*Bull. Nov. Comp. Center*, Comp. Science, 40 (2016), 75–86


# Associative version of the Ramalingam incremental algorithm for the dynamic all-pairs shortest-path problem

A. S. Nepomniaschaya

**Abstract.** The paper proposes an associative version of the Ramalingam algorithm for the dynamic update of the all-pairs shortest paths of a directed weighted graph after inserting an edge. To this end, a model of associative (content addressable) parallel systems with vertical processing (the STAR–machine) is used. The associative version of the Ramalingam incremental algorithm is given as procedure InsertEdge. We present an efficient implementation of this procedure on the STAR–machine, prove its correctness and evaluate the time complexity.

**Keywords:** directed weighted graph, incremental algorithm, associative parallel processor, access data by contents, the time complexity.

## 1. Introduction

Associative (content addressable) parallel processors of the SIMD type with bit–serial (vertical) processing and simple processing elements (PEs) perform the massively parallel search by contents and use 2D tables as the basic data structure. In particular, such an architecture is best suited for natural and efficient implementation of graph algorithms. In [6], we propose an abstract model of the SIMD type (the STAR–machine) that simulates the run of such systems at the micro level. Associative parallel algorithms are represented as corresponding procedures for the STAR–machine. In [7], we present basic associative parallel algorithms that are used to design different associative algorithms for different applications. Of special interest is implementation of dynamic graph algorithms on the STAR–machine. These algorithms update the solution of a problem after dynamic changes faster than the fastest static algorithm that computes the entire graph from scratch. Let us enumerate a group of dynamic graph algorithms implemented on the STAR–machine. In [8], we propose two associative parallel algorithms for the dynamic edge update of a minimum spanning tree (MST) of an undirected graph. In [9–10], we propose associative parallel algorithms for the dynamic reconstruction of an MST after deleting and after inserting a new vertex along with its incident edges. In [11], we present associative versions of the Italiano algorithms for the dynamic update of the transitive closure of a directed graph after inserting and after deleting an edge. In [12–13], we propose associative versions of the Ramalingam algorithms for

updating the shortest paths subgraph with a sink after inserting and after deleting an edge.

In this paper, we study the dynamic update of the all-pairs shortest paths (APSP) by means of the STAR–machine. The most general types of update operations for the APSP problem include insertions and deletions of edges, update operations on edge weights, finding the shortest distance and finding the shortest path between two vertices, if any. An algorithm is called *fully dynamic* if the update operations include both insertions and deletions of edges. A partially dynamic algorithm is called *incremental* if it supports only insertions and *decremental* if only deletions are supported.

In the case of positive edge weights, several solutions have been proposed for the dynamic maintenance of the all–pairs shortest paths. Ausiello et al. [1] propose an efficient solution for the incremental APSP problem assuming that edge weights are restricted in the range of integers $[1..C]$. Chaudhuri and Zaroliagis [2] devise efficient solutions for the APSP problem for bounded treewidth graphs when the weight of edges changes. King [4] proposes fully dynamic algorithms for updating the all–pairs shortest paths in directed graphs with positive integer weights less than $C$. Ramalingam [15] proposes fully dynamic algorithms for updating the all-pairs shortest paths in directed graphs with *positive* edge weights. Demetrescu and Italiano [3] devise fully dynamic algorithms for the dynamic maintenance of the all–pairs shortest paths in directed graphs with non–negative real edge weights. Observe that algorithms of Ramalingam [15] are described by means of the output bounded model in which the running time of an algorithm is analyzed in terms of the output change rather than the input size.

In [14], we have constructed an associative version of the Ramalingam decremental algorithm for the dynamic update of the all–pairs shortest paths in a directed graph. This algorithm has been built as a modification of his previous algorithm for updating the shortest paths subgraph with a sink after deleting an edge from the graph. The associative version is given as a group of algorithms that provide an efficient *parallel* execution of different parts of the Ramalingam decremental algorithm on the STAR–machine. In [14], we have also presented the main advantages of the associative version of the Ramalingam decremental algorithm for updating the all–pairs shortest paths.

Here, we construct an associative version of the Ramalingam incremental algorithm for the dynamic update of the all–pairs shortest paths in a directed graph. This algorithm is built as a modification of his previous algorithm for updating the shortest paths subgraph with a sink after inserting an edge. The associative version is given as procedure InsertEdge whose correctness is proved and the time complexity is avaluated.

## 2. Basic definitions

In this section, we first present preliminaries. Then we recall some operations of the STAR–machine that will be used in the paper.

Let $G = (V, E)$ be a *directed weighted graph* with the set of vertices $V = \{1, 2, \ldots, n\}$, the set of directed edges (arcs) $E$ and the function $wt$ that assigns a weight to every edge. We will consider graphs with a distinguished vertex $z$ called *sink*.

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 $E$.

An arc $e$ directed from $i$ to $j$ is denoted by $e = (i, j)$, where the vertex $i$ is the *tail* of $e$ and the vertex $j$ is its *head*. We assume that all arcs have a positive weight and $wt(u, v) = \infty$ if $(u, v) \notin E$.

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

A *path* from $u$ to $z$ in $G$ is a finite sequence of vertices $u = v_1, v_2, \ldots, v_k = z$, where $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, k-1$ and $k > 1$. The *shortest path* from $u$ to $z$ is the path of the minimal sum of weights of its arcs. Let $dist(u, z)$ denote the length of the shortest path from $u$ to $z$.

By analogy with Ramalingam, we introduce the following notations.

Let $Succ(u) = \{x \,/\, u \to x \in E\}$ and $Pred(u) = \{y \,/\, y \to u \in E\}$.

Let an arc $(i, j)$ be inserted into $G$. A vertex $u$ is called *affected* in $G$ if the length of the shortest path from $u$ to $z$ changes after inserting the edge $(i, j)$. Let *AffectedVert* be a set of all vertices such that the length of their shortest path to $z$ changes after inserting $(i, j)$.

A predicate $SP(a, b, c)$ is defined as follows:

$SP(a, b, c) \equiv (dist(a, c) = wt(a, b) + dist(b, c)) \wedge (dist(a, c) \neq \infty.)$

This predicate verifies whether the arc $(a, b)$ belongs to the shortest path from the vertex $a$ to the selected sink $c$.

Now we consider some operations of our model. Its run is described by means of the language STAR which is an extension of Pascal. To simulate the data processing in the matrix memory, the language STAR uses the data types **word, slice**, and **table**. The types **slice** and **word** are used for the bit column access and bit row access, respectively, and the type **table** is used for defining the tabular data.

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

The bitwise Boolean operations are introduced in the usual way: $X\ and\ Y$, $X\ or\ Y$, $not\ Y$, $X\ xor\ Y$. The predicate SOME($Y$) results in **true** if there is at least a single bit $'1'$ in the slice $Y$.

Let $v, w$ be variables of the type **word** and $i, j$ be variables of the type **integer**. We employ the following operations:

TRIM($i, j, w$) cuts the substring of the string $w$ from the $i$-th through the $j$-th bits, where $1 \leq i < j \leq \mid w \mid$.

REP($i, j, v, w$) replaces the substring $w(i)w(i+1)\ldots w(j)$ of the string $w$ with the string $v$, where $\mid v \mid = j - i + 1$ and $1 \leq i < j < \mid w \mid$.

ADD($v, w$) performs the addition of binary strings $v$ and $w$ that have the same length.

## 3. The Ramalingam incremental algorithm for updating the all-pairs shortest paths

Let $G$ be a directed graph and $z$ be its sink. Let an arc $(i, j)$ with $wt(i, j) = c$ be inserted into $G$. The Ramalingam algorithm for the dynamic update of the all-paths shortest paths after inserting an arc into $G$ runs as follows. At first, it computes a set $AffectedSinks$. Then the *simplified form* of the Ramalingam incremental algorithm for the dynamic update of the shortest paths subgraph is applied to every vertex from the set *AffectedSinks*.

We first explain the simplified form of the Ramalingam incremental algorithm for updating the shortest paths subgraph. It uses a set *WorkSet* to save the updated edges, a set *AffectedVert*, and a set *VisitedVert*. The set *VisitedVert* is used to avoid visiting any vertex more than once. The simplified form of the Ramalingam algorithm runs as follows.

Initially, $AffectedVert = \emptyset$, $WorkSet = \{i, j\}$, and $VisitedVert = \{i\}$.

While $WorkSet \neq \emptyset$, select and remove an edge $(x, u)$ from *WorkSet*. If $wt(x, u) + dist(u, z) < dist(x, z)$, then insert $x$ into the set *AffectedVert* and the new shortest path from $x$ to $z$ is defined as $wt(x, u) + dist(u, z)$.

Then every vertex $y \in Pred(x)$ is updated as follows. One first checks whether the predicate $SP(y, x, z)$ is true and $y \notin VisitedVert$. If it is true, then the arc $(y, x)$ is inserted into *WorkSet* and the vertex $y$ is inserted into the set *VisitedVert*.

After updating all edges from the set *WorkSet*, the set $AffectedVert$ will consist of all affected vertices obtained after inserting the arc $(i, j)$ into $G$. Moreover, one computes the new distance from every affected vertex to the sink $z$.

In [15], the simplified form of the Ramalingam incremental algorithm is given as the function InsertUpdate.

Now, we consider the Ramalingam incremental algorithm for updating the all-paths shortest paths. It runs as follows. At first, the arc $(i, j)$ is inserted into $G$. Then the function InsertUpdate is applied, where the sink

$z = j$ to obtain the set AffectedSinks. After that the function InsertUpdate is applied to every vertex from the set AffectedSinks. In [15], this algorithm is given as procedure InsertEdge.

## 4. Associative version of the Ramalingam incremental algorithm for updating the all–pairs shortest paths

To design an associative version of the Ramalingam incremental algorithm, we employ the following data structure:

– an $n \times n$ adjacency matrix $Adj$, whose every $i$-th column saves with bits $'1'$ the heads of arcs outgoing from the vertex $i$;

– an $n \times hn$ matrix $Weight$ that 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 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 $n \times hn$ matrix $Dist$ that consists of $n$ fields having $h$ bits each. The distance from the vertex $i$ to the vertex $j$ is written in the $j$-th row of the $i$-th field;

– an $n \times hn$ matrix $Dist1$ that consists of $n$ fields having $h$ bits each. The distance from the vertex $i$ to the vertex $j$ is written in the $i$-th row of the $j$-th field;

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

We notice 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 $Adj$ saves with bits $'1'$ the tails of arcs entering the vertex $j$.

In [14], we propose the associative algorithm $A2$ for *parallel* execution of a group of predicates $SP(x, u, z)$ for all *tails* of arcs entering the vertex $u$. This algorithm uses the matrices *Cost, Adj, Dist*, and $Dist1$. It returns a slice that saves the tails of arcs $(x, u)$ for which the corresponding predicates are true.

We first propose the associative version of the *simplified form* of the Ramalingam incremental algorithm for the dynamic update of the shortest paths subgraph. It uses the slices $AffectedV$ and $VisitedV$ and the set $WS$ that is given as a matrix consisting of two columns along with a slice, say $X$, to save the position of the last non-empty row.

The associative version of the simplified form of the Ramalingam incremental algorithm for updating the shortest paths subgraph with a sink $s$ runs as follows.

Step 1. Initially, write zeros into the slices $AffectedV$, $VisitedV$, and $X$.

Step 2. Write the arc $(i, j)$ into the first row of the matrix $WS$. Then perform the operations $VisitedV(i) :=' 1'$ and $X(1) :=' 1'$.

Step 3. While $X \neq \emptyset$ perform the following actions:

(3.1) Select the position (say, $k$) of the uppermost bit $'1'$ in the slice $X$. Let the arc $(u, p)$ be written in the $k$-th row of the matrix $WS$.

(3.2) In the matrix $Weight$, select the weight of the arc $(u, p)$. Denote by $w1$ the value $wt(u, p)$.

(3.3) In the matrix $Dist$, select the distance from the vertex $p$ to the sink $s$. Denote by $w2$ the value $dist(p, s)$.

(3.4) Compute the sum of the values $w1$ and $w2$. Let $w3$ save this sum.

(3.5) In the matrix $Dist$, select the distance from the vertex $u$ to the sink $s$. Denote by $w$ the value $dist(u, s)$.

(3.6) If $w3 \geq w$, then go to point 3.1 to update the current row of the matrix $WS$. Otherwise, include the vertex $u$ into the slice $AffectedV$.

(3.7) In the $s$-th row of the matrix $Dist$, replace the value $dist(u, s)$ with the smaller value $w3$. In the $u$-th row of the matrix $Dist1$, replace the value $dist(u, s)$ with the value $w3$.

(3.8) By means of the procedure ConputePred2, *simultaneously* define all tails of arcs entering the vertex $u$ for which the predicates $SP(y, u, s)$ are true. Let the procedure ConputePred2 return a slice, say $Z$.

(3.9) By means of a slice, say $Z1$, save those vertices from the slice $Z$ that do not belong to the set $VisitedV$. Include these vertices into the set $VisitedV$.

(3.10) Include into the matrix $WS$ every arc entering the vertex $u$ whose tail belongs to the slice $Z1$.

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

The associative version of the Ramalingam incremental algorithm for the dynamic update of the all–pairs shortest paths runs as follows. At first, the arc $(i, j)$ having $wt(i, j) = v0$ is included into the matrix $Weight$. Then by means of the procedure InsertUpdate, one defines the set of all affected sink vertices for the case when $s = j$. After that for every sink vertex, the procedure InsertUpdate is performed.

## 5. Implementation on the STAR–machine of the Ramalingam incremental algorithm for updating the all–pairs shortest paths

In this section, we first present the procedure InsertUpdate and prove its correctness. Then we consider the procedure InsertEdge.

The procedure InsertUpdate uses the arc $(i, j)$ and the matrices $Adj$, $Weight$, $Cost$, $Dist$ and $Dist1$. It returns the slice $AffectedV$ and the updated matrices $Dist$ and $Dist1$.

Remark 1. Let us agree that the record `w: word(Trim)` means that the string $w$ saves the result of the operation Trim and it consists of $h$ bits.

```
procedure InsertUpdate(i,j,h,n,s: integer; Weight,Cost: table;
  Adj: table; var Dist,Dist1: table;
  var AffectedV: slice(Adj));
var WS: array[1..2,1..n] of integer;
  X,Y,Z,VisitedV: slice(Adj);
  k,k1,l,l1,l2,p,u: integer;
  v: word(Weight); w,w1,w2,w3: word(Trim);
1. Begin CLR(AffectedV); CLR(VisitedV); CLR(X);
2.  X(1):='1'; WS[1,1]:=i; WS[1,2]:=j;
/* The arc (i,j) is included into the first row of WS. */
3.  VisitedV(i):='1';
/* The vertex i is included into the slice VisitedV. */
4.  while SOME(X) do
5.    begin k:=STEP(X); u:=WS[k,1]; p:=WS[k,2];
/* The uppermost arc (i,j) is deleted from WS. */
6.      k1:=FND(not X);
7.      v:=ROW(p,Weight);
/* The word v saves the p-th row of the matrix Weight. */
8.      l1:=1+(u-1)h; l2:=uh; w1:=TRIM(l1,l2,v);
/* The word w1 saves wt(u,p). */
9.      v:=ROW(s,Dist);
/* The word v saves the s-th row of the matrix Dist. */
10.     l1:=1+(p-1)h; l2:=ph;
11.     w2:=TRIM(l1,l2,v);
/* The word w2 saves dist(p,s). */
12.     w3:=ADD(w1,w2);
/* The word w3 saves wt(u,p)+dist(p,s). */
13.     l1:=1+(u-1)h; l2:=uh;
14.     w:=TRIM(l1,l2,v);
/* The word w saves dist(u,s). */
15.     if w3<w then
16.       begin AffectedV(u):='1';
/* The vertex u is included into the slice AffectedV. */
17.         REP(l1,l2,w3,v);
/* In the s-th row of the matrix Dist, the value dist(u,s)
  is replaced with the smaller value w3. */
18.         v:=ROW(u,Dist1);
19.         l1:=1+(s-1)h; l2:=sh;
20.         REP(l1,l2,w3,v);
/* In the u-th row of the matrix Dist1, the value dist(u,s)
  is replaced with the value w3. */
21.         ComputePred2(h,u,s,Adj,Cost,Dist,Dist1,Z);
/* The slice Z saves the tails of arcs entering the vertex u
```

```
   for which the predicates SP(q,u,s) are true. */
22.          Y:=Z and (not VisitedV);
23.          VisitedV:=VisitedV or Y;
24.          while SOME(Y) do
25.            begin l:=STEP(Y);
26.              WS[k1,1]:=l; WS[k1,2]:=u;
27.              X(k1):='1'; k1:=k1+1;
28.            end;
29.        end;
30.    end;
31. End;
```

Remark 2. In this procedure, we use two counters for the slice $X$. At the current iteration, the counter $k$ saves the position of the uppermost arc in the matrix $WS$. This position is defined by means of the operation STEP($X$). By means of the counter $k1$, we define the position of the row in the matrix $WS$, where the new arc can be written. This position is defined by means of the operation FND($not\, X$).

Claim 1. *Let an arc (i,j) be inserted into the directed weighted graph G having n vertices and the sink s. Let the matrices Weight, Cost, Dist, Dist1, and Adj be given. Let h be a parameter that is used in the matrices Weight, Cost, Dist, and Dist1. Then the procedure InsertUpdate returns a slice* AffectedV *that saves the vertices from which the shortest paths to the sink should be recomputed.*

**Proof.** (Sketch.) We prove this by induction in terms of the number of updated arcs $l$ in the matrix $WS$.

Basis is checked for $l = 1$, that is, the edge $(i, j)$ is updated in $WS$. After performing lines 1–3, the arc $(i, j)$ is written in the first row of $WS$, the vertex $i$ is included into the slice $VisitedV$, and only the first bit in the slice $X$ is equal to one. After performing lines 5–6, $k = 1$, $u = i$, $p = j$, $X = \emptyset$, that is, the arc $(i, j)$ has been deleted from $WS$, and $k1 = 1$. After fulfilling lines 7–8, the variable $v$ saves the $j$-th row of the matrix $Weight$, $l1 = 1+(i-1)h$, $l2 = ih$, and the variable $w1$ saves $wt(i, j)$. After performing lines 9–11, the variable $v$ saves the $s$-th row of the matrix $Dist$, $l1 = 1+(j-1)h$, $l2 = jh$, and the variable $w2$ saves the shortest path from $j$ to $s$. After fulfilling line 12, the variable $w3$ saves the value $wt(i, j) + dist(j, s)$. After performing lines 13–14, the variable $w$ saves $dist(i, s)$ because in view of line 9, the variable $v$ saves the $s$-th row of the matrix $Dist$.

Now we compare the words $w3$ and $w$ (line 15). If $w3 \geq w$, we go to the procedure end because $X = \emptyset$. Otherwise, we start to perform line 16. After fulfilling lines 16-17, the vertex $i$ is included into the slice $AffectedV$ and in the $s$-th row of the matrix $Dist$, the value $dist(i, s)$ is replaced with the

smaller value $w3$. After fulfilling lines 18–20 in the $u$-th row of the matrix $Dist1$, the value $dist(u,s)$ is replaced with the value $w3$. After performing lines 21–22, the slice $Y$ saves the tails of the not visited arcs entering the vertex $i$, for which the predicates $SP(q,i,s)$ are true. After performing line 23, we add these vertices to the slice $VisitedV$. Finally, by means of the cycle `while SOME(Y) do` (lines 24–28), such an arc entering the vertex $i$ is saved in the corresponding row of the matrix $WS$.

Step of induction. Let the assertion be true for $l \geq 1$. We prove it for the case when $l+1$ arcs are updated in $WS$. By the inductive assumption, after updating $l$ arcs in the matrix $WS$, their positions are marked with zero in the slice $X$, selected affected vertices are included into the slice *AffectedV*, the corresponding distance from every new affected vertex to the sink $s$ is replaced with the smaller value both in the matrix $Dist$ and in the matrix $Dist1$, the arcs entering any new affected vertex $q$ are saved in the matrix $WS$, if their tails $r$ are not visited vertices and they satisfy the predicates $SP(r,q,s)$.

Now we update the $(l+1)$-th arc in $WS$. Since $X \neq \emptyset$, we perform the cycle `while SOME(X) do`. Here we reason by analogy with the basis.

This completes the proof.

Now we evaluate the time complexity of the procedure InsertUpdate. Observe that it uses the auxiliary procedure ComputePred2. In [14], we have shown that it takes $O(h)$ time. Let $q$ be the number of affected vertices obtained after inserting the edge $(i,j)$ into $G$. We obtain that the procedure InsertUpdate takes $O(qh)$ time.

The associative version of the Ramalingam incremental algorithm for updating the all–pairs shortest paths runs as follows. At first, the arc $i \to j$ is inserted into $G$. Then the procedure InsertUpdate is applied for $s = j$ to define the set of *affected sink* vertices. After that the procedure InsertUpdate is applied to every affected sink vertex.

```
procedure InsertEdge(i,j,h,n: integer; v0: word(Trim);
  var Adj: table; var Weight,Cost,Dist,Dist1: table);
/* The arc (i,j) will be inserted into the graph G,
  h is the number of bits for coding the infinity. */
var l1,l2,z1: integer;
  v1: word(Weight);
  X,AffectedSinks: slice(Adj);
1. Begin v1:=ROW(j,Weight);
2.  l1:=1+(i-1)h; l2:=ih;
3.  REP(l1,l2,v0,v1);
4.  ROW(j,Weight):=v1;
/* We insert wt(i,j) in the matrix Weight. */
5.  v1:=ROW(i,Cost);
```

```
 6.  l1:=1+(j-1)h; l2:=jh;
 7.  REP(l1,l2,v0,v1);
 8.  ROW(i,Cost):=v1;
 /* We insert wt(i,j) in the matrix Cost. */
 9.  X:=COL(i,Adj); X(j):='1'. */
 /* We include the arc (i,j) in the matrix Adj. */
10.  InsertUpdate(i,j,h,n,j,Weight,Cost,Adj,
   Dist,Dist1,AffectedV);
11.  AffectedSinks:=AffectedV;
12.  while SOME(AffectedSinks) do
13.    begin z1:=STEP(AffectedSinks);
14.      InsertUpdate(i,j,h,n,z1,Weight,Cost,Adj,
   Dist,Dist1,AffectedV);
15.    end;
16. end;
```

Let us evaluate the time complexity of the procedure InsertEdge. Taking into account the time complexity of the procedure InsertUpdate, we obtain that the procedure InsertEdge takes $O(hkr)$ time per an insertion, where $h$ is the number of bits for coding the infinity, $k$ is the number of affected sink vertices that appear after inserting the arc $(i, j)$ in the graph $G$, and $r$ is the total sum of affected vertices for different sink vertices.

Now, we present the main advantage of the associative version of the Ramalingam incremental algorithm for updating the all–pairs shortest paths. This is finding in *parallel* the tails $y$ of arcs entering any affected vertex $u$ for which the predicates $SP(y, u, s)$ are true.

## 6. Conclusions

We have proposed the associative version of the Ramalingam incremental algorithm for updating the all–pairs shortest paths on the STAR–machine having no less than $n$ PEs. The associative version is represented as procedure InsertEdge whose correctness is proved. We have obtained that the procedure InsertEdge takes $O(hkr)$ time per an insertion, where $h$ is the number of bits for coding the infinity, $k$ is the number of affected sink vertices that appear after inserting the arc $(i, j)$ in the graph $G$, and $r$ is the total sum of affected vertices for different sink vertices. It is assumed that each microstep of the STAR–machine takes one unit of time. We have also presented the main advantage of the associative version of the Ramalingam incremental algorithm for updating the all–pairs shortest paths.

We are planning to build associative algorithm for finding the $k$ simple shortest paths in a directed weighted graph.

## References

[1] Ausiello G., Italiano G.F., Marchetti-Spaccamela A., and Nanni U. Incremental algorithms for minimal length paths // J. of Algorithms. – 1991. – Vol. 12, No. 4. – P. 615–638.

[2] Chaudhuri S., Zaroliagis C.D. Shortest path queries in digraphs of small treewidth // Proc. Intern. Colloquiumon Automata Languages, and Programming, Szeged, Hunhary, July 10–14, 1995. – Springer, 1995. – P. 244–255. – (Lect. Notes Comput. Sci.; 944).

[3] Demetrescu C. and Italiano G.F. Fully dynamic all–pairs shortest paths with real edge weights // Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS'01), Las Vegas, Nevada. – 2001. – P. 260–267.

[4] King V. Fully dynamic algorithms for maintaining all–pairs shortest paths and transitive closure in digraphs // Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99). – 1999. – P. 81–99.

[5] Klein P.N., Rao S., Rauch M., and Subramanian S. Faster shortest path algorithms for planar graphs // Proc. ACM Symposium on Theory of Computing, Montreal, Quebec, Canada. – 1994. – P. 27–37.

[6] 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.

[7] Nepomniaschaya, A. S. Basic associative parallel algorithms for vertical processing systems // Bulletin NCC. Series: Computer Science. – IIS Special Iss. 29. – NCC Publisher, 2009. – P. 63–77.

[8] Nepomniaschaya A.S. Associative parallel algorithms for dynamic edge update of minimum spanning trees // Proc. 7th Intern. Conf. PaCT 2003. – Springer, 2003. – P. 141–150. – (Lect. Notes Comput. Sci.; 2763)

[9] Nepomniaschaya A.S. Associative parallel algorithm for dynamic reconstructing a minimum spanning tree after deletion of a vertex // Proc. 8th Intern. Conf. PaCT 2005. – Springer, 2005. – P. 151–173. – (Lect. Notes Comput. Sci.; 3606)

[10] Nepomniaschaya A.S. Associative parallel algorithm for the dynamic update of a minimum spanning tree after insertion of a new vertex // Cybernetics and System Analysis. – Kiev: Naukova Dumka, 2006. – No. 1. – P. 19–31 (In Russian). (English translation by Plenum Press).

[11] Nepomniaschaya A. S. Efficient implementation of the Italiano algorithms for updating the transitive closure on associative parallel processors // Fundamenta Informaticae. – IOS Press, 1989. – No. 2–3. – 2008. – P. 313–329,

[12] Nepomniaschaya. A. S. Associative version of the Ramalingam decremental algorithm for dynamic updating the single-sink shortest paths subgraph // Proc.

of the 10-th Intern. Conf. on Parallel Computing Technologies, PaCT-2009, Novosibirsk, Russia. – Springer, 2009. – P. 257–268. – (Lect. Notes Comput. Sci.; 5698)

[13] Nepomniaschaya A. S. Associative version of the Ramalingam algorithm for the dynamic update of the shortest paths subgraph after inserting a new edge // Cybernetics and System Analysis. – Kiev: Naukova Dumka, 2012. – No. 3. – P. 45–57 (In Russian). (English translation by Springer).

[14] Nepomniaschaya A. S. Associative version of the Ramalingam decremental algorithm for the dynamic all–pairs shortest–path problem // Bulletin NCC. Series: Computer Science. – 2016. – Iss. 39. – P. 37–50.

[15] Ramalingam G. Bounded Incremental Computation. – Springer, 1996. – (Lect. Notes Comput. Sci.; 1089).