Bull. Nov. Comp. Center, Comp. Science, 14 (2001), 29–42
© 2001 NCC Publisher

An associative version of the Edmonds–Karp–Ford shortest path algorithm[*]

A.S. Nepomniaschaya

In this paper, by means of a model of associative parallel systems with the vertical processing (the STAR-machine), we propose a natural straight forward representation of the Edmonds–Karp version of performing the Ford shortest path algorithm. We present the associative parallel algorithm as the corresponding STAR procedure and prove its correctness. Moreover, we provide special tools for maintaining graphs with the negative arc weights on associative parallel processors.

1. Introduction

Finding the shortest paths in networks is a fundamental problem in the combinatorial optimization. An important version of the shortest path problem is the single-source problem. Given a directed n-vertex and m-arc weighted graph G with a distinguished vertex s, the single-source shortest path problem is to find for each vertex v the length of the shortest path from s to v. When all arc weights are non-negative, the most efficient solution gives Dijkstra's sequential shortest path algorithm [3]. Dijkstra's algorithm runs in $O(m + n \log n)$ time both on the RAM model [9], when the priority queue is realized using the Fibonacci heap, and the EREW PRAM model [4], when the priority queue is given by means of relaxed heaps. In [14], we propose a simple straight forward implementation of Dijkstra's shortest path algorithm on a model of associative parallel systems with the vertical processing (the STAR-machine). In [7], Ford generalizes Dijkstra's algorithm for graphs having negative arc weights but without cycles of negative weight. In [5], Edmonds and Karp present a simple execution of the Ford shortest path algorithm.

In this paper, we study a basic possibility of updating graphs with the negative arc weights using associative parallel processors. Here we propose a natural straight forward representation of the Edmonds–Karp–Ford algorithm on the STAR-machine and prove its correctness. This representation uses the new basic procedures and tools for maintaining graphs with the negative arc weights. Such a technique extends the class of algorithms which can be implemented in a natural way on associative parallel processors.

[*]Partially supported by the Russian Foundation for Basic Research under Grant 99-01-00548.

2. Model of associative parallel machine

Let us recall our model which is based on a Staran-like associative parallel processor [8, 10]. We define it as an abstract STAR-machine of the SIMD type with bit-serial (or vertical) processing and simple single-bit processing elements (PEs). The model consists of the following components:

- a sequential control unit (CU), where programs and scalar constants are stored;
- an associative processing unit consisting of p single-bit PEs;
- a matrix memory for the associative processing unit.

The CU broadcasts an instruction to all the PEs in unit time. All active PEs execute it in parallel while inactive PEs do not perform it. Activation of a PE depends on the data. It should be noted that the time of performing any instruction does not depend on the number of processing elements [8].

Input binary data are loaded in the matrix memory in the form of two-dimensional tables in which each datum occupies an individual row and it is updated by a dedicated processing element. It is assumed that there are more PEs than data. The rows are numbered from top to bottom and the columns – from left to right. Both a row and a column can be easily accessed. Some tables may be loaded in the matrix memory.

The associative processing unit is represented as h vertical registers ($h \geq 4$), each consisting of p bits. The vertical registers can be regarded as a one-column array. The bit columns of the tabular data are stored in the registers which perform the necessary Boolean operations and record the search results.

The STAR-machine run is described by means of the language STAR [11] which is an extension of Pascal. Let us briefly consider the STAR constructions needed for the paper. To simulate data processing in the matrix memory, we use data types **word, slice**, and **table**. Constants for the types **slice** and **word** are represented as a sequence of symbols of $\{0,1\}$ enclosed within single quotation marks. The types **slice** and **word** are used for the bit column access and the bit row access, respectively, and the type **table** is used for defining the tabular data. Assume that any variable of the type **slice** consists of p components which belong to $\{0,1\}$. For simplicity, let us call *slice* any variable of the type **slice**.

Now, we present some elementary operations and predicates for slices.

Let X, Y be variables of the type **slice** and i be a variable of the type **integer**. We use the following operations:

SET(Y) sets all the components of Y to $'1'$;

CLR(Y) sets all the components of Y to $'0'$;

$Y(i)$	selects the i-th component of Y;
FND(Y)	returns the ordinal number i of the first (or the uppermost) component $'1'$ of Y, $i \geq 0$.

In the usual way we introduce the predicates ZERO(Y) and SOME(Y) and the bitwise Boolean operations X *and* Y, X *or* Y, *not* Y, X *xor* Y.

All the operations for the type **slice** can also be performed for the type **word**.

For a variable T of the type **table**, we use the following elementary operations:

ROW(i,T)	returns the i-th column of the matrix T;
COL(i,T)	returns the i-th column of the matrix T.

Remark 1. Note that the STAR statements [11] are defined in the same manner as for Pascal. We will use them later for presenting our procedures.

3. Preliminaries

First, let us present some notions used in the paper.

Let $G = (V, E, w)$ be a *directed weighted graph* with the set of vertices $V = \{1, 2, \ldots, n\}$, the set of directed edges (arcs) $E \subseteq V \times V$ and the function w that assigns a weight to every edge. We assume that $|V| = n$ and $|E| = m$.

We consider graphs which have no self-loops and parallel arcs.

A *weight matrix* of G is an $n \times n$ matrix which contains arc weights as elements. We assume that $w(u, v) = \infty$ if $(u, v) \notin E$.

Note that the weights are integers represented as binary strings.

A *path* from the vertex u to the vertex v in G is a finite sequence of the vertices $u = v_1, v_2, \ldots, v_k = v$, where $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, k-1$ and $k > 0$. The *shortest path between two vertices* in a weighted graph is the path with the minimal sum of weights of its arcs.

A *tree of the shortest paths* T is a connected acyclic subgraph of G with the root vertex s which contains all vertices of G and such that the path from s to any vertex v in T is the shortest path from s to v in G.

Now, recall a group of the basic procedures implemented on the STAR-machine which will be used later on. These procedures use the given global slice X to select by ones the row positions being used in the corresponding procedure.

The procedure MATCH(T, X, v, Z) defines positions of those rows of the given matrix T which coincide with the given pattern v written in binary code. It returns the slice Z, where $Z(i) = '1'$ if and only if ROW$(i, T) = v$ and $X(i) = '1'$.

The procedure MIN(T, X, Z) defines positions of those rows of the given matrix T where the minimal element is located. It returns the slice Z, where $Z(i) = '1'$ if and only if ROW(i, T) is the minimal matrix element and $X(i) = '1'$.

The procedure MAX(T, X, Z) is defined by analogy with the procedure MIN(T, X, Z).

It should be noted that the procedures MATCH, MIN and MAX are based on the corresponding algorithms first examined in [6].

The procedure SETMIN$(T, R, X, Y1)$ defines positions of the matrix T rows being less than the corresponding rows of the matrix R. It returns the slice $Y1$, where $Y1(j) = '1'$ if and only if ROW$(j, T) <$ ROW(j, R) and $X(j) = '1'$.

The procedure ADDV(T, R, X, F) writes into the matrix F the result of adding the matrices T and R having the same number of bit columns. Note that this procedure is based on the corresponding algorithm from [8].

The procedure ADDC(T, X, v, F) adds the binary word v to those rows of the matrix T which are selected by ones in X, and writes down the result into the corresponding rows of the matrix F. The rows of F, which are selected by zeros in X, will be set to zero.

The procedure SUBTV(T, R, X, S) writes into the matrix S the result of subtracting the rows of the matrix R from the corresponding rows of the matrix T selected by ones in the slice X.

The procedure TMERGE(T, X, F) writes into the matrix F those rows of the given matrix T, which are selected by ones in the slice X. Other rows of the matrix F are not changed.

The procedure WCOPY(v, X, F) writes the binary word v into those rows of the matrix F, which are selected by ones in the slice X. The rows of the matrix F, which are selected by zeros in the slice X, will consist of zeros.

The procedure TCOPY1(T, j, h, F) writes h columns from the given matrix T, starting from the $(1 + (j - 1)h)$-th column, into the result matrix F, where $j \geq 1$.

In [12, 13], we have shown that the basic procedures take $O(k)$ time each, where k is the number of bit columns in the corresponding matrix.

4. Basic procedures for updating negative integers

In this section, we propose a group of new basic procedures being applied to matrices with the negative integers. Therefore, every matrix is given along with its slice of signs. These procedures permit extending the class of graph algorithms which can be implemented in a natural way on associative parallel systems with the vertical data processing.

We first consider the procedure MIN$^*(T, X, Y, Z)$ which generalizes the basic procedure MIN(T, X, Z). The slice Y is used to save the signs of the matrix T. The procedure MIN* runs as follows.

```
proc MIN*(T: table; X,Y: slice; var Z: slice);
var X1: slice;
begin X1:= X and Y;
/* By means of ones in the slice X1, we save positions of the matrix T rows
   having negative value. */
  if SOME(X1) then MAX(T,X1,Z) else MIN(T,X,Z)
end;
```

Let us present the procedure ADDV$^*(T, R, X, Y, Z, F, Z1)$ which generalizes the basic procedure ADDV(T, R, X, F). The procedure ADDV* performs in parallel the algebraic addition of the matrices T and R. It uses the slices Y, Z and $Z1$ to save the signs of the matrices T, R, and F, respectively. This procedure runs as follows.

```
proc ADDV*(T,R: table; X,Y,Z: slice; var F: table; Z1: slice);
/* Here X is the global slice. */
var X1,X2,Y1,Y2: slice; M: table;
begin X1:= Y xor Z;
/* By means of ones in the slice X1, we save positions of the corresponding
   rows in T and R having different signs. */
  Y1:= not X1;
  Y1:= Y1 and X;
/* By means of ones in Y1, we select the positions of the corresponding
   rows in T and R having the same sign. */
  ADDV(T,R,Y1,F);
  Z1:= Y1 and Y;
/* The results of adding the corresponding rows with the same sign
   in T and R are written in the corresponding rows of F
   and their signs are saved in the slice Z1. */
  X1:= X1 and X;
  SETMIN(T,R,X1,X2);
/* By means of ones in X2, we select positions of the rows in T
   which are less than the corresponding rows in R. */
  SUBTV(R,T,X2,M);
  TMERGE(M,X2,F);
/* The rows of the matrix F, selected by ones in X2, save the result
   of subtracting the rows of T from the corresponding rows of R. */
  Y1:= Z and X2;
  Z1:= Z1 or Y1;
/* The signs of the matrix R rows, selected by ones in X2,
   are included into the slice Z1. */
  Y2:= X1 and (not X2);
/* By means of ones in Y2, we select positions of the matrix T rows, which are
   greater than the corresponding rows of R. */
  SUBTV(T,R,Y2,M);
  TMERGE(M,Y2,F);
/* The rows of the matrix F, selected by ones in Y2, save the result
   of subtracting the rows of R from the corresponding rows of T. */
  X1:= Y and Y2;
  Z1:= Z1 or X1
/* The signs of the matrix T rows, selected by ones in Y2,
   are included into the slice Z1. */
end;
```

Now, we present the procedure ADDC*$(T, X, Y, v, sign, F, Z)$ which generalizes the procedure ADDC(T, X, v, F). It uses the slices Y and Z to save the signs of the given matrix T and the resulting matrix F, respectively, and the variable *sign* to indicate the sign of the given v. The procedure ADDC* runs as follows.

```
proc ADDC*(T: table; X,Y: slice; v,sign: word;
           var F: table; Z: slice);
/* Here X is the global slice. */
var X1,Z1: slice; R: table;
begin CLR(X1); WCOPY(v,X,R);
  if sign='0' then Z1:=X1 else Z1:=X;
  ADDV*(T,R,X,Y,Z1,F,Z)
end;
```

Let us consider the procedure SETMIN*$(T, R, X, Y, Z, Z1, Z2)$ which generalizes the procedure SETMIN$(T, R, X, Z1)$. It uses the slices Y and Z to select by ones positions of the negative integers in the given matrices T and R, respectively. Along with the slice $Z1$, this procedure returns the slice $Z2$ to select by ones positions of those matrix T rows, in which the negative integers are less than the negative ones located in the corresponding rows of R.

The procedure SETMIN* runs as follows.

```
proc SETMIN*(T,R: table; X,Y,Z: slice; var Z1,Z2: slice);
var X1,X2,X3: slice;
```

```
begin X1:= Y and Z;
  X1:= X1 and X;
/* By means of ones in $X1$, we select positions of the rows of $T$ and $R$,
   where the negative integers are written. */
  X2:= (not Y) and (not Z);
  X2:= X2 and X;
/* By means of ones in $X2$, we select positions of the rows of $T$ and $R$,
   where the positive integers are written. */
  X3:= X1 or X2;
/* By means of ones in $X3$, we select positions of the rows of $T$ and $R$,
   where the corresponding integers have the same sign. */
  SETMIN(T,R,X3,Z1);
/* By means of ones in $Z1$, we select positions of the rows of $T$,
   where the integers less than the corresponding integers from $R$
   and they have the same sign. */
  Z2:= X1 and (not Z1);
/* By means of ones in $Z2$, we select positions of the matrix $T$ rows,
   where $\mathrm{ROW}(i,T) < \mathrm{ROW}(i,R)$ and $Y(i) = Z(i) = '1'$. */
  Z1:= Z1 and X2
/* By means of ones in $Z1$, we select positions of the matrix $T$ rows,
   where $\mathrm{ROW}(i,T) < \mathrm{ROW}(i,R)$ and $Y(i) = Z(i) = '0'$. */
end;
```

Correctness of the procedures ADDV* and SETMIN* is verified by induction on the number of columns in the matrix T. Correctness of the procedures ADDC* and MIN* is evident. It is not difficult to observe that these procedures take $O(k)$ time each, where k is the number of columns in the matrix T.

5. Representation of the Edmonds–Karp–Ford algorithm on the STAR-machine

In this section, we propose a natural straight forward representation of Edmonds–Karp–Ford algorithm on the STAR-machine. Explain the main idea of this algorithm.

For any vertex $v \in V$ we have a superdistance $D(v) \geq \mathrm{dist}(s, v)$, where $\mathrm{dist}(s, v)$ is the *distance*, that is, the weight of the shortest path from the source vertex s to the vertex v. In addition, we have a set of vertices $S \subseteq V$ belonging to the tree of the shortest paths, that is, $\forall u \in S$ we have $D(u) = \mathrm{dist}(s, u)$. Initially, $S = \{s\}$, $D(s) = 0$ and $\forall v \notin S$ $D(v) = \infty$. Let S consist of k vertices $(1 \leq k < n)$ and u be the last vertex added to the set S. Then a new vertex for the set S is defined as follows.

At first, for all $v_i \in V$, we define the arcs (u, v_i). Then, for every vertex $v_i \in V$ we determine the superdistance $D(v_i)$ as $\mathrm{dist}(s, u) + w(u, v_i)$. After that, we exclude those vertices v_j from the set S, for which the current superdistance $D(v_j)$ is less than the previous one. Finally, among the vertices $v_k \notin S$, we select such a vertex v_r that has the minimal superdistance, and add it to the set S.

The process of including the vertices in the set S is completed when $S = V$ and no superdistance $D(x)$ changes.

Consider the main idea of the associative version of the Edmonds–Karp–Ford algorithm.

We save positions of the vertices included in the tree of the shortest paths S. Let k be the last vertex added to S. Then, we define *positions* of all the vertices which are adjacent to k. In the matrix M, we compute the superdistances for the vertices which are adjacent to k. After that, we define *positions* of those vertices whose superdistances have decreased.

Now, we write the new superdistances to the matrix D and remove from S positions of the vertices, whose superdistances have decreased.

Finally, among the vertices not belonging to S, we define the position of the vertex v_p whose superdistance $D(v_p)$ has the minimal value. We include this vertex in the set S.

Remark 2. Note that any current superdistance decreases in the following cases:

- the current superdistance is less than the previous one and both values are positive;
- the current superdistance is greater than the previous one and both values are negative;
- the current superdistance is negative and the previous one is positive.

We assume that $x + \infty = \infty$ and $\min(x, \infty) = x$ for all x. We choose infinity as $\sum_{i=1}^{n} w_i$, where w_i is the maximal weight of arcs incident with vertex i. Let h be the number of bits necessary for coding infinity. Then the weight matrix W consists of hn bit columns and every i-th vertex of the graph G is associated with the i-th *field* having h bit columns.

Remark 3. In view of the vertical data processing, we assume that every graph will be represented in the STAR-machine memory as a transpose weight matrix. Note that some real associative parallel processors allow one to easily transpose every matrix.

We will represent the Edmonds–Karp–Ford algorithm as procedure EKF written in the language STAR. It uses the following input parameters:

- a graph given as transpose weight matrix T and an $n \times n$ matrix Q for indicating the signs of the corresponding weights from T;
- the source vertex s;
- the number of bits h for coding infinity;
- the binary word *inf* for representing infinity.

The procedure returns the distance matrix D and the slice Z for indicating the signs of the corresponding distances from D.

Note that for any i, the weights of arcs, which are incident with the vertex i, are written in the i-th column of T. The distance from the source vertex s to the given vertex i is written in the i-th row of D. The negative weights in the matrix T are indicated by means of ones in the matrix Q, while the negative distances in the matrix D are indicated with the use of ones in the slice Z.

Let us briefly explain the meaning of the main variables being used.

The procedure uses a global slice U where positions of vertices, belonging to the tree of the shortest paths S, are selected by zeros; an integer k – to save the last vertex added to S; a matrix R – to select the weights of the arcs which are incident with the last vertex added to S; a matrix M – to save the current superdistances for the vertices being adjacent to the vertex k.

At the beginning, the slice Z consists of zeros and the tree of the shortest paths S consists of the source vertex s.

Now, we present the procedure EKF.

```
   proc EKF(T,Q: table; s,h: integer; inf: word; var D: table;
            Z: slice);
   var M,R: table; U,V,X,Y,Z1,Z2: slice;
       w,sign: word; k: integer;
1. begin k:=s; SET(U); U(s):='0';
   /* By means of zeros in U, we save positions of vertices
      belonging to the tree of the shortest paths S. */
2.   WCOPY(inf,U,D);
3.   CLR(Z);
   /* By means of ones in Z, we select positions of vertices
      whose superdistances from the vertex s are negative. */
4.   while SOME(U) do
5.     begin TCOPY1(T,k,h,R);
   /* Here R is a matrix of arc weights of those vertices
      which are adjacent to the vertex k. */
6.       SET(X); X(k):='0';
7.       MATCH(R,X,inf,Y);
```

```
8.        X:= X and (not Y);
    /* Positions of vertices which are adjacent to $k$ are selected
       by ones in the slice $X$. */
9.        w:=ROW(k,D);
10.       Y:=COL(k,Q);
11.       sign:=Z(k);
12.       ADDC*(R,X,Y,w,sign,M,V);
    /* By means of ones in the slice $V$, we select the positions
       of vertices whose current superdistances are negative. */
13.       SETMIN*(M,D,X,V,Z,Z1,Z2);
    /* By means of ones in $Z1$ (respectively, $Z2$), we select positions
       of the vertices whose current superdistances are less than the previous ones
       and both superdistances are positive (respectively, negative). */
14.       V:= V and (not Z);
    /* By means of ones in $V$, we select positions of the vertices
       whose current superdistance is negative and the previous one is positive. */
15.       Z1:= Z1 or Z2;
16.       Z1:= Z1 or V;
    /* By means of ones in $Z1$, we select positions of the vertices
       whose current superdistance is less than the previous one. */
17.       TMERGE(M,Z1,D);
18.       Z:= Z or V;
    /* By means of ones in $Z$, we select positions of the vertices
       whose current superdistances are negative. */
19.       U:= U or Z1;
    /* We delete positions of those vertices in the slice $U$,
       whose current superdistances have been decreased. */
20.       MIN*(D,U,Z,X);
21.       k:=FND(X);
    /* Here $k$ is the position of a vertex added to the set $S$. */
22.       U(k):='0'
23.     end;
24. end.
```

Theorem. *Let a directed weighted graph G be given as the transpose weight matrix T and the $n \times n$ matrix Q which saves by ones the corresponding negative weights from T. Let s be the source vertex, and there is no directed cycle from s with the negative weight. Let every arc weight use h bits and let inf be the binary representation of the infinity. Then the procedure $EKF(T, Q, s, h, inf, D, Z)$ returns the distance matrix D, in whose every i-th row there is the distance from s to i, and the slice Z which saves by ones the negative distances.*

To prove this theorem, we will use the following lemmas:

Lemma 1. *Let all assumptions of the theorem be true. Let k be the last vertex added to the tree of the shortest paths S. Then after performing lines 1–12 of the procedure EKF, the current superdistances are written in the matrix M for the vertices being adjacent to k and positions of vertices, whose current superdistances are negative, are selected by ones in the slice V.*

Proof. Obviously, after performing line 1, the variable k saves the source vertex s and only this vertex is included in the set S. After performing lines 2–3, the distance from s to s is equal to zero, the distances from s to other vertices have not been determined yet and all values are non-negative in the matrix D. According to the Edmonds–Karp–Ford algorithm, it is necessary to define *all the vertices* which are adjacent to the vertex k. Therefore, after performing lines 6–7, there is the unique zero in the slice X (in its k-th position) and positions of vertices, which are *not adjacent* to k, are selected by ones in the slice Y, that is, for every i, $Y(i) = '1'$ if and only if $X(i) = '1'$ and $\mathrm{ROW}(i, R) = \mathit{inf}$. In view of defining the procedure MATCH after performing the statement $X := X \; \mathit{and} \; (\mathit{not} \, Y)$ (line 8), we obtain $X(i) = '1'$ if and only if $Y(i) = '0'$, that is, positions of the vertices being adjacent to k are selected by ones in the slice X. Clearly, after performing line 9, the variable w saves the k-th row of the matrix D. Because of defining the matrix Q after performing the statement $Y := \mathrm{COL}(k, Q)$ (line 10), positions of the vertices being formed with k arcs of the negative weights, are selected by ones in the slice Y. Obviously, after performing line 11, we save the sign of the k-th row of the matrix D. As a result of executing the basic procedure ADDC* (line 12), the current superdistances for the vertices being adjacent to the vertex k are written in the matrix M, positions of the vertices whose current superdistances are negative, are selected by ones in the slice V. □

Lemma 2. *Let all assumptions of the theorem be true. Let k be the last vertex added to the tree of the shortest paths S. Then, after fulfilling lines 13–18, positions of the vertices, for which the current superdistances decrease, are selected by ones in the slice Y. Moreover, the rows of the matrix M, selected by ones in $Z1$, are written in the corresponding rows of the matrix D and positions of vertices, whose current superdistances are negative, are selected by ones in the slice Z.*

Proof. In view of Lemma 1, after performing lines 1–12, the current superdistances for the vertices being adjacent to k are written in the matrix M, and positions of the vertices, whose current superdistances are negative, are selected by ones in V. In account of Remark 2, we have to analyze three cases to select positions of the vertices whose current superdistances from s have decreased.

As a result of fulfilling the basic procedure SETMIN* (line 13), by means of ones in $Z1$, we select positions of the vertices whose current superdistances are less than the previous ones and both superdistances are positive, and by means of ones in $Z2$, we select positions of the vertices whose current superdistances are less than the previous ones and both superdistances are negative. Therefore, it remains to define the positions of the vertices whose current superdistances are negative and the corresponding previous ones are positive. We determine them by means of the statement $V := V\, and\, (not\, Z)$ (line 14). Clearly, after performing lines 15–16 by means of ones in the slice $Z1$, we accumulate positions of the vertices whose superdistances have decreased at the current iteration. On fulfilling the statement TMERGE$(M, Z1, D)$ (line 17), the rows of the matrix M, selected by ones in $Z1$, are written in the corresponding rows of the matrix D.

It remains to check that all current negative superdistances are selected by ones in the slice Z. Really, initially the slice Z consists of zeros (line 3). Each time after performing the statement $Z := Z\, or\, V$ (line 18), we increase the number of ones in the slice Z which select positions of the vertices having the negative superdistances from s because any negative superdistance in the matrix D can be replaced with a new negative one. $\square$

Remark 4. One can immediately verify that after performing the statement $U := U\ or\ Z1$ (line 19), we have $U(k) = '0'$.

Lemma 3. *Let all assumptions of the theorem be true. Let k be the last vertex added to the tree of the shortest paths S. Then, after fulfilling lines 19–23, the vertices, whose current superdistances decrease, are deleted from S. Moreover, a new k-th vertex will be included into the tree of the shortest paths S.*

Proof. By Lemma 2, after fulfilling lines 13–18, positions of the vertices whose current superdistances decrease are saved by ones in $Z1$. Therefore after performing the statement $U := U\ or\ Z1$ (line 19), these vertices are deleted from S, because their positions are selected by ones in the slice U. Since some superdistances may be negative in D, we perform the basic procedure MIN*(D, U, Z, X) (line 20) to define positions of the rows having the minimal value. After performing lines 21–22, a new k-th vertex is included in the tree of the shortest paths S. $\square$

Remark 5. One can immediately verify that after performing the first iteration of the procedure EKF for every vertex i being adjacent to s, the weight of the arc (s, i) is written in the i-th row of the matrix D.

Now we explain briefly the proof of the theorem.

Sketch of the proof. We prove by induction on the number of arcs q included in the shortest path from s to any vertex of the graph G.

Basis is verified for $q = 1$, that is, the shortest path for such vertices is the corresponding arc being incident with the vertex s. Let an arc (s, j) be the shortest path from s to the vertex j. Then, in view of Remark 5, after performing the first iteration, $w(s, j)$ is written in the j-th row of the matrix D. By Lemma 2, the j-th row of the matrix D does not change during execution of the procedure EKF, since every other path from s to j is greater than $w(s, j)$.

Step of induction. We assume that the statement is true for all the shortest paths having less than q arcs. Let γ be the shortest path from s to j and let γ consist of q arcs. Assume that $\gamma = \gamma_1\gamma_2$, where $\gamma_2 = (v, j)$. In view of Lemmas 1–3, without loss of generality, it is sufficient to consider the iterations when all the vertices in the path γ_1 are not excluded further from the tree of the shortest paths S. By the induction hypothesis, the statement is true for the path γ_1. Since the vertex j is adjacent to v and γ is the shortest path from s to j, the length of the shortest path γ is written in the j-th row of the matrix D in view of Lemmas 1–3. By Lemma 2, in every next iteration the j-th row of the matrix D does not change. Hence, after performing the procedure EKF, the length of the shortest path from s to j is written in the j-th row of the matrix D. □

6. Conclusions

We have presented an associative version of the Edmonds–Karp–Ford shortest path algorithm as the procedure EKF and proved its correctness. This procedure performs in parallel all steps of the Edmonds–Karp–Ford algorithm using a group of the new basic procedures for updating the negative integers. These procedures permit one to extend the class of graph algorithms which can be implemented in a natural way on the associative parallel processors. We are planning to design an associative version of the Bellman–Ford shortest path algorithm [1, 2] being the best one to update graphs with the negative arc weights. After that, we will compare these representations.

References

[1] Bellman R. On a routing problem // Quarterly of Applied Mathematics. – 1958. – Vol. 16, № 1. – P. 87–90.

[2] Christofides N. Graph Theory. An Algorithmic Approach. – New York: Academic Press, 1975.

[3] Dijkstra E.W. A note on two problems in connection with graphs // Numerische Mathematik. – 1959. – Vol. 1. – P. 269–271.

[4] Driscoll J.R., Gabow H.N., Shrairman Ruth, Tarjan R.E. Relaxed heaps: an alternative to Fibonacci heaps with applications to parallel computation // Communications of the ACM. – 1988. – Vol. 31, № 11. – P. 1343–1354.

[5] Edmonds J., Karp R.M., Theoretical improvements in the algorithmic efficiency for network flow problems // J. ACM. – 1972. – Vol. 19, № 2. – P. 248–264.

[6] Falkoff A.D. Algorithms for parallel–search memories // J. ACM. – 1962. – Vol. 9, № 10. – P. 488–510.

[7] Ford L.R. Network Flow Theory. – Rand Corporation Report P-923, 1956.

[8] Foster C.C. Content Addressable Parallel Processors. – New York: Van Nostrand Reinhold Company, 1976.

[9] Fredman M.L., Tarjan R.E. Fibonacci heaps and their uses in improved network optimization algorithms // J. ACM. – 1987. – Vol. 34, № 3. – P. 596–615.

[10] Mirenkov N. The siberian approach for an open-system high-performance computing architecture // Computing and Control Engineering Journal. – 1992. – Vol. 3, № 3. – P. 137–142.

[11] Nepomniaschaya A.S. Language STAR for associative and parallel computation with vertical data processing // Proc. Intern. Conf. "Parallel Computing Technologies". – Singapure: World Scientific, 1991. – P. 258–265.

[12] Nepomniaschaya A.S. An associative version of the Prim–Dijkstra algorithm and its application to some graph problems // Andrei Ershov Second Intern. Memorial Conf. "Perspectives of System Informatics" / Lecture Notes in Computer Science. – Berlin: Springer-Verlag, 1996. – Vol. 1181. – P. 203–213.

[13] Nepomniaschaya A.S. Solution of path problems using associative parallel processors // Proc. Intern. Conf. on Parallel and Distributed Systems, IEEE Computer Society Press, ICPADS'97. – Korea. Seoul, 1997. – P. 610–617.

[14] Nepomniaschaya A.S., Dvoskina M.A. A simple implementation of Dijkstra's shortest path algorithm on associative parallel processors // Fundamenta Informaticae. – IOS Press, 2000. – Vol. 43. – P. 227–243.