cdots, i')$ is the minimum (maximum) length directed cycle in C.

The problems in (a) and (b) are easy, while the problem in (c) is a little difficult, our algorithm is of the most concise form and easy to be implemented.

More algorithms for graph problems can be developed by D, S and A^* in the same way. Clearly, the key to parallelize derived algorithms is to compute the minimum (maximum) value of a set of n elements in parallel. Supposing n elements are stored in an array $A(1 \cdot\cdot n)$, one such method is to use min (max) for the associative operator $\odot$ in the following parallel algorithm.

Algorithm 4

```
For k := 0 to ⌈lgn⌉ - 1 DO
   For i := 2^k + 1 TO n DO IN PARALLEL
   A(i) := A(i) ⊙ A(i - 2^k);
   END FOR;
END FOR;
```

The minimum (maximum) value is available as $A(n)$ when the algorithm 4 terminates. The time complexity of algorithm 4 is $O(\log n)$ if n processors are available. If only p processors can be used, $1 \leq p \leq n$, each processor can find the minimum of n/p elements in linear time and then a minimum can be found among the p candidates in $O(\log p)$ additional time using the above algorithm. These results can be summarized in the following theorem.

Theorem 4. Given an associative, binary operator $\odot$ computable in constant time and a expressions $A(1) \odot A(2) \odot \cdots \odot A(n)$ can be computed in $O(n/p + \log p)$ steps on a p processor EREW machine, $1 \leq p \leq n$.

It is obvious that parallel algorithms running on EREW PRAMs are the parallel algorithms running on CREW PRAMs, for the case of two dimension array. We can map two dimension array indices to one dimension array indices as we have done in designing algorithm 3, so if we suppose matrices D and S have been calculated, based on theorem 4, the algorithms given in (a), (b), (c) can be implemented parallelly on CREW PRAMs with p, $1 \leq p \leq n^2$, processors in $O(n^2/p + \log p)$ time.

5. Conclusion

All parallel algorithms presented in this paper are the best for the time being with the respect to the time-processor product. As shown in § 4, the matrix S makes APSP have more applications. The two dimension array data structure to store the forward and backward shortest path tree rooted at every vertex i of G can simplify some graph algorithms. One example is the on-line APSP problem, for the incremental algorithm[7] to update D and the shortest paths during edge insertions and edge cost decreases. In addition, it is clear that we can get unbounded and bounded parallel algorithms to compute both D and S by replacing the statement of $D(i, j) := \min\{D(i, j), D(i, k) + D(k, j)\}$ in known parallel algorithms for APSP with the IF statement in the lines 2.4–2.7 of algorithm 2.

References

1) Deo, N., Pang, C. Y. and Lord, R. E.: Two Parallel Algorithms for Shortest Path Problems, *Proc. 1980 Intern. Conf. on Parallel Processing*, pp. 244–253 (1980).

2) Frieze, A. and Rudolph, L.: A Parallel Algorithm for All Pairs Shortest Paths in a Random Graph, *Proc. of the 22nd Allerton Conf.*, Univ. of Illinois, pp. 663–670 (1984).

3) Hirschberg, D. S.: Parallel Algorithms for the Transitive Closure and the Connected Component Problem, *Proceedings of the 8th Annual*

ACM Symposium on the Theory of Computing, pp. 55-57, ACM, New York (1976).
4) Italiano, G. F.: Amortized Efficiency of a Path Retrieval Data Structure, *Theoret. Comput. Sci.,* Vol. 48, No. 2, 3, pp. 273-281 (1986).
5) Italiano, G. F.: Finding Paths and Deleting Edges in Directed Acyclic Graphs, *Inform. Process. Lett.,* Vol. 28, No. 1, pp. 5-11 (1988).
6) Italiano, G. F., Spaccamela, A. M. and Nanni, U.: Dynamic Data Structures for Series Parallel Graphs, *Proceedings, Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science,* Vol. 382, pp. 352-372, Springer-Verlag, Berlin (1989).
7) Ausiello, I., Italiano, G. F., Spaccamela, A. M. and Nanni, U.: Incremental Algorithms for Minimal Length Paths, *J. of Algorithms,* Vol. 12, No. 4, pp. 615-638 (1991).
8) Kruskal, C. P., Rudolph, L. and Snir, M.: Efficient Parallel Algorithms for Graph Problems, *Proc. 1986 Intern. Conf. on Parallel Processing,* pp. 859-876 (1986).
9) Lucas, J. M. and Sackrowitz, M. G.: Efficient Parallel Algorithms for Path Problems in Directed Graphs, *Algorithmica,* Vol. 7, No. 5/6, pp. 631-647 (1992).
10) Ma Jun and Takaoka, T.: An $O(n(n^2/p+\log n)$ Parallel Algorithm to Compute the All Pair Shortest Paths and the Transitive Closure, *J. of Information Processing, Japan,* Vol. 12, No. 2, pp. 119-124 (1989).
11) Ma Jun and Takaoka,T.: Parallel Algorithms for the All Pair Shortest Paths and Transitive Closure, *Chinese Journal of Computers,* Vol. 13, No. 9, pp. 706-709 (1990).
12) Quinn, M. J. and Deo, N.: Parallel Graph Algorithms, *ACM Computing Surveys,* Vol. 16, No. 3, pp. 319-348 (1984).
13) Floyd, R. W.: Algorithm 97, Shortest Path, *CACM,* Vol. 5, No. 6, p. 345 (1962).
14) Jenq, J. F. and Sahni, S.: All Pairs Shortest Paths on a Hyper-cube Multiprocessor, *Proc. Intern. Conf. on Parallel Processing,* pp. 713-716 (1987).
15) Paige, R. C. and Kruskal, C. P.: Parallel Algorithms for Shortest Path Problems, *1985 Intern. Conf. on Parallel Processing,* pp. 14-19 (1985).
16) Reghbati, A. E. and Cornei, D. G.: Parallel Computations in Graph Theory, *SIAM J. Comput.,* Vol. 2, No. 2, pp. 230-237 (1978).
17) Reif, J. H. and Spirakis, J.: *The Expected Time Complexity of Parallel Graph and Digraph Algorithms,* TR-11-82, Aikin Computation Lab., Harvard Univ. (1982).
18) Warshall, S.: A Theorem on Boolean Matrices, *J. ACM,* Vol. 9, No. 1, pp. 11-12 (1962).

(Received July 26, 1993)
(Accepted February 17, 1994)

Ma Jun, born in 1956, received the B.S. degree from Shandong University of China in 1982, the M. S. degree from Ibaraki University in 1988. He is an associate Professor at the Computer Science of Shandong University, his research interests include the design and analysis of algorithms, parallel and distributed algorithms and AI. He received Shandong Provence Science & Technology Award in 1993, he is the member of China Computer Society.

Tadao Takaoka is Professor of Computer Science at Ibaraki University. He joined Ibaraki Univeristy after working at an NTT Laboratory for three years. He got his Ph.D. degree from Kyoto Univeristy, and taught at University of Canterbury and University of Alabama at Birmingham. His research interests are in analysis of algorithms and program verification. He is a member of IPSJ.

Ma Shaohan, born in 1938, graduated from Shandong Univerisity of China in 1962. He was a research fellow at the Illinois University from 1985 to 1986, he is a Professor at the Computer Science of Shandong University, his research interests include the design and analysis of algorithms and AI. He received Shandong Provence Science & Technology Award in 1993, he is the member of China Computer Society.