Bull. Nov. Comp. Center, Comp. Science, 32 (2011), 93–106

Decremental associative algorithm for updating the shortest paths tree

A. S. Nepomniaschaya

Abstract. The paper proposes an efficient associative algorithm for dynamic update of the shortest paths tree of a directed weighted graph after deletion of an edge. To this end, we use the STAR–machine that simulates the run of associative (content addressable) parallel systems of the SIMD type with bit–serial (vertical) processing. We provide a data structure that allows us to perform access to data by contents. On the STAR–machine, the associative algorithm is represented as the main procedure DeleteArcSPT that uses a group of auxiliary procedures. By means of the auxiliary procedures, we execute some parts of the associative parallel algorithm for dynamic update of the shortest paths tree after deleting an arc from the graph. We prove correctness of the procedure DeleteArcSPT and all its parts. On the STAR–machine, this procedure takes $O(hk)$ time, where h is the number of bits required for coding the maximum of the shortest paths weights and k is the number of vertices, whose shortest paths change after deleting an edge from the given graph.

1. Introduction

In many applications, graphs are subject to discrete changes, such as insertions and deletions of edges or vertices. The objective of a dynamic algorithm is to efficiently update the solution to a problem after dynamic changes rather than to recompute the entire graph from scratch each time.

The dynamic version of the single source shortest paths problem consists in updating the shortest paths information after every change on the graph. The most general types of update operations for the single source shortest paths problem include insertions and deletions of edges and update operations on the edge weights. An algorithm is called *fully dynamic* if arbitrary sequences of the above operations are allowed, and it is called *partially* (*semi−*) *dynamic* if only one type of the update is allowed. A partially dynamic algorithm is called *incremental* if it supports only insertions of edges, while it is called *decremental* if it supports only deletions of edges.

In the case of arbitrary real edge weights, Ramalingam and Reps [11, 12] devise fully dynamic algorithms for updating the single source shortest paths using the output bounded model. In this model, the running time of an algorithm is analyzed in terms of the output change rather than the input size. The authors assume that the graph has no negative–length cycles before and after the input update. Frigioni et al. [3] study the semi–dynamic

single source shortest paths problem for both directed and undirected graphs with positive real edge weights in terms of the output complexity. Frigioni et al. [4] propose fully dynamic algorithms for updating the distances and the shortest paths tree (SPT) in either a directed or an undirected graph with positive real edge weights under arbitrary sequences of edge updates. The cost of the update operations is given as a function of the number of output updates by using the notion of k-bounded accounting function. Frigioni et al. [5] propose the fully dynamic solution for the problem of updating the shortest paths from a given source in a directed graph with arbitrary edge weights. The authors devise a new algorithm for performing edge deletions and weight increases that explicitly deals with zero–length cycles. In [6], we propose an efficient parallel implementation of the Ramalingam *decremental* algorithm [11] for dynamic update of the shortest-paths subgraph $SP(G)$ that consists of all shortest paths from every vertex of a given directed graph G to the sink. Our model of computation (the STAR–machine) simulates the run of associative (content addressable) parallel systems of the SIMD type with bit–serial (vertical) processing. The associative version of this algorithm is given as the procedure DeleteArc, whose correctness is proved. We obtain that this procedure takes time proportional to the number of vertices, for which the shortest paths to the sink change after deleting an edge from $SP(G)$. Following [2], it is assumed that each elementary operation of the STAR–machine (its microstep) takes one unit of time. In [8], we propose an efficient parallel implementation of the Ramalingam *incremental* algorithm for dynamic update of the single-sink shortest-paths subgraph $SP(G)$. The associative version of the Ramalingam incremental algorithm is given as the procedure InsertArc, whose correctness is proved. We obtain that this procedure takes the same $O(hk)$ time as in the case of the procedure DeleteArc. We also present the main advantages of the associative version of the Ramalingam incremental algorithm.

In this paper, we propose an efficient associative algorithm for dynamic update of the SPT of a directed weighted graph G after deletion of an edge. To this end, we propose a data structure that allows us to perform access to data by contents. On the STAR–machine, the associative algorithm is represented as the main procedure DeleteArcSPT that uses a group of auxiliary procedures. By means of the auxiliary procedures, we execute some parts of the associative parallel algorithm for updating SPT after deleting an arc from G. We prove correctness of the procedure DeleteArcSPT and all its parts. We obtain that this procedure takes $O(hk)$ time, where h is the number of bits required for coding the maximum of the shortest paths weights and k is the number of vertices, whose shortest paths change after deleting an edge from G.

2. An associative parallel machine model

In this section, we first recall the main operations of the STAR–machine. The description of the model is given, for example, in [7].

Let us present some elementary operations and a predicate for slices.

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

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

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

$Y(i)$ selects the i-th component of Y;

FND(Y) returns the number i of the first (the uppermost) $'1'$ of Y, $i \geq 0$;

STEP(Y) returns the same result as FND(Y), then resets the first $'1'$ found to $'0'$;

CONVERT(Y) returns a row, whose every i-th bit coincides with $Y(i)$. It is applied when a row of one matrix is used as a slice for another matrix.

The operations FND(Y), STEP(Y), and CONVERT(Y) are used only as the right part of the assignment statement, while the operation $Y(i)$ is used as both the right part and the left part of the assignment statement.

To execute the data parallelism, we introduce in the usual way the bitwise Boolean operations: $X\,and\,Y$, $X\,or\,Y$, $not\,Y$, $X\,xor\,Y$. We also use a predicate SOME(Y) that results in **true** if there is at least a single bit $'1'$ in the slice Y. For simplicity, the notation $Y \neq \emptyset$ means that the predicate SOME(Y) results in **true**.

Note that the predicate SOME(Y) and all operations for the type **slice** are also performed for the type **word**.

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

ROW(i, T) returns the i-th row of the matrix T;

COL(i, T) returns its i-th column.

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

Now, we recall a group of basic procedures [9, 10] implemented on the STAR–machine which will be used later on. These procedures use the given slice X to indicate with $'1'$ the row positions used in the corresponding procedure. In [9, 10], we have shown that the basic procedures take $O(r)$ time each, where r is the number of bit columns in the corresponding matrix.

The procedure MATCH(T, X, w, Z) determines the positions of the rows of the matrix T that coincide with the given pattern w. It returns the slice Z, where $Z(i) =' 1'$ if and only if ROW(i, T) $= w$ and $X(i) =' 1'$.

The procedure MIN(T, X, Z) finds the positions of rows in the given matrix T where the minimal element is located. These positions are marked with $'1'$ in the result slice Z.

The procedure SETMIN(T, F, X, Z) finds the positions of the rows of the matrix T that are less than the corresponding rows of the matrix F. It returns the slice Z, where $Z(i) =' 1'$ if ROW$(i, T) <$ ROW(i, F) and $X(i) =' 1'$.

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

The procedure TMERGE(T, X, F) writes the rows of the matrix T, that correspond to positions $'1'$ in the slice X, into the matrix F. Other rows of the matrix F are not changed.

The procedure ADDV(T, F, X, R) writes into the matrix R the result of parallel addition of the corresponding rows of matrices T and F, whose positions are selected with $'1'$ in the slice X. This algorithm uses the table 5.1 from [2].

The procedure ADDC(T, X, v, F) adds the binary word v to the rows of the matrix T selected with $'1'$ in X, and writes down the result into the corresponding rows of the matrix F. Other rows of the matrix F are set to zero.

3. Preliminaries

Let $G = (V, E)$ be a *directed weighted graph* with the set of vertices $V = \{1, 2, \ldots, n\}$ and the set of directed edges (arcs) E. Let $wt(e)$ denote a function that assigns a weight to every edge e. We assume that $|V| = n$ and $|E| = m$. We also assume that all arcs have a non–negative weight and $wt(u, v) = \infty$ if $(u, v) \notin E$.

For the considered problem, the infinity will be implemented by the value $\sum_{i=1}^{n} c_i$, where c_i is the maximum weight of arcs outgoing from the vertex i. Let h be the number of bits for coding this sum.

An arc e directed from u to v is denoted by $e = (u, v)$, where u is the $father$ of e and v is its *son*.

An *adjacency matrix* $Adj = [a_{ij}]$ of a directed graph G is an $n \times n$ Boolean matrix, where $a_{ij} = 1$ if and only if there is an arc from the vertex i to the vertex j in the set E.

The *shortest path* from v_1 to v_k in G is a finite sequence of vertices $v_1, v_2, \ldots, v_k$, where $(v_i, v_{i+1}) \in E$ $(1 \leq i < k)$ and the sum of weights of the corresponding arcs is minimal. Let $dist(l)$ denote the length of the shortest path from v_1 to l.

A *tree of the shortest paths* T_s is a connected acyclic subgraph of G which contains all graph vertices and is such that the path from s to any vertex v in T_s is the shortest path from s to v in G. A $leaf$ in a tree is a vertex that has no outgoing arcs. The *height* of a tree is the number of arcs in the longest path from the root to a leaf.

Let an arc (i, j) be deleted from the graph G. A vertex y is called $affected$ after deleting the arc (i, j) from the shortest paths tree T_s if there is no path from s to the vertex y.

4. Associative parallel algorithm for updating the shortest paths tree

In this section, we first propose the data structure. Then we provide an associative parallel algorithm for dynamic update of the SPT after deleting an arc from a given graph G with a selected source vertex s called the *root*. We observe that the initial shortest paths tree is obtained by means of the classical Dijkstra algorithm [1].

We will employ the following data structure:

– an $n \times n$ adjacency matrix G, whose every i-th column saves with the bit $'1'$ the sons of the vertex i;

– an $n \times n$ adjacency matrix SPT, whose every i-th column saves with the bit $'1'$ the sons of the vertex i that belong to the shortest paths tree;

– an $n \times hn$ matrix $Weight$ that contains the arc weights as entries. It consists of n fields having h bits each. The weight of an arc (i, j) is written in the j-th row of the i-th field;

– an $n \times hn$ matrix $Cost$ that contains the arc weights as entries. It consists of n fields having h bits each. The weight of an arc (i, j) is written in the i-th row of the j-th field;

– an $n \times h$ matrix $Dist$, whose every i-th row saves the shortest distance from the root s to the vertex i;

– a slice $AffectedV$ that saves with $'1'$ the positions of all affected vertices.

We observe that the i-th field of the matrix $Weight$ saves the weights of arcs *outgoing* from the vertex i, while the i-th field of the matrix $Cost$ saves the weights of arcs *entering* the vertex i. Moreover, every j-th row of the matrix G saves with $'1'$ different fathers of the vertex j.

Our algorithm for the dynamic update of the shortest paths tree is based on the main idea of the Ramalingam decremental algorithm. Let an arc (i, j) be deleted from G and T_s. We observe that every vertex in a tree has a single father. Therefore after deleting the arc (i, j) from T_s, the vertex j becomes an affected one. Moreover, all vertices belonging to the subtree T_j are also affected and there is no another affected vertex in T_s.

We first propose an associative parallel algorithm for finding the affected vertices and affected arcs (say *Algorithm 1*) obtained after deleting the arc (i, j) from the shortest paths tree.

The idea underlying this algorithm is as follows. We determine the vertices that belong to the connected component including the vertex j. The

associative parallel algorithm uses the matrix SPT, the slice $AffectedV$ and an auxiliary slice, say Z. It performs the following steps.

Step 1. Simultaneously include the sons of the vertex j into the slices $AffectedV$ and Z. Include the vertex j into the slice $AffectedV$. Then simultaneously delete from the matrix SPT the arcs leaving the vertex j.

Step 2. While $Z \neq \emptyset$, perform the following actions:

– delete the position of the first bit $'1'$ (say r) from the slice Z;

– by means of a slice, save the sons of the vertex r;

– delete from the matrix SPT the arcs leaving the vertex r;

– include all sons of the vertex r into the slices $AffectedV$ and Z.

On the STAR–machine, this algorithm is implemented as the procedure AffectedVertSPT.

An associative parallel algorithm for computing new distances from the root to all affected vertices (say *Algorithm 2*) uses the slice $AffectedV$ and the matrices G, $Cost$, and $Dist$. It runs as follows.

While $AffectedV \neq \emptyset$, determine a new distance from the root to every affected vertex by means of the following steps.

Step 1. Select the position of the current vertex k in the slice $AffectedV$ and mark it with zero.

Step 2. Compute in parallel the weight of every path in the matrix G from the root to the vertex k that does not include affected vertices.

Step 3. Select the minimal distance from the root to k and write it down into the k-th row of the matrix $Dist$.

On the STAR–machine, this algorithm is implemented as the procedure NewDistSPT.

An associative parallel algorithm for updating the arcs leaving an affected vertex k (say *Algorithm 3*) uses the slice $AffectedV$, and the matrices $Weight$ and $Dist$. It performs the following steps.

Step 1. Knowing the slice $AffectedV$ and the distance from the root to the vertex k, simultaneously save the weights of paths in G from the root to every affected vertex r that is the son of k.

Step 2. By means of a slice (say Y), save *positions* of those sons r of the vertex k, for which $dist_{new}(r) < dist_{old}(r)$. Then write $dist_{new}(r)$ in the corresponding rows of the matrix $Dist$.

On the STAR–machine, this algorithm is implemented as the procedure OutgoingArcsSPT.

Now we provide the associative parallel algorithm for dynamic update of the shortest paths tree after deleting the arc (i, j) from G. It performs the following steps.

Step 1. Delete the *position* of the arc (i, j) from the matrix G. If $(i, j) \notin SPT$, then go to exit. Otherwise, delete the *position* of this arc from the matrix SPT.

Step 2. Construct the slice $AffectedV$ and delete affected arcs from the matrix SPT.

Step 3. Determine new distances from the root to all affected vertices and write them in the corresponding rows of the matrix $Dist$.

Step 4. While $AffectedV \neq \emptyset$, update affected vertices taking into account their new distances from the root as follows:

– knowing the slice $AfectedV$ and the matrix $Dist$, determine the position of an affected vertex k having the minimum distance from the root and delete k from the slice $AffectedV$;

– determine the $father$ p of the vertex k and include the position of the arc (p, k) into the matrix SPT;

– recompute the distances from the root to those affected vertices r, for which the vertex k is their father in G and $dist_{new}(r) < dist_{old}(r)$. Write $dist_{new}(r)$ in the corresponding rows of the matrix $Dist$.

On the STAR–machine, it is implemented as the procedure DeleteArcSPT.

5. Implementation of the associative algorithm for updating the *SPT* on the STAR–machine

In this section, we first provide three auxiliary procedures. Then we propose the procedure DeleteArcSPT.

Let us consider the procedure AffectedVertSPT. Knowing the vertices i and j and the current matrix SPT, it returns the slice $AffectedV$ and deletes the positions of affected arcs from the matrix SPT.

```
procedure AffectedVertSPT(i,j: integer; var SPT: table;
  var AffectedV: slice(G));
/* The arc (i,j) has been deleted from G and SPT. */
var X,Y,Z: slice(G);
  r: integer;
1. Begin CLR(Y); AffectedV:=COL(j,SPT);
2.  Z:=COL(j,SPT); AffectedV(j):='1';
3.  COL(j,SPT):=Y;
/* We write zeros into the j-th column of the matrix SPT. */
4.  while SOME(Z) do
5.    begin r:=STEP(Z);
6.      X:=COL(r,SPT); COL(r,SPT):=Y;
/* The arcs leaving the vertex r are deleted from SPT. */
7.      AffectedV:=AffectedV or X;
8.      Z:=Z or X;
9.    end;
10. End.
```

Lemma 1. *Let an arc (i, j) be deleted from the shortest paths tree T_s. Then the procedure* AffectedVertSPT *returns the slice AffectedV, where positions of affected vertices are marked with $'1'$. Moreover, it deletes from the SPT all arcs outgoing from every affected vertex.*

Proof. (Sketch.) To prove the lemma, we have to show that after performing the procedure AffectedVertSPT every vertex l from the subtree T_j belongs to the slice $AffectedV$ and the l-th column of the matrix SPT consists of zeros. We prove this by induction in terms of the height q of the subtree T_j.

Basis is checked for $q = 1$. After performing lines 1–3, the slice Y consists of zeros, the slice Z saves the sons of the vertex j, the slice $AffectedV$ saves the vertex j along with its sons, and the j-th column of the matrix SPT consists of zeros. Now we perform the cycle `while SOME(Z) do` (lines 5–9). After performing lines 5–6, the slice $X = \emptyset$ because any vertex r marked with $'1'$ in Z is a leaf in T_j and the r-th column of the matrix SPT consists of zeros. After performing lines 7–8, none new vertex is included into the slices $AffectedV$ and Z. After updating the last vertex from Z, we obtain $Z = \emptyset$. Therefore we go to the exit.

Step of induction. Let the assertion be true for any subtree T_l of the height $q \geq 1$. We prove the lemma for the subtree T_j of the height $q + 1$. By the inductive hypothesis, after updating any subtree T_l of the height q, all vertices of T_l are included into the slice $AffectedV$ and all arcs leaving every vertex of T_l are deleted from the matrix SPT.

Consider the update of the subtree T_j of the height $q + 1$. By analogy with the basis, after performing lines 1–3, the slice $AffectedV$ saves the vertex j along with its sons and all arcs leaving the vertex j are deleted from the matrix SPT. Now every son r of the vertex j in T_j becomes a root of a subtree T_r, whose height is no greater than q. Therefore we apply the inductive hypothesis to every such a subtree. After updating the last vertex in the slice Z, the slice $AffectedV$ includes all vertices of the subtree T_j and all arcs leaving every vertex of T_j are deleted from the matrix SPT. This completes the proof.

Now we provide the procedure NewDistSPT that determines new distances from s to all affected vertices. It returns the updated matrix $Dist$.

```
procedure NewDistSPT(h: integer; G: table; Cost: table;
  AffectedV: slice(G); var Dist: table);
var v: word(G); v1: word(Dist);
  X,Z,Z1: slice(G);
  W1,W2: table;
  k,r: integer;
1. Begin X:=AffectedV;
```

```
 2.  while SOME(X) do
 3.    begin k:=FND(X); v:=ROW(k,G);
 4.      Z:=CONVERT(v);
 /* The slice Z saves fathers of arcs entering k in G. */
 5.      Z1:=Z and (not AffectedV);
 /* The slice Z1 saves those vertices from Z that
   are not affected. */
 6.      TCOPY1(Cost,k,h,W1);
 7.      ADDV(Dist,W1,Z1,W2);
 /* W2 saves different distances from the root to k. */
 8.      MIN(W2,Z1,Z);
 9.      r:=FND(Z); v1:=ROW(r,W2);
10.      ROW(k,Dist):=v1;
 /* The new distance from the root to k is written
   in the k-th row of the matrix Dist. */
11.      X(k):='0';
12.    end;
13. End.
```

Lemma 2. *Let h be the number of bits for coding the infinity. Let the slice $AffectedV$ and the current matrices G, $Cost$, and $Dist$ be given. Then the procedure* NewDistSPT *returns the updated matrix $Dist$ that saves new distances from the root to all affected vertices.*

Proof. We prove this by induction on the number of affected vertices l.

Basis is checked for $l = 1$. After performing lines 1–4, the slice X is a copy of the slice $AffectedV$, $k = j$, and the slice Z saves the fathers of the vertex j in G. After performing lines 5–7, the matrix $W2$ saves the weights of different paths from the root to the vertex j that do not include affected vertices. After performing lines 8–10, we first determine the vertex r that belongs to the new shortest path from the root to j, then we write down the new distance in the j-th row of the matrix $Dist$. After fulfilling line 11, $X = \emptyset$, and we go to the exit.

Step of induction. Let the assertion be true for l ($l \geq 1$) affected vertices. We prove this for $l+1$ vertices. By the inductive assumption, after updating the first l affected vertices, their new distances from the root are written in the corresponding rows of the matrix $Dist$, and there is only a single affected vertex in the slice X. Further we reason by analogy with the basis.

This completes the proof.

Let us proceed to the procedure OutgoingArcsSPT. It uses the current updated vertex k, the slice $AffectedV$, and the matrices $Weight$ and $Dist$. The procedure returns the updated matrix $Dist$.

```
procedure OutgoingArcsSPT(h,k: integer; G: table; Weight: table;
  var Dist: table);
var v: word(Dist);
  W1,W2: table;
  Y,Z: slice(Dist);
1. Begin Z:=COL(k,G);
2.  TCOPY1(Weight,k,h,W1);
/* The matrix W1 saves the weights of arcs leaving k. */
3.  v:=ROW(k,Dist);
4.  ADDC(W1,Z,v,W2);
/* The matrix W2 saves the new distances from the root
  to the vertices p that belong to the arc (k,p). */
5.  SETMIN(W2,Dist,Z,Y);
6.  TMERGE(W2,Y,Dist);
7. End;
```

Lemma 3. *Let h be the number of bits for coding the infinity and k be the current updated vertex. Let the current matrices G, $Weight$ and $Dist$ be given. Then the procedure* OutgoingArcsSPT *maintains the matrix $Dist$, where the new distances from the root are written for the sons r of the vertex k whose $dist_{new}(r)$ is decreased.*

This lemma is proved by contradiction. Let an arc (k, r) belong to the graph G and $dist_{new}(r) < dist_{old}(r)$. However, after performing the procedure OutgoingArcsSPT, the r-th row of the matrix $Dist$ does not change. We prove that this cotradicts the execution of the procedure OutgoingArcsSPT.

Indeed, since $(k, r) \in G$, then $Z(r) =' 1'$ after performing line 1. After performing lines 2–4, the weight of the shortest path from the root to the vertex r, that includes the edge (k, r), is written into the r-th row of the matrix $W2$. By the assumption, $dist_{new}(r) < dist_{old}(r)$. Therefore $Y(r) =' 1'$ after fulfilling the basic procedure SETMIN (line 5). Hence, after performing line 6, the edge $dist_{new}(r)$ is written into the r-th row of the matrix $Dist$. This contradicts our assumption.

Finally, we proceed to the procedure DeleteArcSPT. Knowing the deleted arc (i, j) and the current matrices G, $Weight$, $Cost$, $Dist$, and SPT, the procedure returns the updated matrices G, SPT, and $Dist$ with the use of the above auxiliary procedures.

```
procedure DeleteArcSPT(i,j,h: integer; Weight,Cost: table;
  var G,SPT: table; var Dist: table);
/* The arc (i,j) has been deleted from the graph G. */
var k,r: integer;
  AffectedV,X,Y,Z: slice(G);
```

```
   W1,W2: table;
   v: word(G); v1: word(Dist);
   label 1;
 1. Begin X:=COL(i,G); X(j):='0';
 2.  COL(i,G):=X;
 /* The arc (i,j) is deleted from G. */
 3.  X:=COL(i,SPT);
 4.  if X(j)='0' then goto 1;
 5.  X(j):='0'; COL(i,SPT):=X;
 /* The arc (i,j) is deleted from SPT. */
 6.  AffectedVertSPT(i,j,SPT,AffectedV);
 /* This procedure returns the updated matrix SPT
   and the slice AffectedV. */
 7.  NewDistSPT(h,G,Cost,AffectedV,Dist);
 /* This procedure returns the updated matrix Dist. */
 8.  while SOME(AffectedV) do
 9.    begin MIN(Dist,AffectedV,Z);
 /* The slice Z saves positions of the matrix Dist rows,
   where the minimal value is located. */
10.       k:=FND(Z); AffectedV(k):='0';
 /* The vertex k is included into the shortest paths tree. */
11.       v:=ROW(k,G); Z:=CONVERT(v);
12.       X:=Z and (not AffectedV);
 /* The slice X saves fathers of arcs entering k in G
   that are not affected. */
13.       v1:=ROW(k,Dist);
14.       TCOPY1(Cost,k,h,W1);
15.       ADDV(Dist,W1,X,W2);
 /* The matrix W2 saves different distances in G
   from the root to the vertex k. */
16.       MATCH(W2,X,v1,Y); r:=FND(Y);
17.       Y:=COL(r,SPT); Y(k):='1';
18.       COL(r,SPT):=Y;
 /* The arc (r,k) is included into the matrix SPT. */
19.       OutgoingArcsSPT(h,k,Weight,AffectedV,Dist);
20.    end;
21. 1: End;
```

Theorem. *Let a directed weighted graph be given as an adjacency matrix G and a matrix Weight. Let the matrices Cost, SPT, and Dist and the number of bits h for coding the infinity be given. Let an arc (i,j) be deleted from the graph. Then after performing the procedure* DeleteArcSPT*, this arc is deleted from the matrices G and SPT. Moreover, the matrices*

SPT and Dist are updated according to Algorithms 1–3.

Proof. (Sketch.) We prove this by induction in terms of the number q of affected vertices that appear after deleting the arc (i, j) from the shortest paths tree.

Basis is proved for $q = 1$. One can check that after performing lines 1–5 the arc (i, j) is deleted from the matrices G and SPT. After performing line 6, in view of Lemma 1, the slice $AffectedV$ saves the position of the affected vertex j and all arcs, leaving this vertex, are deleted from the matrix SPT. Observe that the j-th column of the matrix SPT consists of zeros before applying Lemma 1, because j is a single affected vertex in T_s. After performing line 7, in view of Lemma 2, $AffectedV(j) =' 1'$ and the new distance from the root to j is written in the j-th row of the matrix $Dist$. Since $AffectedV \neq \emptyset$, we perform the cycle `while SOME(AffectedV) do` (line 8). After fulfilling lines 9-10, we have $k = j$, $AffectedV = \emptyset$ and the vertex j is included into the tree. To determine the father of the vertex j in T_s, we first save the fathers of arcs entering j in G that are not affected (lines 11–12). After performing lines 13–15, we determine different distances in G from the root to j. After performing line 16, we select the distance from the root to j, where the minimum value $v1$ is achieved. After fulfilling lines 17–18, the new arc (r, j) is included into the matrix SPT.

Step of induction. Let the assertion be true when no more than $q \geq 1$ affected vertices are updated in the given graph. We will prove the assertion for $q + 1$ affected vertices.

One can immediately check that, after performing lines 1–7, the arc (i, j) is deleted from the matrices G and SPT, the slice $AffectedV$ saves the positions of $q + 1$ affected vertices, all affected arcs are deleted from SPT, and the new distances from the root to all affected vertices are written in the corresponding rows of the matrix $Dist$. Since $AffectedV \neq \emptyset$, we execute line 8.

After performing lines 9–10, we determine the position of the affected vertex k having the minimum new distance from the root and mark it with $'0'$ in the slice $AffectedV$. By analogy with the basis, after fulfilling lines 11–18, we first determine the father of the new arc (r, k) that belongs to the new shortest path from the root to the vertex k. Then we include this arc into the matrix SPT. After performing line 19, in view of Lemma 3, we write new distances for those sons of the vertex k which are affected vertices and $dist_{new}(r)$ is decreased.

Now, there are only q affected vertices, whose positions are marked with bit $'1'$ in the slice $AffectedV$. By the inductive assumption, after updating q affected vertices, the new distance from the root to every affected vertex r is written into the r-th row of the matrix $Dist$ and a new arc, entering the vertex r, is included into the matrix SPT. Moreover, each time after

including a current vertex p into T_s, we recompute the weights of paths from the root to those sons of the vertex p which are affected vertices and their new distance from the root is decreased. This assures to select correctly the shortest path from the root to every affected vertex that has to be included into T_s later.

This completes the proof.

Let us evaluate the time complexity of the procedure DeleteArcSPT. We first evaluate the time complexity of the auxiliary procedures. The auxiliary procedure AffectedVertSPT takes $O(k)$ time because the cycle from line 4 is performed k times and every operation inside this cycle takes $O(1)$ time. The auxiliary procedure NewDistSPT takes $O(kh)$ time because the cycle `while SOME(X) do` (lines 2–11) is performed k times and inside this cycle the basic procedures require $O(h)$ time each. The auxiliary procedure OutgoingArcsSPT takes $O(h)$ time. In the procedure DeleteArc, the cycle `while SOME(AffectedV) do` (lines 8–20) takes $O(kh)$ time, because inside this cycle the basic procedures and the auxiliary procedure require $O(h)$ time each. Hence, the procedure DeleteArcSPT takes $O(kh)$ time.

6. Conclusions

We have proposed the efficient associative algorithm for dynamic update of the shortest paths tree after deletion of an arc from the given graph. On the STAR–machine having no less than n PEs, this algorithm is represented as the main procedure DeleteArcSPT that uses a group of auxiliary procedures. We prove correctness of the procedure DeleteArcSPT and all its parts. We have shown that this procedure takes time which is proportional to the number of affected vertices that appear in SPT after deleting an arc. It is obvious that this estimation is optimal.

We are planning to design an associative algorithm for dynamic update of the shortest paths tree after insertion of an arc into the given graph.

References

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

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

[3] Frigioni D., Marchetti-Spaccamela A., Nanni U. Semi–dynamic algorithms for maintaining single source shortest paths trees // Algorithmica. – Berlin: Springer–Verlag, 1998. – Vol. 25, N 3. – P. 250–274.

[4] Frigioni D., Marchetti-Spaccamela A., and Nanni U. Fully dynamic algorithms for maintaining shortest paths trees // J. of Algorithms, Academic Press. – 2000. – Vol. 34, N 2. – P. 351–381.

[5] Frigioni D., Marchetti-Spaccamela A., and Nanni U. Fully dynamic shortest paths in digraphs with arbitrary arc weights // J. of Algorithms, Elsevier Science. – 2003. – Vol. 49, N 1. – P. 86–113.

[6] Nepomniaschaya A. S. Associative version of the Ramalingam decremental algorithm for dynamic update of the single-sink shortest paths subgraph // Proc. of the 10-th Intern. Conf. on Parallel Computing Technologies, PaCT-2009, Novosibirsk, Russia. – Lect. Notes Comput. Sci. – Berlin: Springer-Verlag, 2009. – Vol. 5698. – P. 257–268.

[7] Nepomniaschaya A. S. Basic associative parallel algorithms for vertical processing systems // Bull. of the NCC. – Ser.: Comput. Sci. – 2009. – IIS Special Iss. 29. – P. 63–77.

[8] Nepomniaschaya A. S. Parallel implementation of the Ramalingam incremental algorithm for dynamic update of the shortest–paths subgraph // Bull. of the NCC. – Ser.: Comput. Sci. – 2010. – Iss. 30. – P. 53–69.

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

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

[11] Ramalingam G. Bounded incremental computation // Lect. Notes Comput. Sci. – Berlin: Springer–Verlag. – 1996. – Vol. 1089.

[12] Ramalingam G. and Reps T. An incremental algorithm for a generalization of the shortest paths problem // J. of Algorithms. – Academic Press, 1996. – Vol. 21. – P. 267–305.