Bull. Nov. Comp. Center, Comp. Science, 24 (2006), 89–103
© 2006 NCC Publisher

Associative parallel algorithm for dynamic reconstruction of the shortest paths tree after insertion of a vertex*

A. Sh. Nepomniaschaya, T. V. Snytnikova

1. Introduction

Finding shortest paths in a weighted graph is a fundamental and well studied problem in computer science. Such a problem arises in practice in different application settings.

In this paper we study the single source shortest paths version of the problem in a directed graph with non-negative edge weights. The best known static solution for this problem on directed graphs with n vertices and m edges is the $O(m + n \log n)$ implementation of Dijkstra's algorithm [4] that uses Fibonacci heaps [7].

The dynamic version of this problem consists of maintaining the shortest paths information while the graph changes without recomputing everything from scratch after every update on the graph. In this framework the most general types of update operations for the single source shortest paths problem include insertions and deletions of edges, update operations on the weight of edges, insertions or deletions of isolated vertices [9]. The typical operations for the all-pairs shortest paths problem include update operations on weights, finding the shortest distance and finding the shortest path between two vertices, if any. When arbitrary sequences of the above operations are allowed, we refer to the *fully dynamic* problem. If we consider only insertions (deletions) of edges, we refer to the *incremental* (*decremental*) problem.

In the case of positive edge weights several solutions have been proposed for dynamic maintaining the shortest paths. Ausiello et.al. [1] propose an efficient solution for the all-pairs incremental problem assuming that edge weights are restricted in the range of integers $[1..C]$. Chaudhuri and Zaroliagis [3] devise efficient solutions for the all-pairs shortest paths problem for bounded treewidth graphs when the weight of edges changes. Klein et.al. [12] propose a fully dynamic solution to maintain all-pairs shortest paths for planar graphs with unrestricted edge weights. Franciosa et.al. [6] devise fast algorithms that maintain a single source shortest paths tree (*sp-tree*) of a

*Partially supported by the Russian Foundation for Basic Research under Grant 03-01-00399 general directed graph with integer edge weights in the range of integers $[1..C]$ during a sequence of edge deletions or a sequence of edge insertions.

However, on general graphs with arbitrary edge weights, neither a fully dynamic solution nor a decremental solution for the single source shortest paths problem is known in the standard models (the worst case and the amortized analysis) that is assymptotically better than recomputing a new solution from scratch.

In the case of arbitrary real edge weights, Ramalingam and Reps [18, 19] 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. In [18, 19] the authors assume that the graph has no negative-length cycles before and after input update. Note that they do not deal with zero-length cycles. Frigioni et.al. [10] 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. The decremental solution works only for planar graphs, while the incremental solution works for any graph and its complexity depends on the existence of a k-bounded accounting function for the graph. Frigioni et.al. [9] propose fully dynamic algorithms for updating the distances and an sp-tree 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. For general graphs with n vertices and m edges the algorithms require $O(\sqrt{m}\log n)$ worst case time per output update. Frigioni et al. [11] 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. They also propose an algorithm for handling edge insertions and weight decreases that explicitly deals with negative-length cycles. The cost of the update operations is evaluated as a function of the structural property of the graph and of the number of output updates. Note that algorithms from [6, 9–11, 18, 19] use the dynamic version of Dijkstra's algorithm. In [13], Narváez et.al. study a group of algorithms for dynamic maintaining an sp-tree after performing the update operations on the edge weights. The authors propose two incremental methods to transform the well-known static algorithms of Dijkstra, Bellman-Ford, and D'Esopo-Pape into new dynamic algorithms.

Frigioni and Italiano [8] consider graphs whose vertices may be switched either on or off. The authors study the problem of dynamically maintaining a planar graph under an intermixed sequence of edge and vertex updates. They show how to efficiently maintain connectivity on planar graphs under an intermixed sequence of switch-on, switch-off, insert, delete, and query operations in polylogarithmic time for updates and queries. Their algorithms use separator properties of planar graphs.

In this paper, we propose a simple associative parallel algorithm for dynamic maintenance of distances and the shortest paths from a source vertex when a new vertex is inserted in the underlying graph. Let G be a directed graph with non-negative edge weights and T be an sp-tree. Let a new vertex be added to G. We want to compute a new sp-tree for the altered graph by performing changes in the given sp-tree. 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 and simple single-bit processing elements (PEs). Such an architecture is best suited to solving the graph problems.

Our algorithm is also based on the dynamic version of Dijkstra's algorithm. Differently from the above-mentioned papers, it uses simple and natural data structures and it is implemented in parallel. The algorithm is represented on the STAR-machine as a procedure InsertV whose correctness is proved. We obtain that this procedure takes $O(hk)$ time, where h is the number of bits for coding the infinity and k is the number of vertices whose tree paths change after inserting a new vertex along with its incident edges into the current graph. Following [5], it is assumed that each elementary operation of the STAR-machine (its microstep) takes one unit of time.

2. Model of associative parallel machine

Here, we propose a short description of the model. It is defined as an abstract STAR-machine of the SIMD type with a vertical data processing [14]. It consists of the following components:

– a sequential control unit (CU), where programs and scalar constants are stored;

– an associative processing unit consisting of p single-bit PEs;

– a matrix memory for the associative processing unit.

The CU passes an instruction to all PEs in one unit of time. All active PEs execute it in parallel while inactive PEs do not perform it. Activation of a PE depends on the data.

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

An associative processing unit is represented as h vertical registers each consisting of p bits. Vertical registers can be regarded as a one-column array. The bit columns of the tabular data are stored in the registers which perform the necessary Boolean operations.

Its run is described by means of the language STAR [14] being an extension of Pascal. Let us briefly consider the STAR constructions needed for the paper. To simulate data processing in the matrix memory, we use data types **word, slice**, and **table**. Constants for the types **slice** and **word** are represented as a sequence of symbols of the set $\{0, 1\}$ enclosed within single quotation marks. The types **slice** and **word** are used for the bit column access and the bit row access, respectively, and the type **table** is used for defining the tabular data. Assume that any variable of the type **slice** consists of p components which belong to $\{0, 1\}$. For simplicity let us call *slice* any variable of the type **slice**.

Now we present some elementary operations and predicates for slices.

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

SET(Y) sets all 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 (or the uppermost) $'1'$ of Y, $i \geq 0$;

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

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

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

$w(i)$ returns the i-th component (bit) of w;

TRIM(i, j, w) returns the substring $w(i)w(i+1)...w(j)$, where $1 \leq i < j \leq\mid w \mid$;

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

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

Note that all operations for the type **slice** are also performed for the type **word**.

Remark 1. Note that the STAR statements are defined in the same manner as for Pascal. We will use them later for presentation of our procedures.

Following [5], we assume that each elementary operation of the STAR-machine takes one unit of time. Therefore we will measure *time complexity* of an algorithm by counting all elementary operations performed in the worst case.

Now we consider a group of basic procedures to be used later. Implementation of these procedures on the STAR-machine has been given in [15, 16]. They use the given slice X to select by $'1'$ *positions* of rows being used in the corresponding procedure.

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

The procedure MIN(T, X, Z) defines positions of those rows of the given matrix T, where minimum elements are located. It returns the slice Z, where $Z(i) =' 1'$ if and only if ROW(i, T) is the minimum element of the matrix T and $X(i) =' 1'$. To return the minimum element of T, we define the position of the first $'1'$ in Z and then select the corresponding row of T.

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

The procedure ADDC(T, X, v, F) adds the binary word v to those rows of the matrix T which are selected by $'1'$ in X, and writes down the result into the corresponding rows of the matrix F. The rows of F, which are selected by $'0'$ in X, will consist of zeros.

The procedure ADDV(T, R, X, F) writes the result of adding the rows of matrices T and R selected by ones in the slice X into the corresponding rows of the matrix F. Note that this procedure is based on the associative algorithm from [5].

The procedure TMERGE(T, X, F) writes into the matrix F those rows of the given matrix T which are selected by $'1'$ in X. The rows of the matrix F, which are selected by $'0'$ in X, are not changed.

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

The procedure TCOPY2(F, j, h, T) writes the given matrix F, consisting of h columns, into the result matrix T beginning with its $(1 + (j - 1)h)$-th column, where $j \geq 1$.

In [15, 16], we have shown that basic procedures take $O(k)$ time each, where k is the number of bit columns in the corresponding matrix.

3. Preliminaries

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

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

Note that the weights are non-negative integers represented as binary strings.

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

A *tree of the shortest paths* T with the root vertex 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 is the shortest path from s to v in G.

Following [6], let us call an $sp - tree$ the single source shortest paths tree.

In [17], we propose an associative parallel algorithm for finding the matrix of the shortest distances D along with the sp-tree T.

For any arc $v_i \to v_j$, let v_i be the *parent* of v_j and v_j be the *son* of v_i.

For the sake of convenience, we will use an $n \times n$ matrix of descendants T whose every i-th column saves by $'1'$ the sons of vertex v_i.

Remark 2. The sp-tree is obtained as follows. Knowing the matrix of descendants T, we easily build a matrix T_1 whose every i-th row saves by $'1'$ *positions* of vertices belonging to the tree path from s to v_i. Then we transpose the matrix T_1 and obtain the corresponding sp-tree.

Let us agree to denote the shortest distance from s to a vertex v_i as $dist(s, v_i)$.

We will also use the auxiliary procedure WTRANS(w, h, n, R) [17] having the input parameters: the given binary string w and the integers h and n, where h is the number of bits for coding infinity and n is the number of graph vertices. It returns the matrix R in whose each i-th row there is the string $w_i =$TRIM$(1 + (i-1)h, ih, w)$.

4. Reconstruction of the sp-tree and the matrix of distances

In this section, we provide an associative parallel algorithm that reconstructs the matrix of descendants T and recomputes the matrix of the shortest distances D for the vertices that belong to a subtree rooted in a vertex from a set L.

Our algorithm performs the following steps.

Step 1. In the set L, select a vertex, say v_i, having the shortest distance in D. Delete v_i from L.

Step 2. Determine in parallel those vertices v_j for which there is an arc $v_i \to v_j$. Then compute in parallel $dist(s, v_i) + wt(v_i, v_j)$.

Step 3. By means of a slice, say Z, save those vertices v_k whose paths include the vertex v_i and $dist(s, v_k) > dist(s, v_i) + wt(v_i, v_k)$. Add v_k to the set L.

Step 4. For vertices marked with $'1'$ in the slice Z, write new distances in the corresponding rows of the matrix D and write a new parent v_i in the matrix T.

Step 5. If $L \neq \Theta$ [1], go to Step 1.

This algorithm is implemented as a procedure Propagate which uses the following input parameters: the transposed weight matrix for the given graph G; the binary code of infinity inf; the added vertex v.

It returns the matrix of descendants T and the matrix of the shortest distances D that correspond to the sp-tree for the graph G with the added vertex v.

Initially the shortest path from s to v has been written in the v-th row of the matrix D and the parent of v has been written in the v-th row of the matrix T.

Now we propose the following procedure.

```
procedure Propagate(G: table; inf: word; v: integer;
  var T: table; var D: table);
var i,j,h: integer;
  w,w1: word;
  A,L,X,Y,Z: slice;
  D1,R1: table;
 1. Begin SET(Y); CLR(L);
 2.  L(v):='1';
 3.  while SOME(L) do
 4.  begin MIN(D,L,X);
 5.    i:=STEP(X); w:=ROW(i,D);
 6.    L(i):='0';
/* In the slice $L$, we delete the vertex having
   the minimal distance from the root $s$. */
 7.    TCOPY1(G,i,h,R1);
/* The matrix $R1$ saves the weights of arcs incident to vertex $v_i$. */
 8.    MATCH(R1,Y,inf,Z); A:= not Z;
 9.    ADDC(R1,A,w,D1);
10.    SETMIN(D1,D,A,Z);
/* The slice $Z$ saves the vertices whose distances from $s$
   were decreased. */
11.    if SOME(Z) then
12.    begin L:=L or Z;
13.      TMERGE(D1,Z,D);
/* The matrix $D$ saves new distances for vertices
   selected in the slice $Z$. */
14.      CLR(w1); w1(i):='1';
15.      while SOME(Z) do
16.      begin j:=STEP(Z);
17.        ROW(j,T):=w1;
```

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

```
/* For any vertex $v_j$ marked with '1' in the slice $Z$,
   a new parent is written in the $j$-th row of $T$. */
18.       end;
19.     end;
20.   end;
21. End;
```

Theorem 1. *Let an undirected graph G be represented as a transposed weight matrix. Let the distance matrix D and the matrix of descendants T be given for G. Let a vertex v be added to G, the shortest path from s to v be written in the v-th row of D, the v-th column consisting of zeros and the v-th row saving the parent of v be added to the matrix T. Then the procedure Propagate(G,inf,v,T,D) returns the matrices D and T that correspond to the sp-tree for the graph G with the vertex v.*

Proof. (Sketch) We will prove the theorem by induction on the number of arcs which are added to the subtree $S(v)$ of the new sp-tree.

Basis is checked for the case when $S(v)$ consists of a single arc. After performing lines 1–6, the slice L consists of the vertex v, the variable w saves $dist(s,v)$, and v is deleted from L. Then it is included into $S(v)$ because, in view of Remark 4, the value of $dist(s,v)$ cannot be decreased.

After performing lines 7–10, the slice Z saves *positions* of those sons v_k of vertex v for which $dist(s,v_k) > dist(s,v) + wt(v,v_k)$, where $dist(s,v_k)$ is the length of the shortest path in the previous sp-tree. Let us call such a vertex v_k *a selected vertex.*

After performing lines 11–18, the selected vertices are included into the slice L, the smaller new distances $dist(s,v)+wt(v,v_k)$ are written in the corresponding rows of the matrix D instead of the previous values of $dist(s,v_k)$. Besides, the vertex v is marked as a parent of every vertex v_k. Then we run to the end of the statement while SOME(L) do (line 20). Since $L \neq \Theta$, we perform line 4. Here, among the vertices of L, we choose the son v_j for which the new shortest distance from s has the minimum value. This son is deleted from L because the distance from s to v_j cannot be decreased. Actually, assume that during the construction of the new sp-tree, there exists another path fom s to v_j, say γ, that includes the vertex v. Then obviously γ includes no less than two arcs. Since the weights of arcs are non-negative and $wt(v,v_j) \leq wt(v,v_r)$ for all $v_r \neq v_j$, the path γ will have the greater length than the path including the arc $v \rightarrow v_j$. Since the new shortest path to the vertex v_j has been written in the j-th row of the matrix D and its parent has been written in the j-th row of the matrix T, the vertex v_j is included into $S(v)$. Hence, the arc $v \rightarrow v_j$ belongs to $S(v)$.

Step of induction. Let vertices $v = v_1, v_2, \ldots, v_k$ have been already included into $S(v)$. Let v_j be the son of v_r $(2 \leq r \leq k)$. Let, at the current iteration, the path from s to v_j including the vertex v have the minimum value in L. Then by analogy with the basis, after deleting the vertex v_r from the slice L, the new distance from s to v_j is written in the j-th row of the matrix D and its parent v_r is written in the j-th row of the matrix T. Since this distance from s to v_j cannot be decreased, v_j is deleted from the slice L and then it is included into $S(v)$. Therefore, a new arc (v_r, v_j) is added to the subtree $S(v)$. ■

Now we propose an associative parallel algorithm that performs the dynamic update of an sp-tree after insertion of a new vertex v to the graph.

Let InE be a row that saves the weights of arcs entering the vertex v. Let $OutE$ be an $n \times h$ matrix whose every i-th row saves the weight of an arc outgoing from the vertex v.

An associative parallel algorithm for the dynamic update of the sp-tree after insertion of a vertex v to G performs the following steps:

Step 1. Insert the vertex v into the graph G.

Step 2. Determine the parent of v in the current sp-tree and the shortest distance from s to v.

Step 3. Recompute the matrix of the shortest paths D and the matrix of descendants T for the graph G with the added vertex v.

On the STAR-machine this algorithm is implemented as a procedure InsertV.

It uses the following input parameters: integers h and n described above; the vertex v; the code of infinity inf; the row InE; the matrix $OutE$.

It returns the transposed weight matrix for the graph G with the added vertex v and the recomputed matrices D and T that correspond to the extended graph G.

Now we provide the following procedure.

```
procedure InsertV(h,n,v: integer; inf: word; InE: word;
  OutE: table; var G: table; var T: table; var D: table);
var i: integer;
  R1,D1: table;
  u,w: word;
  X,Y: slice;
1. Begin SET(X); CLR(u);
2.  ROW(v,G):=InE;
3.  TCOPY2(OutE,v,h,G);
4.  WTRANS(InE,h,n,R1);
5.  MATCH(R1,X,inf,Y);
6.  X:= not Y;
7.  ADDV(R1,D,X,D1);
8.  MIN(D1,X,Y); i:=FND(Y);
9.  w:=ROW(i,D1);
```

```
10.  ROW(v,D):=w;
11.  u(i):='1'; ROW(v,T):=u;
12.  CLR(Y); COL(v,T):=Y;
13.  Propagate(G,inf,v,T,D);
14. End;
```

Theorem 2. *Let a directed graph G be represented as a transposed weight matrix. Let the distance matrix D and the matrix of descendants T be given for G. Let the code of infinity inf, its length h, and the additional vertex v for G be known. Let the weights of arcs entering v be given as a row InE and the weights of arcs outgoing from v be given as a matrix OutE. Then the procedure InsertV(h,n,v,inf,InE,OutE,G,T,D) returns the altered weight matrix and matrices D and T that correspond to the sp-tree for the graph G with the added vertex v.*

Proof. (Sketch) According to Theorem 1, the procedure Propagate returns the matrices D and T that correspond to the sp-tree for the graph G along with the added vertex v. Therefore it is necessary to check that, after performing lines 1–12, we obtain, in particular, all input parameters for the procedure Propagate.

One can immediately verify that, after performing lines 1–3, we obtain the transposed weight matrix for the graph G with the added vertex v. By means of the technique from [17], after performing lines 4–6, we determine *positions* of arcs entering v. Then, after fulfilling lines 7–8, we first determine the length of different paths from s to the new vertex v. Knowing these paths, we select a vertex v_i such that the arc $v_i \to v$ belongs to the tree path from s to v and this path has the minimum weight.

After performing lines 9–11, this shortest distance is written in the v-th row of the matrix D. Then in the v-th row of the matrix T, the vertex v_i is marked as the parent of v. Finally, after performing line 12, we set zeros in the v-th column of the matrix T.

Now all conditions of Theorem 1 are satisfied. Therefore we can apply the procedure Propagate. ∎

Let us evaluate time complexity of the procedure InsertV. By analogy with [17], we obtain that this procedure takes $O(hk)$ time, where h is the number of bits required for coding the maximum weight of the shortest paths from the source vertex s and k is the number of vertices whose tree paths change. Note that the static algorithm for finding the sp-tree from [17] requires $O(hn)$ time, where n is the number of graph vertices.

Remark 3. Let a new arc, say $v_i \to v_j$, be inserted in the graph. Then we update the sp-tree as follows. We first determine the weight of the *new* tree path from s to v_j that includes v_i. If it is no less than $dist(s, v_j)$

written in the j-th row of the matrix D, then matrices T and D do not change. Otherwise, we write the new distance in the j-th row of the matrix D. Moreover, in the j-th row of the matrix T, we mark the vertex v_i as the $father$ of the vertex v_j. After that, by means of the procedure Propagate, we recompute tree paths and distances for the vertices from the subtree rooted at the vertex v_j.

5. Experiments

In this section, we provide an example of implementing the procedure InsertV on the STAR-machine. To this end, we will use the experimental system VisualStar [2] which allows one to edit, compile, and implement procedures written in the language STAR. It was realized by means of Borland Delphi 4.0.

The original graph G and its sp-tree are given in Figure 1.

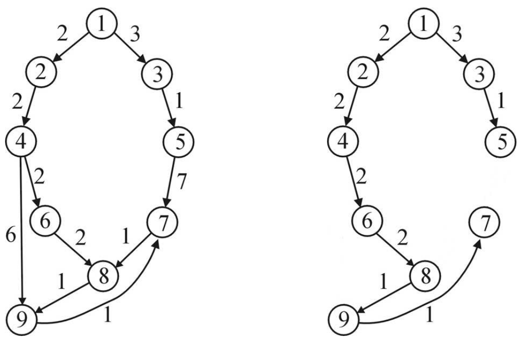

Figure 1. Graph G and its sp-tree

Let the matrix of the shortest distances D and the matrix of descendants T that correspond to the sp-tree from Figure 1 be given for the graph G. Let vertex 10 along with the arcs $2 \to 10$, $5 \to 10$, $10 \to 6$, and $10 \to 7$ be added to G. Let $wt(2, 10) = wt(5, 10) = wt(10, 6) = 1$ and $wt(10, 7) = 6$ as shown in Figure 2.

We have to reconstruct the matrices D and T in such a way that they will correspond to the sp-tree of the altered graph G after adding vertex 10.

Note that in any iteration of performing the procedure InsertV, we indicate both the current distance from s to every vertex v of G and the current father of any v.

Let us agree to mark in bold the vertices from the set L whose distances from s decrease at the current iteration. Let $P(v)$ denote a father of the vertex v.

After performing the 6-th iteration, we obtain the matrix of distances D and the matrix of descendants T for the altered graph G after adding vertex

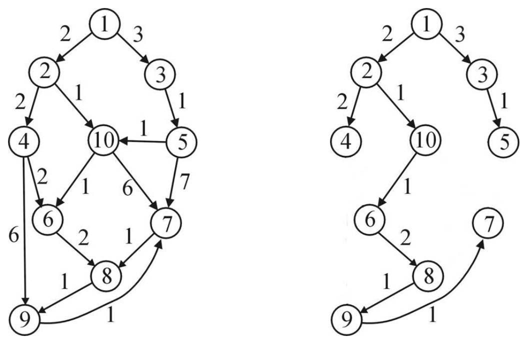

Figure 2. The altered graph G and its sp-tree

	Matrix T									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0
10	0	1	0	0	0	0	0	0	0	0

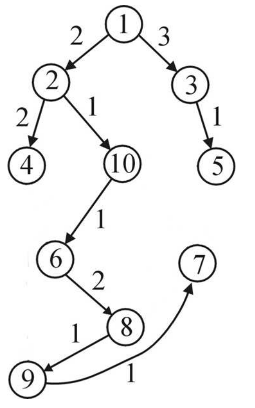

Figure 3. The sp-tree for G with the added vertex 10

10. The obtained matrix of descendants from Table 1 corresponds to the tree of the shortest paths given in Figure 3.

6. Conclusions

We proposed a new result concerning the dynamic maintenance of the shortest paths tree after insertion of a new vertex in a directed graph with non-negative arc weights. The algorithm is implemented as the procedure InsertV on the STAR-machine that simulates the run of associative parallel processors with vertical processing. We apply some constructions from [18] being designed in implementing Dijkstra's algorithm on the STAR-machine to obtain the shortest paths tree along with the matrix of the shortest distances. We proved correctness of the procedure InsertV and evaluate its time complexity. We have obtained that it takes $O(hk)$ time, where h is the number of bits required for coding the maximum weight of the shortest

Table 1. Implementation of InsertV

	Iteration 1			Iteration 2			Iteration 3		
	D	L	P(v)	D	L	P(v)	D	L	P(v)
1	0	0	0	0	0	0	0	0	0
2	2	0	1	2	0	1	2	0	1
3	3	0	1	3	0	1	3	0	1
4	4	0	2	4	0	2	4	0	2
5	4	0	3	4	0	3	4	0	3
6	6	0	4	**4**	**1**	**10**	4	0	10
7	10	0	9	**9**	**1**	**10**	9	1	10
8	8	0	6	8	0	6	**6**	**1**	**6**
9	9	0	8	9	0	8	9	0	8
10	**3**	**1**	**2**	3	0	2	3	0	2

	Iteration 4			Iteration 5			Iteration 6		
	D	L	P(v)	D	L	P(v)	D	L	P(v)
1	0	0	0	0	0	0	0	0	0
2	2	0	1	2	0	1	2	0	1
3	3	0	1	3	0	1	3	0	1
4	4	0	2	4	0	2	4	0	2
5	4	0	3	4	0	3	4	0	3
6	4	0	10	4	0	10	4	0	10
7	9	1	10	**8**	**1**	**9**	8	0	9
8	6	0	6	6	0	6	6	0	6
9	**7**	**1**	**8**	7	0	8	7	0	8
10	3	0	2	3	0	2	3	0	2

paths from s and k is the number of vertices whose tree paths change.

We are planning to design an associative parallel algorithm for dynamic reconstruction of the shortest paths tree after deleting a vertex along with its incident edges from a digraph.

References

[1] Ausiello G., Italiano G.F., Marchetti-Spaccamela A., Nanni U. Incremental algorithms for minimal length paths // J. of Algorithms. — 1991. — Vol. 12, N 4. — P. 615–638.

[2] Borets T.V. A programming system VisualStar // Proc. of the Conference of young scientists / Ed.: G.A. Michailov. — Novosibirsk, 2004. — P. 20–26.

[3] Chaudhuri S., Zaroliagis C.D. Shortest path queries in digraphs of small treewidth // Lect. Notes Comput. Sci. — 1995. — Vol. 944. — P. 244–255.

[4] Dijkstra E. W. A Note on Two Problems in Connection with Graphs // Numerische Mathematik. — 1959. — Vol. 1. — P. 269–271.

[5] Foster C.C. Content Addressable Parallel Processors. — NY: Van Nostrand Reinhold Company, 1976.

[6] Franciosa P.G., Frigioni D., Giaccio R. Semi-dynamic shortest paths and breadth-first searsch in digraphs // Lect. Notes Comput. Sci. — 1997. — Vol. 1200. — P. 33–46.

[7] Fredman M.L., Tarjan R.E. Fibonacci heaps and their use in improved network optimization algorithms // J. of the ACM. — 1987. — Vol. 34. — P. 596–615.

[8] Frigioni D., Italiano G.F. Dynamically switching vertices in planar graphs // Algorithmica. — Berlin: Springer–Verlag, 2000. — Vol. 28, N 1. — P. 76–103.

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

[10] 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.

[11] Frigioni D., Marchetti-Spaccamela A., 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.

[12] Klein P.N., Rao S., Rauch M., Subramanian S. Faster shortest path algorithms for planar graphs // Proc. ACM Symp. on Theory of Computing, Montreal, Quebec, Canada, May 23–25, 1994. — P. 27–377.

[13] Narváez P., Siu K.-Y., Tzeng H.-Y. New dynamic algorithms for shortest paths tree computation // IEEE/ACM Trans. Networking. — 2000. — Vol. 8, N 6. — P. 734–746.

[14] Nepomniaschaya A. S. Language STAR for associative and parallel computation with vertical data processing // Proc. of the Internat. Conf. “Parallel Computing Technologies”. — Singapure: World Scientific, 1991. — P. 258–265.

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

[16] 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.

[17] Nepomniaschaya A. S. Concurrent selection of the shortest paths and distances in directed graphs using vertical processing systems // Bull. Novosibirsk Comp. Center. Ser. Computer Science. — Novosibirsk, 2003. — Iss. 19. — P. 61–72.

[18] Ramalingam G. Bounded incremental computation // Lect. Notes Comput. Sci. — 1996. — Vol. 1089.

[19] 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.