Joint NCC & IIS Bull., Comp. Science, 11 (1999), 45–58

An associative algorithm for finding a maximum-weight cycle in directed graphs

A. S. Nepomniaschaya

In this paper we propose a novel associative parallel algorithm for selecting a directed cycle of the maximal weight on an abstract model of the SIMD type with vertical data processing (the STAR–machine). This algorithm is represented as the corresponding STAR procedure whose correctness is verified and time complexity is evaluated.

1. Introduction

The problem of selecting a cycle of the maximal weight in a directed graph arises when performing Edmonds' algorithm for finding optimum branchings [2]. This algorithm has important applications to graph theory and combinatorial optimization. In particular, it is used for finding a minimal spanning tree in a directed graph. Edmonds' algorithm takes polynomial time on conventional sequential computers. Therefore in [1, 4, 5, 16] different algorithms are devised for improving the time bound. Among them, on the one hand, is a very complicated algorithm [4]. On the other hand, there is an algorithm [5] which uses a special data structure, the Fibonacci heap.

To obtain a natural and relatively simple implementation of Edmonds' algorithm one can utilize associative parallel processors which are mainly oriented to solving non–numerical problems. This class of parallel computers includes the well–known systems STARAN, DAP, MPP, and CM–2. Such an architecture provides a massively parallel search by contents and processing of unordered data represented in the form of two–dimensional tables [15].

Let us enumerate graph problems being represented on associative parallel processors. In [7, 8, 12, 14, 15], the problem of finding a minimal spanning tree is studied for different graph representation forms using different algorithms and different formal models. In [9], the associative version of Gabow's algorithm for finding the smallest spanning trees with a degree constraint is represented on the STAR–machine which is a model of associative parallel processors with vertical data processing. Problems of finding connected components and the transitive closure of a graph, verifying an articulation point and verifying a bridge are considered for undirected unweighted graphs

represented as an adjacency matrix both on the orthogonal machine [14] and the STAR–machine [10]. Algorithms for solving two shortest path problems are examined both on the associative array processor LUCAS [3] and on the STAR–machine [10,12].

In this paper, we propose a novel associative parallel algorithm for selecting a cycle of the maximal weight in a directed weighted graph represented on the STAR–machine as a list of triples (edge vertices and the weight). For this purpose a special construction with a trap is used to save positions of those arcs which cannot be included in a directed cycle. The algorithm is represented as the corresponding STAR procedure whose correctness is proved. We have obtained that this procedure takes $O(n \log n)$ time, where n is the number of graph vertices.

2. Model of Associative Parallel Machine

The model is defined as an abstract STAR–machine of the SIMD type with bit–serial (or vertical) processing. It consists of the following components:

– a sequential control unit where programs and scalar constants are stored;

– an associative processing unit consisting of p single–bit processing elements (PEs);

– a matrix memory for the associative processing unit.

The 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 has a dedicated processing element. The rows are numbered from top to bottom and the columns from left to right. A row (word) or a column (slice) may be accessed with equal ease. Some tables may be loaded in the matrix memory.

The associative processing unit is represented as h vertical registers each consisting of p bits ($h \geq 4$). A vertical register 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 [13] which is an extension of Pascal. Consider briefly the STAR constructions needed for the paper. To simulate data processing in the matrix memory, we use three new 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 apostrophes. We use the types **slice** and **word** for bit column access and bit row access, respectively, and the type **table** for a definition of the tabular data. We assume that any variable of the type **slice** consists of p components belonging to $\{0, 1\}$.

Recall some operations and predicates for slices.

Let X, Y be variables of the type **slice** and i be a variable of the type **integer**. We define 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, $i \geq 0$; STEP(Y) returns the same result as FND(Y) and then resets the first component '1'.

We utilize the bitwise Boolean operations X *and* Y, X *or* Y, *not* Y, X *xor* Y and predicates ZERO(Y) and SOME(Y) which are introduced in the obvious way.

For a variable T of the type **table** we use the following two operations: ROW(i, T) returns the i-th row of the matrix T; COL(i, T) returns its i-th column.

3. Preliminaries

At first, let us recall some notions being 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 will denote $|V|$ by n and $|E|$ by m. An arc $e = (u, v)$ directed from u to v has *head* v and *tail* u. We will also use the notations $h(e) = v$ and $t(e) = u$.

In the STAR–machine matrix memory, a directed weighted graph is represented as association of the matrices *left*, *right* and *weight* where each arc $(u, v) \in E$ is matched with the triple $< u, v, w(u, v) >$. Recall that vertices and weights are written in binary code.

The following notions are borrowed from [6].

A set $C \subseteq E$ is called a *cycle* if $C = \{e_1, e_2, \ldots, e_k\}$, where $t(e_i) = h(e_{i+1})$, $i = 1, 2, \ldots, k-1$ and $t(e_k) = h(e_1)$. An arc e is called *critical* if no arc with the same head has a greater positive weight than e. A *critical cycle* is a cycle with all the arcs being critical.

A set $B \subseteq E$ is called a *branching* if B does not contain a cycle and every pair of distinct arcs in B has different heads. In other words, no two arcs of B are directed to the same vertex. A branching of maximal weight is called *optimum*.

Let C be a critical cycle in G. Following [5], the notation G/C will be used when the cycle C is contracted in G.

The *reduced graph* for a critical cycle C is the graph G/C with the weight function of G modified as follows. Let e be an arc of G with $t(e) \notin C$ and $h(e) \in C$, and let e' be an arc of C with $h(e') = h(e)$. Let e'' be an arc of C having the minimal weight. Then in the reduced graph the new weight of

e is defined as $w(e) + w(e'') - w(e')$. All other arcs of G/C do not change their weights.

Now, let us recall three basic procedures [11,12] which use the given global slice X to indicate by $'1'$ the row positions being used in the corresponding procedure.

The procedure MATCH(T, X, v, Z) defines the row positions in the given matrix T coinciding with the given v. Its result is the slice Z in which $Z(i) =' 1'$ if ROW$(i, T) = v$ and $X(i) =' 1'$. The procedure MAX(T, X, Z) defines the row positions in the given matrix T where the maximal element is located. It returns the slice Z in which $Z(i) =' 1'$ if ROW(i, T) is the maximal matrix element and $X(i) =' 1'$. The procedure HIT(T, F, X, Z) defines positions of the corresponding identical rows in the given matrices T and F using the global slice X. It returns the slice Z in which $Z(i) =' 1'$ if ROW$(i, T) =$ ROW(i, F) and $X(i) =' 1'$.

In [11,12], we have shown that any basic procedure takes $O(k)$ time, where k is the number of bit columns of the corresponding matrix.

4. Finding a Critical Cycle in a Graph

In this section we present an associative algorithm for selecting a critical cycle. It will be used to perform the first phase of Edmonds' algorithm for finding optimum branchings.

The *first phase* of Edmonds' algorithm selects critical arcs. It begins with an empty set of arcs. For every vertex v being a head of an arc, a critical arc (u, v) is selected and added to this set. If (u, v) forms a critical cycle C together with other selected arcs, this cycle is contracted to form a new vertex v^* and the reduced graph G/C for C is obtained. In this graph every arc with the endpoint in C has the endpoint replaced with v^*. Moreover, the weights of all the arcs entering the cycle C are redefined as described above. The process of selecting and reducing critical cycles is being continued until all the vertices of the graph are contracted into a single vertex or none of new critical cycles can be extracted.

The *second phase* of Edmonds' algorithm discards some redundant critical arcs. It consists of expanding the nested cycles in the reverse order of their contraction and deleting one arc from each cycle with the use of a special rule.

The associative algorithm for selecting a critical cycle will be given as a procedure CYCLE written in the language STAR. It uses the following input parameters:

– a graph represented as association of the matrices *left*, *right* and *weight*;

– a list of the graph vertices given as a matrix *node* in whose i-th row there is a binary code of the vertex i, $1 \leq i \leq n$.

The procedure returns two slices X and W. The slice X is used to indicate by $'1'$ the positions of arcs belonging to the critical cycle, while W is used for indicating the positions of all the critical arcs selected during the procedure run.

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

The procedure uses a global slice D for the matrices *left, right*, and *weight*; a global slice Y for the matrix *node*; an integer t for saving the time when any critical arc is selected; two arrays A and *tag* for defining a one–to–one correspondence between the position of any critical arc and the time of its selection; a slice *trap* for indicating positions of those critical arcs which cannot be included into the critical cycle.

Initially the time t is equal to zero, each of the slices X and W consists of zeros while each of the slices D, Y, and *trap* consists of ones.

The associative algorithm for selecting a critical cycle consists of two phases. Informally it runs as follows.

The first phase includes the following three steps.

At *the first step*, by means of STEP(Y), we select the vertex (say, v) in the matrix *node* which corresponds to the position of the first component $'1'$ in the slice Y and for which there exists an incoming arc in G. This step yields a slice (Z) where the positions of all arcs with the *head* v are indicated by $'1'$. Then we go to the second step.

At *the second step* we select a position i of a critical arc (say, e) with the *head* v. This position is indicated by $'1'$ in the slice W. After recording the current time t, we set a one-to-one correspondence of the form $i \leftrightarrow t$ between the position of the critical arc e and the time of its selection as follows. In the t-th component of the array A we write the integer i, while in the i-th component of the array *tag* we write the integer t. After deleting the position i from the global slice D we go to the third step.

At *the third step* we select the tail u of the critical arc e from the i-th position. The following two cases are possible.

Case 1. The vertex u has not been processed yet. Then, after defining its position k in the matrix *node*, we perform the statement $Y(k) :=' 0'$. If u has an incoming edge, we go to the second step to select the next critical arc. Otherwise, we write $'0'$ in the i-th component of the slice *trap* to indicate the position of the critical arc e. Such an arc cannot be included in the critical cycle because its tail has no incoming edge.

Case 2. The vertex u has already been processed. Then, by means of the matrix *right* and the slice W, we define the position j of the critical arc (say, e') which enters the vertex u. If u has no incoming edge, the position of e is saved in the slice *trap* by analogy with Case 1, and we go to the first step. Otherwise, by means of the statement $s := tag[j]$, we define the time s of including the position of e' to W. By means of the array A in the slice X we indicate by $'1'$ the positions of all the critical arcs which have been

included in W beginning with the time s until the current time t. Then we go to the second phase.

In **the second phase** we check whether there is a critical arc whose position is indicated by '1' in the slice X and by '0' in the slice *trap*. If there is such an arc, we set zeros in the slice X and jump to the first step. Otherwise, we verify whether the head of the arc selected at the time t has an outgoing edge whose position is indicated by '1' in X and the tail of the arc selected at the time s has an incoming edge whose position is also indicated by '1' in X. If these conditions are true, the algorithm terminates and the arcs whose positions are indicated by '1' in the slice X form a critical cycle.

Obviously every vertex included in a directed cycle has both an incoming and outgoing edge. Two last conditions of the second phase allow one to eliminate the cases when a sequence of arcs do not form a directed critical cycle, however, their positions are indicated by '1' in the slice *trap*. Such cases are examined in the proof of Theorem 2.

Remark 1. Before performing the first phase, the positions of arcs forming a loop are defined as the positions of the corresponding coinciding rows of the matrices *left* and *right* with the use of the basic procedure HIT. These positions will not be further considered.

Now, consider the procedure CYCLE.

```
proc CYCLE(left, right, weight, node: table; var X, W: slice(left));
/* We will use the global variable m of the type integer. */
label 1, 2, 3;
var i, k, l, r, s, t: integer;
  Q, Y: slice(node); u, v: word;
  F, D, P, Z, trap: slice(left);
  tag, A: array [1 : m] of integer;
/* The first phase. */
1.  Begin t := 0; CLR(W); CLR(X); SET(D);
2.    SET(trap); SET(Y);
3.    HIT(left, right, D, P); D := D and (not P);
  /* Positions of arcs forming a loop are indicated by '1'
     in P. */
4.    while SOME(Y) do
5.      begin k :=STEP(Y); v :=ROW(k, node);
6.        MATCH(right, D, v, Z);
  /* In the slice Z we indicate by '1' positions of all
     the unexamined arcs e with the head v. */
7.        while SOME(Z) do
8.          begin MAX(weight, Z, F);
9.            i :=FND(F); W(i) :=' 1';
  /* In the slice W we indicate by '1' the position i
```

```
    of the critical arc (e) with the head v. */
10.       t := t + 1; A[t] := i; tag[i] := t;
11.       u :=ROW(i, left); D(i) :=' 0';
12.       MATCH(node, Y, u, Q);
13.       if SOME(Q) then
  /* The case when u has not been processed yet. */
14.        begin k :=FND(Q); Y(k) :=' 0';
15.          MATCH(right, D, u, Z);
16.          if SOME(Z) then goto 3
17.          else
  /* There is no incoming arc for u. */
18.          begin trap(i) :=' 0'; goto 2
19.          end;
20.        end
21.        else
  /* The case when u has already been processed. */
22.        begin MATCH(right, W, u, F);
23.         if ZERO(F) then
24.          begin trap(i) :=' 0'; goto 2
25.          end;
  /* The second phase. */
26.          k :=FND(F); s := tag[k];
27.           for l := s to t do
28.            begin r := A[l]; X(r) :=' 1'
  /* In the slice X we indicate by '1' the positions of the arcs
     which are candidates for being included in a critical circuit. */
29.            end;
30.            P := trap; P := P and X; P := P xor X;
  /* We verify whether there is an arc whose position corresponds
     to '1' in the slice X and to '0' in the slice trap. */
31.            if SOME(P) then
32.             begin CLR(X); goto 2
33.             end;
34.             v :=ROW(r, right);
35.             MATCH(left, X, v, F);
36.              if ZERO(F) then
37.               begin CLR(X); goto 2
38.               end;
39.               r := A[s]; u :=ROW(r, left);
40.               MATCH(right, X, u, F);
41.                if ZERO(F) then
42.                 begin CLR(X); goto 2
43.                 end
```

```
44.                    else goto 1;
45.              end;
46.          3: end;
47.       2: end;
48.    1: End;
```

Remark 2. It is obvious that the initialization of variables is performed in lines 1–2 of the procedure CYCLE. In line 3, positions of the arcs forming a loop are indicated by $'1'$ in the slice P. By means of the statement $D := D\ and\ (not\ P)$, they are deleted from the global slice D.

Now, evaluate the time complexity of the procedure CYCLE. Assume that any elementary operation of the STAR–machine takes one unit of time. Therefore we will measure *the time complexity* of any algorithm by counting all the elementary operations performed in the worst case.

At first, let us observe that any of the procedures MATCH and HIT takes $O(\log n)$ time, where n is the number of graph vertices. It is obvious that the procedure MAX requires $O(1)$ time, since the arc weights do not depend on n. Therefore, the first stage requires $O(n \log n + \log n)$ time and the second phase requires $O(\log n)$ time. Hence, the procedure CYCLE takes $O(n \log n)$ time.

5. Correctness

To verify correctness of the procedure CYCLE, we will prove the following two theorems.

Theorem 1. *Let G=(V,E,w) be a directed weighted graph represented as association of matrices left, right, and weight and let node be a matrix in whose j-th row there is a binary code of the vertex j,* $1 \leq j \leq n$. *Then in the first phase, when processing any vertex k, which is a head of an arc, we select a position i of the critical arc which enters k. In addition, the variables* $D, Y, W, A, tag,$ *and trap are changed as follows:*

(1) $Y(k) :=' 0'$; $D(i) :=' 0'$;

(2) $W(i) :=' 1'$; $t := t + 1$; $A[t] := i$; $tag[i] := t$;

(3) $trap(i) :=' 0'$ *if there is no arc which enters the tail of the arc from the i-th position.*

Proof. We will prove this by induction on the number of vertices n.

Basis is verified for $n = 2$. There are two cases depending on whether there exists an incoming arc for one vertex or for two vertices.

(a) Let only one vertex have an incoming edge. There are two subcases.

(a1) Only vertex 1 has an incoming edge. Then, on performing line 5 and the procedure MATCH$(right, D, v, Z)$ from line 6, we obtain $Y(1) =' 0'$,

$v = 1$ and $Z \neq \Theta$.[1] Therefore, from the procedure MAX($weight, Z, F$) and the statement i :=FND(F) (lines 8–9), we find the position i of a critical arc (say γ) which enters the vertex $v = 1$. Then for $t = 1$ we carry out the statements $W(i) :=' 1'$, $t := t + 1$, $A[t] := i$, and $tag[i] := t$ (lines 9–10). On executing the statement u :=ROW($i, left$) (line 11), we obtain $u = 2$ since $u \neq v$ in view of Remark 1. By means of the statement $D(i) :=' 0'$ (line 11), the position of γ is deleted from D. Since $u = 2$ and $Y(2) =' 1'$, the procedure MATCH($node, Y, u, Q$) (line 12) returns $Q \neq \Theta$. Therefore, we perform the statements k :=FND(Q) and $Y(k) :=' 0'$ (line 14) and obtain $Y = \Theta$. In view of our assumption, for $u = 2$ the procedure MATCH($right, D, u, Z$) (line 15) returns $Z = \Theta$. Hence, we carry out the statement $trap(i) :=' 0'$ (line 18). Since $Y = \Theta$, we jump to the end of the procedure CYCLE.

(a2) Only vertex 2 has an incoming edge. Then one can easily verify that on performing lines 5–6, we obtain $Y(1) =' 0'$, $v = 1$ and $Z = \Theta$. Since $Y(2) \neq' 0'$, we fulfil line 5 again and obtain $Y = \Theta$. In view of our assumption for $v = 2$, the procedure MATCH($right, D, v, Z$) returns $Z \neq \Theta$. Therefore we carry out line 8. Further reasoning differs from (a1) in line 12. Here for $u = 1$ the procedure MATCH($node, Y, u, Q$) returns $Q = \Theta$. Therefore we carry out the procedure MATCH($right, W, u, F$) (line 22) which returns $F = \Theta$. Hence, we perform the statement $trap(i) :=' 0'$ from line 24 and jump to the end of the procedure CYCLE. So, the assertion is true when only one vertex has an incoming edge.

(b) Both vertices have incoming edges. Initially we reason by analogy with (a1). Difference arises in line 15, when fulfilling the procedure MATCH($right, D, u, Z$). Here, for $u = 2$ it returns $Z \neq \Theta$. Therefore, after performing line 16 we carry out line 8. Now, we reason by analogy with (a2). However, there arises the following distinction. Here, for $u = 1$ the procedure MATCH($right, W, u, F$) (line 22) returns $F \neq \Theta$, since vertex 1 is the left vertex of the critical arc selected when $t = 2$. Moreover, the position of the critical arc which enters vertex 1 was selected when $t = 1$ and it was indicated by '1' in W. Therefore we jump to the second phase. Hence, for any of two vertices the corresponding critical arc is defined and the variables D, Y, W, A, tag, and $trap$ satisfy (1)–(3).

Step of induction. Let Theorem 1 be true for graphs with no more than n vertices. We will prove it for the graphs with $n+1$ vertices. Let v_0 be the $(n+1)$-th vertex which has an incoming edge. By induction hypothesis the assertion is true for those vertices from the matrix *node* which are processed before v_0. Therefore it is sufficient to analyze only the case when the second phase is fulfilled after reading the vertex v_0. There are two cases.

Case 1. The vertex v_0 is processed starting from line 5. Since $v_0 \neq 1$,

[1]The notation $Z \neq \Theta$ denotes that there is at least one component '1' in the slice Z.

this is possible when the statement **while** SOME(Y) **do** is performed after the statements from line 18 or line 24. By analogy with the basis, on fulfilling lines 1–11 the position i of a critical arc (say σ) entering the vertex v_0 is defined and the variables D, Y, W, A, and *tag* satisfy (1)–(2). It remains to check that *trap* satisfies (3).

Really, on performing the statement u :=ROW($i, left$) (line 11) the contents of u is the left vertex (say v') of the arc σ. By means of the procedure MATCH($node, Y, u, Q$) (line 12) we verify whether v' has already been processed or not. There are two subcases.

(a) v' has not been processed yet. Then on performing line 14 and the procedure MATCH($right, D, u, Z$) (line 15), we verify whether v' has an incoming edge. It is easy to check that the statement $trap(i) :=' 0'$ from line 18 is performed only in the case when v' has no incoming edge. It takes place when the procedure MATCH from line 15 returns $Z = \Theta$.

(b) v' has already been processed. Again, it is easy to verify that the statement $trap(i) :=' 0'$ (line 24) is performed only in the case when v' has no incoming edge indicated by '1' in the slice W. It takes place when the procedure MATCH($right, W, u, F$) (line 22) returns $F = \Theta$. Hence, in both subcases the slice *trap* satisfies (3).

Case 2. The vertex v_0 is the left one of the critical arc from the i-th position. It means that on performing the statement u :=ROW($i, left$) (line 11) v_0 is the contents of u. Since v_0 has not been processed yet and it has an incoming edge, the procedure MATCH($right, D, u, Z$) (line 15) returns $Z \neq \Theta$. Therefore v_0 will be analyzed beginning from line 8 as shown above.

This completes the proof. □

Let us agree that an arc whose position is indicated by '1' in the result slice X of the procedure CYCLE is called *a resulting arc.*

Remark 3. To verify correctness of the procedure CYCLE, it is sufficient to prove that the resulting arcs make up a directed circuit, since in the first phase every time we select the position of the arc of the maximal weight.

Theorem 2. *Let G=(V,E,w) be a directed weighted graph represented as association of the matrices left, right, and weight and let node be a matrix in whose j-th row there is a binary code of the vertex j,* $1 \leq j \leq n$. *Then on performing the procedure CYCLE(left,right,weight,W,X) the positions of arcs forming a directed critical cycle are indicated by '1' in the slice X, whereas the positions of all the critical arcs selected during the procedure run are indicated by '1' in the slice W.*

Sketch of the proof. We will prove it by contradiction. Let the procedure CYCLE return a non-empty slice X (that is, $X \neq \Theta$) and the resulting arcs do not form a directed circuit. Three cases are possible.

Case 1. There exist at least two resulting arcs γ_1 and γ_2 such that $t(\gamma_1) = t(\gamma_2)$, but $h(\gamma_1) \neq h(\gamma_2)$. However, such a case is impossible, since any vertex of the matrix *node* is processed only once.

Case 2. There exist at least two resulting arcs γ_1 and γ_2 such that $h(\gamma_1) = h(\gamma_2)$, but $t(\gamma_1) \neq t(\gamma_2)$. Such a case is also impossible, since the selection of any critical arc is performed either for the left vertex of the current selected critical arc which has not been processed yet and has an incoming edge, or for the vertex in the matrix *node* which corresponds to the position of the first component '1' in Y.

Case 3. There is a sequence of the resulting arcs $\gamma_1, \gamma_2, \ldots, \gamma_k$ $(k \leq n)$ satisfying the following properties: for all i, j, $1 \leq i \leq k-1$, $1 \leq j \leq n$ we have $t(\gamma_i) = h(\gamma_{i+1})$; $h(\gamma_1) \neq t(\gamma_j)$; $t(\gamma_k) \neq h(\gamma_j)$. Without loss of generality it is sufficient to analyze only the case when a sole arc is missed from the directed circuit. We have the following subcases.

(3.1) The position of every arc γ_i has been selected at the time i. Without loss of generality we can assume that γ_i enters the vertex i. Let $\gamma_k = (1, k)$ be absent in the cycle. Hence, we have $k \to k-1 \to \ldots \to 2 \to 1$. As shown in Theorem 1, the positions of γ_i have been indicated by '1' in the slice W. It is easy to verify that as soon as $Y = \Theta$, we jump to the end of the procedure CYCLE. Since the second phase is not performed, the contents of X do not change. Hence, we obtain $X = \Theta$. This is a contradiction.

(3.2) The position of every arc γ_i has been selected at the time i. Let γ_i enter the vertex i. However, let us assume that the arc $\gamma_{k-1} = (k, k-1)$ is absent in the cycle. Therefore, we have $k-1 \to \ldots \to 2 \to 1 \to k$. In such a case in view of Theorem 1 the positions of arcs $\gamma_1, \gamma_2, \ldots, \gamma_{k-2}$ are indicated by '1' in W. Moreover, the position of γ_{k-2} is indicated by '0' in the slice *trap*, since no arc enters the vertex $k-1$. On selecting the position of $\gamma_k = (1, k)$, we obtain that its tail has been already processed in the matrix *node*. Therefore, one can easily verify that the procedure MATCH(*right*, W, u, F) (line 22) returns $F \neq \Theta$ for $u = 1$. Thus, on performing lines 27–29 of the second phase, the positions of the arcs $\gamma_1, \gamma_2, \ldots, \gamma_{k-2}, \gamma_k$ will be indicated by '1' in X. Clearly, among them there is the position of γ_{k-2} indicated by '0' in the slice *trap*. Hence, after performing the statement CLR(X) from line 32, we obtain $X = \Theta$. This is, again, a contradiction.

(3.3) Without loss of generality one can assume that the arcs γ_i form the following sequence: $k \to \ldots \to 5 \to 4 \to 1 \to 2 \to 3$, that is, the arc $(3, k)$ is absent in the cycle. By analogy with the previous subcase, on selecting the positions of arcs $k \to \ldots \to 4 \to 1 \to 2$ we obtain $X = \Theta$. One can verify that the arc $\gamma_p = (1, 2)$ is selected after the arc $(k, k-1)$. It is obvious that $\gamma_j = (2, 3)$ is selected immediately after γ_p. Since the left vertex of γ_j has already been processed, the procedure MATCH(*right*, W, u, F) (line 22) returns $F \neq \Theta$ for $u = 2$. Clearly, the positions of γ_p and γ_j correspond to '1' in the slice *trap*. Thus, on performing the statements from lines 27–29,

we obtain $X \neq \Theta$. Therefore, from line 30, we obtain $P = \Theta$. Then, the statement $v := \mathrm{ROW}(r, right)$ (line 34) will be performed immediately after line 30. It is not difficult to verify that r denotes the position of γ_j and v saves its tail, that is, vertex 3. Therefore, by means of the procedure $\mathrm{MATCH}(left, X, v, F)$ (line 35) we check whether vertex 3 has an outgoing edge whose position has been indicated by '1' in X. Since only positions of the arcs γ_p and γ_j are indicated by '1' in X, we obtain $F = \Theta$. Thus, on performing lines 36–38, we have $X = \Theta$. This is, again, a contradiction.

(3.4) Now, it is sufficient to assume that the arcs γ_i form the following sequence: $k \to k-1 \to \ldots \to 6 \to 1 \to 2 \to 5 \to 4 \to 3$, that is, the arc $(3, k)$ is absent in the cycle. By analogy with the previous subcase, on selecting the positions of arcs $k \to k-1 \to \ldots \to 6 \to 1 \to 2$ we obtain $X = \Theta$. As a result of performing **goto 2** we will update vertex 3 in the matrix *node*. One can verify that the arc (1,2) is selected immediately after the arc $(k, k-1)$. After selecting the arcs (4,3), (5,4) and (2,5), we obtain that the tail of the arc (2,5) has already been processed. Therefore as a result of performing the procedure $\mathrm{MATCH}(right, W, u, F)$ (line 22) we have $F \neq \Theta$ for $u = 2$. On fulfilling the statements from lines 26–33, the positions of arcs (1,2), (4,3), (5,4) and (2,5) are indicated by '1' in X. Moreover, the positions of these arcs are indicated by '1' in the slice *trap*. Therefore we carry out the statements from lines 34–38. One can easily check that the arc (1,2) has been selected at the time s. Since there is no incoming edge for vertex 1 among these arcs, on performing lines 39–43 we obtain $X = \Theta$. It is a contradiction. This completes the proof. □

6. Conclusion

In this paper we have proposed a novel associative parallel algorithm for finding a critical cycle in a directed weighted graph represented on the STAR-machine as a list of triples. It consists of two phases. In the first phase, in a special order, we update those vertices which have an incoming edge. Along with selecting every critical arc position in the slice W using two arrays A and *tag*, we establish a one–to–one correspondence between the critical arc position and the time of its selection. Moreover, the position of every processed critical arc whose tail has no incoming edge is saved in the slice *trap*. As soon as we update a critical arc whose left vertex has already been processed, we define the time s of selecting the arc which enters it. Then, using the array A and the slice W, we easily define positions of the critical arcs selected from the time s until the current time t. In the second phase, from the candidates, we easily select those arcs which form a critical cycle. To do this, we use the slice trap and verify whether the head of the arc selected at the time t has an outgoing edge and the tail of the arc selected

at the time s has an incoming edge.

This algorithm has been represented as the corresponding STAR procedure CYCLE whose correctness has been proved. We have obtained that it takes $O(n \log n)$ time. In particular, this procedure will be used to represent the associative version of Edmonds' algorithm for finding optimum branchings on the STAR–machine.

References

[1] R. M. Camerini, L. Fratta, F. Maffioli, *A Note on Finding Optimum Branchings*, Networks, No. 9, 1979, 309–312.

[2] J. Edmonds, *Optimum Branchings*, J. Res. Nat. Bur. Standards, **71B**, 1967, 233–240.

[3] C. Fernstrom, J. Kruzela, B. Svensson, *LUCAS Associative Array Processor. Design, Programming and Application Studies*, Lecture Notes in Computer Science, **216**, 1986.

[4] H. N. Gabow, Z. Galil, T. Spencer, *Efficient Implementation of Graph Algorithms Using Contraction*, Proc. 25-th Annual IEEE Symp. on Found. of Comp. Sci., 1984, 347–357.

[5] H. N. Gabow, Z. Galil, T. Spencer, R. E. Tarjan, *Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs*, Combinatorica, **6**, No. 2, 1986, 109–122.

[6] R. M. Karp, *A Simple Derivation of Edmonds' Algorithm for Optimum Branchings*, Networks, No. 1, 1972, 265–272.

[7] A. S. Nepomniaschaya, *Comparison of two MST Algorithms for Associative Parallel Processors*, Proc. of the 3-d Intern. Conf. "Parallel Computing Technologies", Lecture Notes in Computer Science, **964**, 1995, 85–93.

[8] A. S. Nepomniaschaya, *Representations of the Prim–Dijkstra Algorithm on Associative Parallel Processors*, Proc. of VII Intern. Workshop on Parallel Processing by Cellular Automata and Arrays. Parcella'96, Academie Verlag, Berlin, 1996, 184–194.

[9] A. S. Nepomniaschaya, *Representation of the Gabow Algorithm for Finding Smallest Spanning Trees with a Degree Constraint on Associative Parallel Processors*, Proc. of the Second Intern. Euro-Par Conf, Lecture Notes in Computer Science, **1123**, 1996, 813–817.

[10] A. S. Nepomniaschaya, O. V. Taborskaya, *Effective Representation of Some Graph Problems on Associative Parallel Processors*, Proceedings of the Twelfth International Symposium on Computer and Information Sciences, Bogazici University Printhouse, ISCIS XII, October 27–29, 1997, Antalya, Turkey, 430–437.

[11] A. S. Nepomniaschaya, *An Associative Version of the Prim–Dijkstra Algorithm and its Application to Some Graph Problems*, Proc. of the Andrei Ershov Second Intern. Memorial Conf. "Perspectives of System Informatics", Lecture Notes in Computer Science, **1181**, 1996, 203–213.

[12] A. S. Nepomniaschaya, *Solution of Path Problems Using Associative Parallel Processors*, Proceedings of the International Conference on Parallel and Distributed Systems, IEEE Computer Society Press, ICPADS'97, December 10–13, (1997), Korea, Seoul, 610–617.

[13] A. S. Nepomniaschaya, *Language STAR for Associative and Parallel Computation with Vertical Data Processing*, Proc. of the Intern. Conf. "Parallel Computing Technologies", World Scientific, Singapure, 1991, 258–265.

[14] B. Otrubova, O. Sykora, *Orthogonal Computer and its Application to Some Graph Problems*, Parcella'86, Berlin, Academie Verlag, 1986, 259–266.

[15] J. L. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers*, Kent State University, Plenum Press, New York and London, 1992.

[16] R. E. Tarjan, *Finding Optimum Branchings*, Networks, No. 7, 1977, 25–35.