Joint NCC & IIS Bull., Comp. Science, 4 (1996), 55–66
© 1996 NCC Publisher

Associative version of the Gabow algorithm for finding smallest spanning trees with a degree constraint*

A.S. Nepomniaschaya

In this paper by means of an abstract model (the STAR-machine) we describe an associative version of the Gabow algorithm for the degree-restricted problem. We have obtained that on an associative parallel processor this algorithm takes the same time as a minimal spanning tree algorithm. In Appendix the associative algorithm is represented as a procedure written in the language STAR.

1. Introduction

The revived interest in associative (content-addressable) architectures is caused by remarkable advances in the VLSI technology [1]. We analyze algorithms for associative parallel processors (APPs) belonging to fine-grained SIMD-systems with bit-serial (vertical) processing and simple single-bit processing elements [2]. These systems can process rows and columns organized as a matrix of storage elements. Therefore such an architecture is well-suited for solving problems employing tabular data. Our prime interest is in associative system application for solving graph theoretical problems.

For graphs represented as a distance matrix the Prim–Dijkstra algorithm is implemented both on the associative array processor Lucas [3] and on a group of computers, including the Staran, Aspro, Wavetracer and CM-2, where the language ASC has been installed [4]. In [5], for graphs represented as a list of triples (edge vertices and the weight) the Kruskal algorithm is used on the orthogonal machine. In [6], for the same graph representation the effective program MST is based on the Baase algorithm. In [7], we compare the implementations of the Baase algorithm and an associative version of the Prim–Dijkstra algorithm on the STAR-machine. In [3, 5] some other graph problems are considered on the associative array processor LUCAS and on the orthogonal machine.

In this paper, we study a problem of finding a smallest spanning tree with a degree constraint. Such an important problem arises in computer networks and communication networks. On conventional sequential computers the

*Supported in part by the Russian Foundation for Basic Research under Grant 96-01-01704.

Gabow algorithm [8] allows an effective solution which takes $O(m \log\log n + n \log n)$ time, where m is the number of graph edges and n is the number of vertices. In [8], Gabow posts a question whether there is an algorithm for finding the smallest spanning tree with a degree constraint as fast as a minimal spanning tree algorithm.

In [9], it has been shown that on sequential computers the degree-restricted problem is time-equivalent to the MST-problem. In [10], the best sequential algorithm for the MST-problem runs in time $O(m \log \beta(m,n))$, where $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}$, $\log^{(i)} x$ is defined by $\log^{(0)} x = x$, $\log^{(i)} x = \log\log^{(i-1)} x$. To this end a special data structure F-heap is used.

Here, we study the above-mentioned problem using the STAR-machine. Its run is described by means of the language STAR [11] being an extension of Pascal by adding new data types for simulating the parallel associative processing at micro-level. For representing the Gabow algorithm on the STAR-machine we suggest new techniques. We have obtained that the associative version of the Gabow algorithm for the degree-restricted problem takes the same time $O(n \log n)$ as the minimal spanning tree algorithm on the STAR-machine. Note that the STAR-machine can employ only simple tabular data structures.

In Appendix the associative algorithm is represented as a STAR procedure SSTEQ.

2. Model of associative parallel machine

We define our model as an abstract STAR-machine of the SIMD–type with vertical data processing. It consists of the following components:

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

Data are loaded in the matrix memory in the form of two-dimensional arrays written in binary code. Each array element occupies an individual row and all elements have the same length (coinciding with the length of the maximal 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 arrays may be loaded in the matrix memory.

The associative processing unit is represented as h vertical registers each consisting of m bits. The bit columns of the data array are stored in the registers which perform the necessary Boolean operations and record the search results. We assume that the associative processing unit has a sufficient number of vertical registers to store intermediate results of data processing without using the matrix memory.

Let us briefly consider the STAR constructions from [11] needed for the paper. To simulate data processing in the matrix memory we introduce three new data types **word, slice** and **table** in the same manner as in Pascal. Constants for the types **slice** and **word** are represented as a sequence of symbols from $\{0, 1\}$ 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 defining 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 be a variable of the type **integer**. We define the following operations:

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

CLR(Y) sets all 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'.

The following bitwise Boolean operations are introduced in the usual way: X *and* Y is conjunction, X *or* Y is disjunction, *not* X is negation and X *xor* Y is exclusive '*or*'.

For slices there are two predicates ZERO(Y) and SOME(Y) which are introduced in the obvious way.

Let T be a variable of the type **table**. We use the following operations:

$T(i)$ returns the i-th row in the matrix T $(1 \leq i \leq m)$;

col(i, T) returns the i-th column in the matrix T;

with(Y, T) attaches to the matrix T left the contents of the slice Y.

We employ the function size(T) which returns the number of columns in the matrix T. Notice that after performing the operation with(Y, T) the value of size(T) is incremented to one.

Remark 1. Note that the STAR statements resemble those of Pascal.

Remark 2. When we define a variable of the type **slice** we put in brackets the name of the matrix which uses it. Therefore if the matrix consists of n rows, where $n < m$, then only the **first** n components of the corresponding slice (column) will be used in the vertical processing.

3. The Gabow algorithm

At first let us recall some notions needed for the paper.

Let $G=(V, E, w)$ be an *undirected weighted graph* with the vertex set $V = \{1, 2, \ldots, n\}$, the edge set $E \subseteq V \times V$ and the weight function w correlating each edge $e = (p, q) \in E$ with an integer $w(p, q)$. **A minimal spanning tree** T_S of G is defined as a connected graph without loops containing all vertices from V where the sum of weights of the corresponding edges is minimal. **A degree of a graph vertex** is the number of edges incident to it. **A connected component** of a graph is a maximal connected subgraph.

Let t be a degree constraint for the vertex r in G and $\mathbf{deg}(r) = k$. The Gabow algorithm [8] constructs the smallest spanning tree of G containing only t edges incident to r. Denote by R a set of edges incident to r. The basic idea of the algorithm is to find a smallest spanning tree containing R. Then we execute elementary exchanges until the degree of r decreases to t. For speeding up the computation at first we construct a minimal spanning forest U of the graph $G - r$. Then only edges from $R \cup U$ will be used. The elementary exchanges satisfy the following property.

Let T be a smallest spanning tree having k edges incident to r. If there are such edges $e \in T \cap R$ and $f \notin T \cup R$ that $T - e + f$ is a spanning tree and $w(f) - w(e)$ is the smallest among all such pairs, then $T - e + f$ is the smallest spanning tree having $k - 1$ edges incident to r.

For finding exchanges a system of priority queues is employed. Consider an edge $e \in T \cap R$. An exchange that removes e from T adds an edge f which joins two connected components of $T - e$. A priority queue $F(e)$ stores all such edges f. The priority of f is its weight $w(f)$. Thus the smallest edge f is easily found. Another priority queue X stores the exchanges (e, f), where for any edge $e \in T \cap R$, f is the smallest edge that can replace e. The priority of (e, f) is $w(f) - w(e)$. Therefore the smallest exchange is easily found.

Finally, let there exist two edges e and e' from R which can be replaced with the same edge f and let (e, f) be the current exchange. Then it is necessary to remove the edge f from $F(e)$ and from $F(e')$ and to remove the exchange (e', f') containing e' from X. Thereafter we need to merge $F(e)$ and $F(e')$ into a new priority queue $F(e')$ and choose a new smallest edge f' in it. Moreover, we need to add the new exchange (e', f') to the queue X.

4. Basic procedures

For performing the Gabow algorithm on conventional sequential computers the Cheriton–Tarjan algorithm is used both for finding a minimal spanning forest and for finding a smallest spanning tree which includes all edges incident to r. However, on associative parallel processors for the same goals we utilize the Prim–Dijkstra algorithm.

In the STAR-machine matrix memory we represent a graph G in the form of association of the matrices *left*, *right* and *weight* in which each edge $e = (p, q)$ corresponds to the triple $\langle p, q, w(p, q)\rangle$, where $p \in$ *left*, $q \in$ *right*, $w(p, q) \in$ *weight*.

Remark 3. Without loss of generality we assume that in the matrix memory any edge incident to r has the form: $\langle r, s, w(r, s)\rangle$. Moreover, we assume that all the edges incident to r occupy the first k rows in the graph representation.

At first, let us enumerate a group of procedures which will be used for finding a smallest spanning tree including all edges incident to r. These procedures employ the given global slice X for indicating by '1' the row positions being used in the corresponding procedure.

We employ the procedure MERGE(T, X, F) for writing into the resulting matrix F those rows of the given matrix T which correspond to positions '1' in the slice X.

We apply the procedures MATCH, MIN and MAX from [12]. The procedure MATCH(T, X, w, Z) defines the row positions in the given matrix T coinciding with the given w. Its result is the slice Z in which $Z(i) =$ '1' if $T(i)=w$. The procedure MIN(T, X, Z) defines the row positions in the given matrix T where the minimal element is located. It returns the slice Z in which $Z(i) =$ '1' if $T(i)$ is the minimal matrix element. The procedure MAX(T, X, Z) is defined by analogy with MIN.

For performing the arithmetic operations we utilize the procedures ADDC and SUBTV. The procedure ADDC(T, X, w, F) adds the binary word w to the rows of the matrix T and writes the result into the matrix F. The procedure SUBTV(T, Q, X, F) writes into the matrix F the result of subtraction of the matrix Q from the matrix T.

For finding a minimal spanning tree of a connected component we employ the procedure MSTC(*left*, *right*, *weight*, $S1, S, R$) from [13]. It uses the slice $S1$ for indicating the positions of the graph edges among which a connected component is constructed. The procedure returns the resulting slices S and R. In the slice R we store positions of the edges belonging to the minimal spanning tree of the connected component. In the slice S we save positions of those edges from $S1$ which are not incident to the vertices included into the minimal spanning tree.

We construct a minimal spanning forest of $G - r$ by means of the procedure FOREST(*left*, *right*, *weight*, Z, U) from [13]. It uses the slice Z for indicating positions of the edges among which the connected components are looked for. The procedure returns the resulting slice U in which positions of edges belonging to the minimal spanning forest are indicated by '1'.

Remark 4. Correctness of the procedure `MSTC` is established by induction on the number of edges included in the minimal spanning tree. Correctness of the other procedures is verified by induction either on the number of connected components (for the procedure `FOREST`) or on the number of bit columns in the matrix T.

The procedure `EXCHANGE`$(left, right, j, Z, R, U)$ returns the slice U in which positions of all the replacing edges for an edge $e = (r, s)$ from the j-th position are indicated by '1'. It uses the slice Z for indicating positions of edges deleted from the minimal spanning forest of $G - r$ and the slice R for indicating positions of edges which belong to the smallest spanning tree containing all the edges incident to r.

Remark 5. Let us briefly explain a construction realized in the procedure `EXCHANGE`. For any edge $e = (r, s)$ belonging to the j-th position we construct a path from its right vertex (s) to a terminal vertex by means of the procedure `MATCH`. Knowing the slice R, for every new vertex w added to this path we verify whether there are such edges, incident to w, whose positions are indicated by '1' in the slice Z. Positions of such edges are accumulated in U.

5. Associative algorithm

At first, consider the main constructions.

The first construction allows one by means of the Prim–Dijkstra algorithm to obtain the smallest spanning tree including all the edges incident to r.

Construction 1. At first, we define the maximal weight ($w1$) of the edges incident to r. Then we construct a new matrix *cost* from the matrix *weight* as follows. Weights of the edges, not incident to r, are increased by the value of $w1$. Now, we perform the procedure `MSTC` using the matrix *cost* instead of the matrix *weight*.

For performing the first construction we utilize the STAR procedures `MAX`, `ADDC`, `MERGE` and `MSTC`.

In the second construction for any edge e belonging to the j-th position in the graph representation we define the position of its replacing edge f.

Construction 2. For any edge $e=(r,s)$ from the j-th position by means of the procedure EXCHANGE(*left*, *right*, j, Z, R, U) we define in the slice U positions of all the edges which can replace it. If there are such positions, among them using the procedure MIN we define the position i of the replacing edge f having the minimal weight. The position j of the edge e is saved in the slice W and the position i of its replacing edge f is stored in the j-th component of the array A. At last, we carry out the statement $cost(j) := cost(i)$.

For performing this construction we apply the procedures MATCH and MIN.

The third construction is employed for selecting the current exchange.

Construction 3. After performing the procedure SUBTV(*cost*, *weight*, W, *cost*1) in the matrix *cost*1 we obtain the augments of the rows corresponding to positions '1' in the slice W. For defining the current exchange position we carry out the procedure MIN(*cost*1, W, X) and the statement $m :=$ FND(X). Thus, the pair $(m, A[m])$ is the current exchange. For finding the replacing edge position we fulfil the statement $i := A[m]$. Now, we perform the exchange as follows. We delete edge position m both from the slice W (using the statement $W(m) :=$ '0') and from the resulting slice R (by means of the statement $R(m) :=$ '0'). Then we add the edge position i to the slice R (with the use of the statement $R(i) :=$ '1').

In the fourth construction we define whether there are two edges incident to r having the same replacing edge.

Construction 4. Assume that we have performed the current exchange (m, i) by means of Construction 3. Knowing the position m of the deleted edge e we define the column number p in the matrix *change* where positions of its replacing edges are stored. We delete the i-th position from the p-th column. Now, if there exists another edge incident to r with the same replacing edge, then in the i-th row of the matrix *change* a bit '1' must be necessary. Knowing the i-th row by means of the operation STEP we define the column number (say l) in the matrix *change* whose i-th bit is '1' and replace this bit with '0'. Finally, we perform the disjunction between the p-th column and the l-th column and we write the result into the l-th column of the matrix *change*. This is equivalent to removing the occurrence of the edge f from $F(e')$ in the Gabow algorithm.

The associative algorithm consists of the following three stages.

At the first stage, using the procedure FOREST, we construct a minimal spanning forest of $G - r$. Using Construction 1 we obtain the matrix *cost*. Now, knowing positions of edges incident to r and positions of edges included in the minimal spanning forest by means of the procedure MSTC we construct the smallest spanning tree T containing all edges incident to r. Finally, positions of edges from the minimal spanning forest not belonging to T are stored into a slice Z.

At the second stage, using Construction 2, we get the slice W and the array A forming the list of pairs $(j, A[j])$. Moreover, for any deleted edge from the j-th position in the current p-th column of the matrix *change* we save positions of its replacing edges. Finally, by means of the statements $B[j] := p$ and $C[p] := j$ we determine a one-to-one correspondence between j and p. Note that the matrix *change* will be used at the next stage.

At the third stage, using Construction 3, we carry out the current exchange (m, i). If the added edge from the i-th position is a replacing edge for another edge e' incident to r, then by means of Construction 4 we update the l-th column of the matrix *change*, where positions of new replacing edges for e' will be written. Now, knowing the l-th column with the use of the procedure **MIN** we define the position (say q) of a new replacing edge for e' having the minimal weight. For defining the position in the graph representation, where the edge e' is located, we perform the statement $h := C[l]$. Finally, we fulfil the statement $cost(h) := cost(q)$.

Note that this stage is repeated until $\mathtt{deg}(r) > t$.

In Appendix we describe the representation of the Gabow algorithm on the STAR-machine using the procedure SSTEQ. Its correctness is established by means of the following theorem.

Theorem. *Let a graph G be represented by the association of the matrices left, right and weight. For a given vertex r let* node *be its binary code, k be its degree and t be a constraint degree. Then, using the procedure* SSTEQ(*left*, *right*, *weight*, *node*, k, t, R) *we construct a minimal spanning tree T_S, whose edge positions are indicated by* '1' *in the slice R and T_S satisfies the degree constraint.*

For proving the theorem we employ a group of the following claims.

Claim 1. At the first stage after performing the procedure MSTC(*left*, *right*, *cost*, X, U, R) a minimal spanning tree T having k edges incident to r is constructed, and the edge positions included in T are indicated by '1' in the resulting slice R.

In view of Construction 1 and Remark 4 it is necessary to prove that the minimal spanning tree T includes k edges incident to r. This is checked by induction on the number of edges incident to r.

Remark 6. Owing to Remark 3 after performing the first stage for any j, $1 \leq j \leq k$, we get $cost(j) = weight(j)$.

Claim 2. At the second stage, the current edge from the j-th position of the slice Y is indicated by '1' in the slice W if there is at least one replacing edge for it. The position of the replacing edge having the minimal weight is stored in the j-th component of the array A.

Claim 3. At the third stage the execution of each current exchange decreases the number of edges incident to r included in the minimal spanning tree.

Claim 4. At the third stage by means of the loop **if** SOME(v) **then** the case when two edges incident to r have a common replacing edge is realized.

Now, let us evaluate the time of executing the procedure SSTEQ. Following [5], assume that any elementary operation of the STAR-machine needs one unit of time. Therefore we measure *time complexity* of an algorithm by counting all elementary operations performed in the worst case.

In view of [7] and Remark 4, the procedure SSTEQ takes $O(n \log n)$ time since its first stage takes $O(n \log n)$ time, the second stage requires $O((n - \mathrm{deg}(r)) \log n)$ time and the third stage takes $O(\mathrm{deg}(r) - t)$ time.

6. Conclusions

In this paper, by means of the STAR-machine we have presented the associative algorithm for finding the smallest spanning tree with a degree constraint which is based on the Gabow algorithm. To this end we have suggested new techniques. In Appendix the algorithm is represented as the STAR procedure SSTEQ whose correctness is verified by means of Theorem. We have obtained that the associative algorithm for finding the smallest spanning tree with a degree constraint takes the same time as the minimal spanning tree algorithm which is proportional to $n \log n$ of the STAR-machine elementary operations.

References

[1] K.E. Grosspietsch, *Associative Processors and Memories: A Survey*, IEEE, Micro, June, 1992, 12–19.

[2] Y.I. Fet, *Vertical Processing Systems: A Survey*, IEEE, Micro, February, 1995, 65–75.

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

[4] J. Potter, J. Baker, A. Bansal, S. Scott, C. Leangsuksun, C. Asthagiri, *ASC – An Associative Computing Paradigm*, IEEE Computer: Special Issue on Associative Processing, **11**, 1994, 19–25.

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

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

[7] 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.

[8] H. Gabow, *A Good Algorithm for Smallest Spanning Trees with a Degree Constraint*, Networks, **8**, No. 3, 1978, 201–208.

[9] H.N. Gabow, R.E. Tarjan, *Efficient Algorithms for a Family of Matroid Intersection Problems*, J. Algorithms, **5**, 1984, 80–131.

[10] 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.

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

[12] A.S. Nepomniaschaya, *Investigation of Associative Search Algorithms in Vertical Processing Systems*, Proc. of the Intern. Conf. "Parallel Computing Technologies", Obninsk, Russia, 1993, 631–641.

[13] 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", Novosibirsk, Academgorodok, Russia, June 25–28, 1996, 87–92.

Appendix

```
proc SSTEQ(left, right, weight: table; k, t: integer; node: word;
            var R: slice(left));
/* The variable node stores the binary code of the given vertex (r),
   k is its degree and t is the degree constraint. */
var X, Y, Z, U, W: slice(left); g, h, i, j, l, m, p: integer;
   node1, v, w1: word; A, B, C: array [1 : k] of integer;
   cost, cost1, change: table;
begin
/* The first stage. */
      CLR(W); SET(X); p := 0; MATCH(left, X, node, Y);
```

```
/* In the slice $Y$ we indicate by '1' positions of edges incident to $r$. */
    $X := not\ Y$; FOREST($left, right, weight, X, U$);
/* In the slice $U$ we flag by '1' positions of edges included in the
   minimal spanning forest of $G - r$. */
    MAX($weight, Y, Z$); $j :=$ FND($Z$); $w1 := weight(j)$;
/* The maximal weight of edges incident to $r$ is stored in $w1$. */
    ADDC($weight, U, w1, cost$);
/* For any $i$-th edge included in the minimal spanning forest we have
   $cost(i) = weight(i) + w1$. */
    if size($cost$) > size($weight$) then with($W, weight$);
/* We attach the slice $W$ left of the matrix $weight$ if necessary. */
    MERGE($weight, Y, cost$);
/* In the rows of the matrix $cost$ which correspond to positions
   of edges incident to $r$ we write the corresponding rows from
   the matrix $weight$. */
    $X := Y\ or\ U$; MSTC($left, right, cost, X, U, R$);
/* We have constructed the smallest spanning tree whose edge positions
   are indicated by '1' in $R$ using the set of edges incident to $r$ and the
   set of edges included in the minimal spanning forest of $G - r$. */
    $Z := X\ and\ (not\ R)$;
/* In the slice $Z$ we indicate by '1' positions of those edges from the
   minimal spanning forest which do not belong to $R$. */

/* The second stage. As a result of the first stage we use the matrix
   $cost$ and the slices $R, Y$ and $Z$. */
      while SOME($Y$) do
        begin $j :=$ STEP($Y$); EXCHANGE($left, right, j, Z, R, U$);
/* In the slice $U$ we indicate by '1' positions of the replacing edges
   for the edge $e = (r, s)$ from the $j$-th position. */
          if SOME($U$) then
            begin $W(j) :=$ '1'; MIN($cost, U, X$);
            $i :=$ STEP($X$); $A[j] := i$; $cost(j) := cost(i)$;
/* Position $j$ of the current edge $(r, s)$ is indicated by '1' in the slice $W$
   if there are replacing edges for it. Then in the $j$-th component of
   the array $A$ we store the position of the replacing edge having
   the minimal weight. */
            $p := p + 1$; $B[j] := p$; $C[p] := j$; col($p, change$) $:= U$;
/* In the $p$-th column of the matrix $change$ we indicate by '1' positions
   of all replacing edges for the current edge $(r, s)$. */
            end;
        end;
```

```
/* The third stage. As a result of the second stage we utilize the slice
   $W$, the arrays $A, B, C$ and the matrices change and cost. */
      for $j := k - 1$ downto $t$ do
        begin SUBTV(cost, weight, W, cost1);
  /* In any matrix cost1 row corresponding to the position '1' in $W$
     there is an augment. */
          MIN(cost1, W, X); m:=FND(X);
  /* We have located the position $m$ of the current exchange. */
          $W(m)$ := '0'; $i := A[m]$; $R(i)$ := '1'; $R(m)$ := '0';
  /* The edge (say $e$) from the $m$-th position is replaced with the edge
     (say $f$) from the $i$-th position. */
          $p := B[m]$;
  /* We have defined the number of the matrix change column $p$ in which
     there are positions of all the replacing edges for the edge $e$. */
          Y:=col(p, change); $Y(i)$ := '0'; col(p, change) := Y;
  /* From the $p$-th column of the matrix change we delete the occurrence
     of the edge $f$. */
          v := change(i); if SOME(v) then
            begin
  /* It is a case when another edge incident to $r$ has the replacing
     edge $f$. */
              l := STEP(v); h := C[l]; X := col(l, change); X := X or Y;
              col(l, change) := X;
  /* We merge the $p$-th and the $l$-th columns of the matrix change. */
              MIN(cost, X, Z); g := FND(Z); A[h] := g;
  /* For the edge from the position $h$ we have defined the position $g$ of
     its new replacing edge. */
              cost(h) := cost(g)
            end;
          CLR(Y); col(p, change) := Y
  /* We set all the components of the $p$-th column to '0'. */
        end;
end
```