Bull. Nov. Comp. Center, Comp. Science, 9 (1998), 77–92

An associative version of Edmonds' algorithm for finding optimum branchings

A.S. Nepomniaschaya

In this paper we propose a representation of Edmonds' algorithm for finding optimum branchings on an abstract model of the SIMD type with vertical data processing (the STAR-machine). To this end for a directed graph given as a list of triples (edge vertices and the weight) we construct associative algorithms for finding a critical cycle (or a cycle of the maximal weight), for contracting a critical cycle and for expanding the nested critical cycles. We obtain that on vertical processing systems Edmonds' algorithm takes $O(n \log n)$ time, where n is the number of graph vertices.

Introduction

Edmonds' algorithm for finding optimum branchings [2] has important applications to the graph theory and the combinatorial optimization. In particular, it is used for finding a minimal spanning tree in a directed graph. Edmonds' algorithm takes polynomial time. Therefore for improving the time bound in [1, 4, 5, 17] different algorithms for its implementation are devised. 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. On sequential computers this algorithm takes $O(n \log n + m)$ time, where n is the number of graph vertices and m is the number of edges.

For improving the time bound of Edmonds' algorithm and for obtaining its natural implementation we will use associative parallel processors. These systems are oriented to solving non-numerical problems. They provide a massively parallel search by contents and processing of unordered data represented in the form of two-dimensional tables [16].

Let us enumerate graph problems being represented on associative parallel processors. In [8, 9, 13, 15, 16], the problem of finding a minimal spanning tree in undirected graphs is studied for different graph representation forms using different algorithms and different models of associative parallel processors. In [10], the associative version of Gabow's algorithm for the degree-restricted problem is represented on the STAR-machine being a model of associative parallel processors with vertical data processing.

Problems of finding connected components, transitive closure of a graph, verifying an articulation point and verifying a bridge are considered for graphs represented as an adjacency matrix both on the orthogonal machine [15] and on the STAR-machine [12]. The problem of finding the shortest path between two given vertices and the all-pairs shortest path problem are examined both on the associative array processor LUCAS [3] and on the STAR-machine [11, 13].

In this paper, we study a representation of Edmonds' algorithm for finding optimum branchings on the STAR-machine. To this end for a directed graph given as a list of triples (edge vertices and the weight) we construct associative algorithms for finding a critical cycle (or a cycle of the maximal weight), for contracting a critical cycle and for expanding the nested critical cycles. We obtain that on vertical processing systems Edmonds' algorithm takes $O(n \log n)$ time, assuming that any elementary operation of the model under consideration (its microstep) takes one unit of time.

1. 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:

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

2) an associative processing unit consisting of m single-bit processing elements (PEs);

3) 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 equally easy. 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 m bits. 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 [14] being 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`. We employ the types `slice` and `word` for bit column access and bit row access, respectively, and

the type **table** for defining the tabular data. Assume that any variable of the type **slice** consists of m components which belong to $\{0, 1\}$.

Consider operations and predicates for slices.

Let X, Y be variables of the type **slice** and i, j be variables 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 component '1' of Y, $i \geq 0$;
STEP(Y)	returns the same result as **FND**(Y) and then resets the first component '1';
NUMB(Y)	returns the number i of components '1' in Y, $i \geq 0$;
MASK(Y, i, j)	sets components '1' from the i-th through the j-th positions, inclusively, and components '0' in other positions of the slice Y $(1 \leq i < j \leq m)$.

We utilize the bitwise Boolean operations 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.

2. Preliminaries

At first, let us recall some notions being used in the paper.

Let $G = (V, E, w)$ be a *directed weighted graph* in which $V = \{1, 2, \ldots, n\}$ is a finite set of vertices, $E \subseteq V \times V$ is a finite set of directed edges (arcs) and w is a function 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 $t(e) = u$.

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 with 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 a group of basic procedures [12, 13] which use the given global slice X for indicating by $'1'$ the row positions being used in the corresponding procedure.

The procedure $\mathtt{MATCH}(T, X, v, Z)$ 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 $\mathtt{ROW}(i, T) = v$ and $X(i) = '1'$. The procedure $\mathtt{MAX}(T, X, Z)$ 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 $\mathtt{ROW}(i, T)$ is the maximal matrix element and $X(i) = '1'$. The procedure $\mathtt{MIN}(T, X, Z)$ is defined by analogy with MAX. The procedure $\mathtt{HIT}(T, R, X, Z)$ defines positions of the corresponding identical rows in the given matrices T and R using the global slice X. It returns the slice Z in which $Z(i) = '1'$ if $\mathtt{ROW}(i, T) = \mathtt{ROW}(i, R)$ and $X(i) = '1'$. The procedure $\mathtt{WMERGE}(w, X, R)$ writes the given w into those rows of the given matrix R which correspond to positions $'1'$ in the slice X. The procedure $\mathtt{TCOPY}(T, R)$ writes the given matrix T in the result martix R.

We will also use the procedure $\mathtt{NEWCOST}(u1, u2, X, R)$ which changes the result matrix R as follows. At first, it adds the given binary word $u1$ to the rows of R corresponding to positions $'1'$ in the slice X. Then, it subtracts the given word $u2$ from the rows of R indicated by $'1'$ in X.

Following [15], we assume that any elementary operation of the STAR-machine needs one unit of time. Therefore time complexity of an algorithm is measured by counting all the elementary operations in the worst case.

Note that any of the considered procedures takes $O(k)$ time, where k is the number of bit columns of the corresponding matrix.

3. Representation of Edmonds' algorithm on the STAR-machine

In this section at first we describe Edmonds' algorithm. Then, we propose its representation on the STAR-machine.

Edmonds' algorithm for finding optimum branchings consists of two phases [5, 7].

The **first phase** selects critical arcs. It begins with an empty set of arcs. For any vertex v being a head of an arc a critical arc (u, v) is selected and it is 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 any arc with the endpoint in C has that endpoint replaced with v^*. Moreover, the weights of all the arcs entering the cycle C are redefined as one describes 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** 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 to obtain the maximal branching. For deleting an arc from any current critical cycle Edmonds' algorithm uses the following rule from [7]. Let C be a critical cycle and let e'' be its arc having the minimal weight. Let B be an optimum branching in the reduced graph G/C for C. Let e be an arc from B enters C and let e' be such an arc from C that $h(e) = h(e')$. Then, $B + C - e'$ is the optimum branching in G. If there is no arc entering the cycle C, then $B + C - e''$ is the optimum branching.

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

Edmonds' algorithm will be represented as procedure `EDMONDS` having the following input parameters:

1) a graph G given as association of the matrices *left*, *right* and *weight*;
2) a list of graph vertices and super-vertices given as matrix *node* in whose any i-th row there is the binary code of the vertex i, $1 \le i \le n$, while in any j-th row, $n + 1 \le j \le 2n$, there is the binary code of the super-vertex j.

This procedure returns a slice W in which positions of arcs, belonging to an optimum branching, are indicated by '1'. It uses global variables n and m, where $n = |V|$ and $m = |E|$.

The procedure `EDMONDS` consists of the following phases:

- selecting a critical cycle;
- contracting a critical cycle;
- expanding the nested cycles.

For convenience, when describing any phase we will consider only those auxiliary variables which will be immediately used.

3.1. Associative algorithm for finding a critical cycle

At first, we briefly explain the meaning of the main variables being used here for describing the associative algorithm for finding a critical cycle.

The procedure **EDMONDS** uses a global slice D for the matrices *left*, *right* and *weight*; a global slice Y for the matrix *node*; a slice X for accumulating positions of selected critical arcs; an integer t for recording 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 by '0' positions of those critical arcs which cannot be included into the critical cycle; auxiliary slices F, P and Z for the matrix *right* and the slice Q for the matrix *node*.

At the stage of initializing the procedure **EDMONDS** at first we perform the following statements: $t := 0$, **CLR**(X), **CLR**(W), **SET**(D), **SET**$(trap)$, and **MASK**$(Y, 1, n)$. Then, by means of the basic procedure **TCOPY**, we obtain matrices *left1*, *right1*, and *weight1* being the copies of the corresponding matrices *left*, *right*, and *weight*. Finally, by means of the procedure **HIT** we define positions of all the arcs forming a cycle of length 1. These positions are deleted from the global slice D.

For any unexamined vertex of the matrix *node* for which there is an incoming edge the associative algorithm defines the position of a critical arc which enters it using *a special order* of selecting critical arcs. Let us explain this. At first, in the matrix *node* by means of the statement $k :=$ **STEP**(Y) we define the position of the vertex v for which there is an incoming edge. Then, in the matrix *right1* we select the position i of a critical arc (say, e_1) which enters v. Now, by means of the statement $u :=$ **ROW**$(i, left1)$ we select the left vertex of e_1. If u is such an unexamined vertex in the matrix *node* for which there is an incoming edge, we select the next critical arc (say, e_2) which enters u, etc. In other words, so long as it is possible, we select a set of critical arcs $e_1, e_2, \ldots, e_k$ which satisfy the condition $t(e_i) = h(e_{i+1})$, $1 \leq i \leq k-1$. If for the unexamined vertex u of the matrix *node* there is no incoming edge, the position of the critical arc e_1 is indicated by '0' in the slice *trap*. In such a case the next critical arc is selected for the unexamined vertex v' of the matrix *node* which corresponds to the position of the first component '1' in the slice Y and for which there is an incoming edge.

On the STAR-machine the associative parallel algorithm runs as follows.

At the **first step** by means of the statement $k :=$ **STEP**(Y) we select the position of the vertex v in the matrix *node* for which there exists an incoming edge in G. We do it with the use of the procedure **MATCH**$(right1, D, v, Z)$. This step yields a slice Z in which positions of all the arcs with the *head* v are indicated by '1'.

At the **second step** we define a position i of the critical arc (say, e) with the *head* v after performing the procedure **MAX**$(weight1, Z, F)$ and the

statement $i := \mathtt{FND}(F)$. This position is indicated by '1' in the slice W by means of the statement $W(i) := '1'$. After performing the statements $t := t + 1$, $A[t] := i$, and $tag[i] := t$ we set a one-to-one correspondence of the form $i \leftrightarrow t$ between the position i of the critical arc e and the time of its selection. After that we delete the position i from the global slice D by means of the statement $D(i) := '0'$. Then we jump to the third step.

At the **third step** we select the tail u of the critical arc e from the i-th position using the statement $u := \mathtt{ROW}(i, left1)$. Then, we perform the procedure $\mathtt{MATCH}(node, Y, u, Q)$ to verify whether the vertex u has been examined in the matrix *node*. 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 run to the second step to select the next critical arc. Otherwise, in the i-th component of the slice *trap* we indicate by '0' the position of the critical arc e by means of the statement $trap(i) := '0'$. Such an arc cannot be included in the critical cycle because it destroys the cycle.

Case 2. The vertex u has been already processed. Then by means of the matrix *right1* we define the position k of the critical arc (say, e') from the slice W which enters the vertex u. We do it using the procedure $\mathtt{MATCH}(right1, W, u, F)$ and the statement $k := \mathtt{FND}(F)$. If u has no incoming edge, that is, $F = \Theta$,[1] the position of e is saved in the slice *trap* by analogy with Case 1, and we jump to the first step. Otherwise, by means of the statement $s := tag[k]$ 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 jump to the fourth step.

At the **fourth step** 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 components '0' in the slice X by means of the operation $\mathtt{CLR}(X)$ 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. Otherwise, after performing the operation $\mathtt{CLR}(X)$, we jump to the first step.

Since critical arcs are selected for different graph vertices, this process will be completed.

Hence, after terminating the phase of constructing a critical cycle, we obtain the following information:

[1]The notation $F = \Theta$ denotes a slice F with components '0'.

1) positions of selected critical arcs are indicated by '1' in the slice W and by '0' in the global slice D;
2) positions of unexamined graph vertices of the matrix *node* are indicated by '1' in the global slice Y;
3) positions of arcs, forming a critical cycle, are indicated by '1' in X;
4) positions of arcs which cannot be included in a critical cycle are indicated by '0' in the slice *trap*.

Moreover, if $X \neq \Theta$, then the first critical arc was selected at the time s, while the last one was selected at the time t.

3.2. Associative algorithm for contracting a critical cycle

At first, we briefly explain the meaning of the main variables being used here.

To perform the second phase, the procedure **EDMONDS** uses a slice *newvert* for indicating positions of super-vertices in the matrix *node*; an integer j for counting the number of critical cycles; a matrix *protoc* in whose j-th column the positions of the j-th critical cycle are indicated by '1'; an array B in whose j-th component the position of the arc having the minimal weight in the j-th critical cycle is stored; a binary word $w0$ being the minimal weight of the j-th critical cycle; a matrix *incomer* in whose j-th column the positions of the outer arcs, which enter the j-th critical cycle, are indicated by '1'; a matrix *tour* in whose j-th column one saves the positions of the arcs e from the j-th critical cycle for which $h(e)$ is the head of an outer incoming edge e', that is, $h(e) = h(e')$; auxiliary slices $D1$, H, P, and $W1$.

Note that at the phase of initialization we fulfil the statements $j := 0$ and **MASK**$(newvert, n+1, 2n)$ being used when performing the second phase.

The associative algorithm for contracting a critical cycle consists of the following steps which are performed one by one.

At the **first step** in the current j-th column of the matrix *protoc* we memorize the j-th critical cycle, that is, the slice X. Then, by means of the statements $W := W$ **and (not** X**)** the arc positions included in the j-th critical cycle C are deleted from the slice W. After that by means of the procedure **MIN**$(weight1, X, F)$ and the statement $i :=$ **FND**(F) we define the arc position from the j-th critical cycle having the minimal weight. By means of the statements $B[j] := i$ and $w0 :=$ **ROW**$(i, weight1)$ we memorize this position in the j-th component of the array B and the minimal weight in $w0$. Now, by means of the statements $k :=$ **STEP**$(newvert)$, $Y(k) :=$ '1', and $v0 :=$ **ROW**$(k, node)$ we select the current super-vertex $v0$.

At the **second step** at first, we define positions of those arcs in the reduced graph G/C whose endpoints belong to C. To take into account the arc positions from the slice W by means of the statement $D1 := D$ **or** W we

introduce a new global slice $D1$. By means of the statement **while** SOME(X) **do** for any critical arc from the i-th position we define its right vertex $w1$. Using the procedure MATCH(*left1*, $D1, w1, F1$) and the statement $U1 := U1$ **or** $F1$ in the slice $U1$ we save positions of those arcs whose left vertex is $w1$. Similarly, in the slice $P1$ we save positions of the arcs whose right vertex is $w1$. Now, by means of the statement $F := F1$ **and (not** W**)** we define positions of *unexamined* arcs with the head $w1$ whose weights should be redefined. If $F \neq \Theta$, then with the use of $H(i) := '1'$ we memorize this arc position in the slice H. After that by means of the procedure **NEWCOST** we modify in parallel the weights of the arcs indicated by '1' in F.

Hence, for any j-th critical cycle as soon as $X = \Theta$, positions of outer arcs whose left vertices belong to C are saved in the slice $U1$, while positions of outer arcs whose right vertices belong to C are saved in $P1$. Moreover, in the slice H we indicate by '1' positions of all the arcs from C whose right vertices are the heads of outer unexamined incoming edges.

At the **third step**, by means of the procedures WMERGE($v0, P1,$ *right1*) and WMERGE($v0, U1,$ *left1*), we replace the arc endpoints belonging to C with the super-vertex $v0$. Then, using the statement $Z := U1$ **and** $P1$ we define positions of the unexamined arcs which connect two vertices from C. These positions are deleted from the global slice D after performing the statement $D := D$ **and (not** Z**)**. Now, by means of the statements COL(j, *tour*) := H, $P := P1$ **and (not** W**)**, and COL(j, *incomer*) := P we memorize the remaining information about the j-th critical cycle being used later on.

At **the fourth step** we set a new value $s - 1$ for the current time t. In other words, we eliminate the time which is used for selecting the last critical cycle. If $t = 0$, then we jump to the first step of the associative algorithm for finding a critical cycle. Otherwise, we delete the super-vertex $v0$ position from the global slice Y using the statement $Y(k) := '0'$. After that we select a next critical arc beginning with the current time $s-1$. As it can be proved, for doing it we need to perform the procedure MATCH(*right1*, $D, v0, Z$). Then, we jump to the second step of the associative algorithm for finding a critical cycle.

3.3. Associative algorithm for expanding the nested cycles

Here, we explain how to implement the second stage of Edmonds' algorithm. It should be noted that we will use the matrix *right*.

The associative algorithm for expanding the nested cycles is fulfilled as follows. While the number of critical cycles $j > 0$, we perform the following statements. At first, we select the information about the j-th critical cycle being gathered at the phase of contracting a critical cycle. We do it with the use of statements $Z :=$ COL(j, *protoc*), $P :=$ COL(j, *incomer*), $H :=$ COL(j, *tour*), and $r := B[j]$. Then, after performing the statement $P := P$

and W we verify whether there is such an arc, entering the j-th critical cycle, whose position belongs both to the slice P and the slice W. There are two cases.

Case 1. $P = \Theta$. Then we delete from the j-th critical cycle the position of its minimal weight using the statement $Z(r) := '0'$. Now, we perform two statements $W := W$ **or** Z and $j := j - 1$.

Case 2. $P \neq \Theta$. Then we have to define the arc position from the j-th critical cycle being the head of an incoming edge. To this end at first we perform the statements $i :=$ **FND**(P) and $w1 :=$ **ROW**$(i, right)$. Then we fulfil the procedure **MATCH**$(right, H, w1, F)$ and the statements $s :=$ **FND**(F) and $Z(s) :=' 0'$. Now, we perform the statements $W := W$ **or** Z and $j := j - 1$.

The algorithm completes as soon as $j = 0$.

Now, let us evaluate time complexity of the associative version of Edmonds' algorithm. We obtain that the procedure **EDMONDS** takes $O(\log n)$ time for the initialization, $O(n \log n)$ time for finding a critical cycle, $O(n \log n)$ time for contracting a critical cycle and $O(\log n)$ time for expanding the nested cycles. Hence, the procedure **EDMONDS** takes $O(n \log n)$ time.

4. Conclusion

In this paper we have shown that Edmonds' algorithm for finding optimum branchings can be represented on vertical processing systems in a natural manner. To this end we have used a graph representation as a list of triples (edge vertices and the weight). We have proposed associative algorithms for finding a critical cycle, for contracting a critical cycle, for expanding nested critical cycles. We have obtained that after constructing of a critical cycle C we have to exclude from the further consideration positions of those unexamined arcs which connect two vertices from C. In particular, these vertices can belong to different arcs from C. We have also obtained that on the STAR-machine Edmonds' algorithm for optimum branchings takes $O(n \log n)$ time. Hence, the use of vertical processing systems for representing Edmonds' algorithm for optimum branchings gives an evident time improvement over the sequential computers.

References

[1] R.M. Camerini, L. Fratta, and 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, and B. Svensson, *LUCAS associative array processor. Design, programming and application studies*, Lecture Notes in Computer Science, **216**, Springer–Verlag, Berlin, 1986.

[4] H.N. Gabow, Z. Galil, and 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, and 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] E. Minieka, *Optimization Algorithms for Networks and Graphs*, Marcel Dekker Inc., New York and Bacel, 1978.

[8] A.S. Nepomniaschaya, *Comparison of two MST algorithms for associative parallel processors*, Proc. of the 3-d Intern. Conf. "Parallel Computing Technologies", PaCT-95, St.-Petersburg, Russia, Lecture Notes in Computer Science, **964**, 1995, 85–93.

[9] 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.

[10] A.S. Nepomniaschaya, *Representation of the Gabow algorithm for finding smallest spanning trees with a degree constraint on associative parallel processors*, Euro-Par'96 Parallel Processing. Second Intern. Euro-Par Conf. Lyon, France. Proceedings, Lect. Notes in Comp. Sci., Springer-Verlag, Berlin, **1123**, 1996, 813–817.

[11] 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, Antalya, Turkey, October 27–29, 1997, 430–437.

[12] A.S. Nepomniaschaya, *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, Springer-Verlag, Berlin, **1181**, 1996, 203–213.

[13] 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, Korea, Seoul, December 10–13, 1997, 610–617.

[14] 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.

[15] B. Otrubova and O. Sykora, *Orthogonal computer and its application to some graph problems*, Parcella'86, Berlin, Academie Verlag, 1986, 259–266.

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

[17] R.E. Tarjan, *Finding optimum branchings*, Networks, No. 7, 1977, 25–35.

Appendix

Here, at first we verify the correctness of the associative version of Edmonds' algorithm for finding optimum branchings. Then, we examine the representation of Edmonds' algorithm for finding optimum branchings on the STAR-machine.

Correctness of the associative algorithm for finding a critical cycle is established by means of the following two theorems.

Theorem 1. *Let G be a directed weighted graph represented as association of matrices left, right and weight. Let node be a matrix in whose l-th row there is the binary code of the vertex l, $1 \leq l \leq n$, while in its j-th row $(n+1 \leq j \leq 2n)$ there is the binary code of the j-th super-vertex. Then when processing any vertex k from the matrix node, being a head of an arc in the graph G or in the reduced graph G/C, we select a position i of the critical arc which enters k. In addition, the variables t, 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.*

This theorem is proved by induction on the number of vertices.

Theorem 2. *Let G be a directed weighted graph represented as association of the matrices left, right and weight and let node be a matrix in whose $l-th$ row there is the binary code of the vertex l, $1 \leq l \leq n$, while in its j-th row $(n+1 \leq j \leq 2n)$ there is the binary code of the j-th super-vertex. Then the phase of finding a critical cycle returns a slice X in which the positions of arcs, forming a directed critical cycle, are indicated by $'1'$.*

This theorem is proved by contradiction.

Correctness of the associative algorithm for contracting a critical cycle is immediately verified. It is only necessary to make some remarks.

Remark 1. For any critical cycle C after performing the statements

$$Z := U1 \text{ and } P1 \quad \text{and} \quad D := D \text{ and } (\text{not } Z)$$

we exclude from further consideration all the multiple arcs with the arcs from C, all the loops of length 1 formed by the vertices from C and all the unexamined arcs which connect two different vertices from C.

Remark 2. After contracting a critical cycle C whose first arc was selected at the time s we perform the statement $t := s - 1$. After that we will select other critical arcs beginning with the time s. Really, at the time $s - 1$ we have selected a critical arc from the l-th position, where $l = A[s-1]$, whose left vertex (say, v') belongs to C. Therefore after contracting the cycle C the vertex v' is replaced with the super-vertex $v0$. If $t \neq 0$, in view of performing the procedure `MATCH`$(right1, D, v0, Z)$ we run to the second step of the associative algorithm for finding a critical cycle. Here, at the time s we will select a critical arc which enters the vertex $v0$.

The associative algorithm for expanding the nested cycles is based on the second phase of Edmonds' algorithm. Its correctness is checked by induction on the number of nested cycles.

Now, consider the following procedure.

```
proc EDMONDS(left,right,weight: table; node: table;
  var W: slice(left));
/* We use global variables n and m of the integer type. */
  label 1, 2, 3;
  var i,j,k,l,r,s,t: integer;
    newvert,Q,Y: slice(node);
    D,D1,F,F1,H,P,P1,U1,X,Z,trap: slice(left);
    protoc,incomer,tour,left1,right1,weight1: table;
    tag,A,B: array [1:m] of integer;
    u,v,v0,w0,w1,w2: word;
/* Initialization. */
1. Begin t:=0; j:=0;
2.   CLR(W); CLR(X);
3.   SET(D); SET(trap);
4.   MASK(Y,1,n);
5.   MASK(newvert,n+1,2n);
6.   TCOPY(left,left1);
7.   TCOPY(right,right1);
8.   TCOPY(weight,weight1);
9.   HIT(left,right,D,P);
10.  D:=D and (not P);
/* The stage of selecting critical arcs. */
```

```
11.  while SOME(Y) do
     /* The phase of constructing a critical cycle. */
12.    begin k:=STEP(Y); v:=ROW(k,node);
13.      MATCH(right1,D,v,Z);
     /* In the slice Z we indicate by '1' positions of all the unexamined arcs
        with the head v. */
14.      while SOME(Z) do
15.        begin MAX(weight1,Z,F);
16.          i:=FND(F); W(i):='1';
     /* In the slice W we indicate by '1' the position i of the critical arc (e)
        having the head v. */
17.          t:=t+1; A[t]:=i; tag[i]:=t;
18.          u:=ROW(i,left1); D(i):='0';
19.          MATCH(node,Y,u,Q);
20.          if SOME(Q) then
     /* The case when u has not been processed yet. */
21.            begin k:=FND(Q); Y(k):='0';
22.              MATCH(right1,D,u,Z);
23.              if SOME(Z) then goto 3
24.              else
     /* There is no incoming arc for u. */
25.                begin trap(i):='0'; goto 2
26.                end;
27.            end
28.          else
     /* The case when u has been already processed. */
29.            begin MATCH(right1,W,u,F);
30.              if ZERO(F) then
31.                begin trap(i):='0'; goto 2
32.                end;
33.              k:=FND(F); s:=tag[k];
34.              for l:=s to t do
35.                begin r:=A[l]; X(r):='1'
     /* In the slice X we indicate by '1' positions of the arcs being candidates
        for including into a critical circuit. */
36.                end;
37.              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. */
38.             if SOME(P) then
39.               begin CLR(X); goto 2
40.               end;
41.             v:=ROW(r,right1);
42.             MATCH(left1,X,v,F);
43.             if ZERO(F) then
44.               begin CLR(X); goto 2
45.               end;
46.             r:=A[s]; u:=ROW(r,left1);
47.             MATCH(right1,X,u,F);
48.             if ZERO(F) then
49.               begin CLR(X); goto 2
50.               end;
    /* The phase of contracting a critical cycle. */
51.             j:=j+1; COL(j,protoc):=X;
    /* In j-th column of the matrix protoc we save the positions of
       j-th critial cycle. */
52.             W:=W and (not X); CLR(H);
53.             MIN(weight1,X,F); i:=FND(F);
54.             B[j]:=i; w0:=ROW(i,weight1);
55.             if ZERO(D) then
56.               begin COL(j,incomer):=H;
57.                 COL(j,tour):=H; goto 1
    /* We jump to the stage of expanding the nested cycles. */
58.               end;
59.             k:=STEP(newvert); Y(k):='1';
60.             v0:=ROW(k,node);
    /* We select the current super-vertex v0. */
61.             D1:=D or W; CLR(P);
62.             CLR(P1); CLR(U1);
63.             while SOME(X) do
64.               begin i:=STEP(X);
65.                 w1:=ROW(i,right1);
66.                 MATCH(left1,D1,w1,F1);
67.                 U1:=U1 or F1;
68.                 MATCH(right1,D1,w1,F1);
69.                 P1:=P1 or F1; F:=F1 and (not W);
```

```
    /* In the slice F there are positions of arcs whose weights should be redefined. */
70.                   if SOME(F) then
71.                     begin H(i):='1'; w2:=ROW(i,weight1);
72.                       NEWCOST(w0,w2,F,weight1)
73.                     end;
74.                 end;
75.               Z:=U1 and P1; D:=D and (not Z);
    /* From the slice D we delete positions of the unexamined arcs which connect
       two vertices from the critical cycle. */
76.               if ZERO(D) then
77.                 begin CLR(H); COL(j,tour):=H;
78.                   COL(j,incomer):=H; goto 1
79.                 end;
80.               if SOME(P1) then WMERGE(v0,P1,right1);
81.               if SOME(U1) then WMERGE(v0,U1,left1);
82.               COL(j,tour):=H; P:=P1 and (not W);
83.               COL(j,incomer):=P; t:=s-1
84.             end;
85.           if t=0 then goto 2
86.           else
87.             begin Y(k):='0';
88.               MATCH(right1,D,v0,Z)
89.             end;
90.      3: end;
91.  2: end;
    /* The stage of expanding the nested cycles. */
92. 1:while j>0 do
93.     begin Z:=COL(j,protoc);
94.       H:=COL(j,tour); r:=B[j];
95.       P:=COL(j,incomer); P:=P and W;
96.       if ZERO(P) then Z(r):='0'
    /* We delete the position of the arc having the minimal weight. */
97.       else
98.         begin i:=FND(P); w1:=ROW(i,right);
99.           MATCH(right,H,w1,F); s:=FND(F);
100.          Z(s):='0'
101.        end;
102.      W:=W or Z; j:=j-1
103.    end;
104. End;
```