Incremental algorithms for optimal flows in networks

Laura A. Ciupală

***Abstract*—** In this paper, we will use incremental algorithms in order to save computational time when solving different network flow problems. We will focus on two important network flow problems: maximum flow problem and minimum cost flow problem.

Incremental algorithms are appropriated to be used when we have a network in which we already have established an optimal flow (in our case either a maximum flow or a minimum cost flow), but we must modify the network by inserting a new arc or by deleting an existent arc. An incremental algorithm starts with an optimal flow in the initial network and determines an optimal flow in the modified network.

First, we present incremental algorithms for the maximum flow problem. These algorithms were developed by S. Kumar and P. Gupta in 2003. They described algorithms for determining maximum flows in a network obtained from a given network in which a maximum flow is already known and in which a new arc is inserted or an existent arc is deleted.

Finally, we describe our incremental algorithms for the minimum cost flow problem. Let us consider a network in which we already established a minimum cost flow. We describe and solve the problem of establishing a minimum cost flow in this network after inserting a new arc and after deleting an existent arc. We focus on these problems because they arise in practice.

***Keywords*—** Incremental computation, Maximum flow, Minimum cost flow, Network algorithms, Network flow.

I. Introduction

NETWORK flow problems are a group of network optimization problems with widespread and diverse applications. The literature on network flow problems is extensive. Over the past 60 years researchers have made continuous improvements to algorithms for solving several classes of problems. From the late 1940s through the 1950s, researchers designed many of the fundamental algorithms for network flow, including methods for maximum flow and minimum cost flow problems. In the next decades, there are many research contributions concerning improving the computational complexity of network flow algorithms by using enhanced data structures, techniques of scaling the problem data etc.

One of the reasons for which the maximum flow problem and that minimum cost flow problem were studied so intensively is the fact that they arise in a wide variety of situations and in several forms.

We can save computational time by using incremental algorithms when the network, in which we know an optimal flow (in our case either a maximum flow or a minimum cost flow), is modified by inserting a new arc or by deleting an existent arc. After the network is modified, we need to find an optimal flow. A first way to solve this problem is to apply a maximum flow algorithm or a minimum cost algorithm starting from scratch. But, this is not the fastest way to solve the problem. The efficient way to solve this problem is to start from the optimal flow in the original network and to use an incremental algorithm which will gradually transform the optimal flow in the original network in an optimal flow in the modified network. Consequently, the incremental algorithms, that we will describe, are used to update solutions of optimal flow problems, after the network was modified by inserting a new arc or by deleting an existent arc.

In the following two sections we will describe incremental algorithms for maximum flows and for minimum cost flows respectively.

II. Maximum Flows

The maximum flow problem is one of the fundamental problems in network flow theory and it was studied extensively. The importance of the maximum flow problem is due to the fact that it arises in a wide variety of situations and in several forms. Sometimes the maximum flow problem occurs as a subproblem in the solution of more difficult network problems, such as the minimum cost flow problem or the generalized flow problem. The maximum flow problem also arises in a number of combinatorial applications that on the surface might not appear to be maximum flow problems at all. The problem also arises directly in problems as far reaching as machine scheduling, the assignment of program modules to computer processors, the rounding of census data in order to retain the confidentiality of individual households, tanker scheduling and several others.

The maximum flow problem was first formulated and solved using the well known augmenting path algorithm by Ford and Fulkerson in 1956. Since then, two types of maximum flow algorithms have been developed: augmenting path algorithms and preflow algorithms:

1) The augmenting path algorithms maintain mass

Manuscript received August 2, 2011: Revised version received January 24, 2012.

L. A. Ciupală is with the Department of Computer Science, Transilvania University of Braşov, Romania (corresponding author to provide phone: 0040-268414016; fax: 0040-268414016; e-mail: laura_ciupala@ yahoo.com).

balance constraints at every node of the network other than the source node and the sink node. These algorithms incrementally augment flow along paths from the source node to the sink node. By determining the augmenting paths with respect to different selection rules, different algorithms were developed. For details see [1].

2) The preflow algorithms flood the network so that some nodes have excesses. These algorithms incrementally relieve flow from nodes with excesses by sending flow from the node forward toward the sink node or backward toward the source node. By imposing different rules for selecting nodes with excesses, different preflow algorithms were obtained. These algorithms are more versatile and more efficient than the augmenting path algorithms. For details see [1].

A. Notation and Definitions

Without any loss of generality, we can consider a network with zero lower bounds, because any maximum flow problem in a network with positive lower bounds can be transformed in an equivalent maximum flow problem in a network with zero lower bounds (for details see [1]).

Let $G = (N, A, c, s, t)$ be a capacitated network with a nonnegative capacity $c(i, j)$ associated with each arc $(i, j) \in A$. We distinguish two special nodes in the network G: a source node s and a sink node t.

Let $n=|N|$, $m = |A|$ and $C = \max \{c(i, j) \mid (i, j) \in A\}$.

A *flow* is a function $f : A \rightarrow \mathbf{R}_+$ satisfying the next conditions:

$$f(s, N) - f(N, s) = v \quad (1)$$
$$f(i, N) - f(N, i) = 0, \; i \neq s,t \quad (2)$$
$$f(t, N) - f(N, t) = -v \quad (3)$$
$$0 \leq f(i, j) \leq c(i, j), \; (i, j) \in A \quad (4)$$

for some $v \geq 0$

We refer to v as the *value* of the flow f.

The maximum flow problem is to determine a flow f for which v is maximized.

For the maximum flow problem, a *preflow* is a function $f : A \rightarrow \mathbf{R}_+$ satisfying the next conditions:

$$f(i, N) - f(N, i) \geq 0, \; i \neq s,t \quad (5)$$
$$0 \leq f(i, j) \leq c(i, j), \; (i, j) \in A \quad (6)$$

Let f be a preflow. We define the *excess* of a node $i \in N$ in the following manner:

$$e(i) = f(i, N) - f(N, i) \quad (7)$$

Thus, for the maximum flow problem, for any preflow f, we have:

$$e(i) \geq 0, \; i \in N \setminus \{s, t\}.$$

We say that a node $i \in N \setminus \{s, t\}$ is *active* if $e(i) > 0$ and *balanced* if $e(i) = 0$.

A preflow f for which

$$e(i) = 0, \; i \in N \setminus \{s, t\}$$

is a flow. Consequently, a flow is a particular case of preflow.

A *pseudoflow* is a function $f : A \rightarrow \mathbf{R}_+$ satisfying the only conditions (4).

For any pseudoflow f, we define the *imbalance* of node i as

$$e(i) = v(i) + f(N, i) - f(i, N), \quad \text{for all } i \in N.$$

If $e(i) > 0$ for some node i, we refer to $e(i)$ as the *excess* of node i; if $e(i) < 0$, we refer to $-e(i)$ as the *deficit* of node i. If $e(i) = 0$ for some node i, we refer to node i as the *balanced*. Consequently, a preflow is a particular case of psedoflow.

For the maximum flow problem, the *residual capacity* $r(i, j)$ of any arc $(i, j) \in A$, with respect to a given psedoflow f, is given by

$$r(i, j) = c(i, j) - f(i, j) + f(j, i).$$

By convention, if $(i, j) \in A$ and $(j, i) \notin A$, then we add the arc (j, i) to the set of arcs A and we set $c(j, i) = 0$. The residual capacity $r(i, j)$ of the arc (i, j) represents the maximum amount of additional flow that can be sent from the node i to node j using both of the arcs (i, j) and (j, i).

The network $G(f) = (N, A(f))$ consisting only of those arcs with strictly positive residual capacity is referred to as the *residual network* (with respect to the given pseudoflow f).

A directed path from the source node s to the sink node t in the residual network $G(f) = (N, A(f))$ is called an *augmenting path*.

Let P be an augmenting path. Then

$$r(P) = \min\{r(i, j) \mid (i, j) \in P\}$$

is the *residual capacity* of P.

If in the residual network $G(f) = (N, A(f))$ there is an augmenting path P then we can send $r(P)$ units of flow along the path P, obtaining in this way a flow whose value is with $r(P)$ units greater than the value of the initial flow f.

Theorem 1.([1]) *A flow f in the network* $G = (N, A, c, s, t)$ *is a maximum flow if and only if the residual network* $G(f) = (N, A(f))$ *contains no augmenting paths.*

In the residual network $G(f) = (N, A(f))$ the *distance function* $d : N \rightarrow \mathbf{N}$ with respect to a given preflow f is a function from the set of nodes to the nonnegative integers.

We say that a distance function is *valid* if it satisfies the following validity conditions:

$$d(t) = 0$$
$$d(i) \leq d(j) + 1, \text{ for every arc } (i, j) \in A(f).$$

We refer to $d(i)$ as the distance label of node i.

Theorem 2.([1])*(a) If the distance labels are valid, the distance label* $d(i)$ *is a lower bound on the length of the shortest directed path from node i to sink node t in the residual network.*

(b) If $d(s) \geq n$, *the residual network contains no directed path from the source node s to the sink node t.*

B. Incremental algorithms for the maximum flow problem

In this subsection we describe incremental algorithms which update the solution of a maximum flow problem after inserting a new arc and after deleting an arc. These algorithms were described by S. Kumar and P. Gupta in [19].

First, we will focus on the problem of determining a maximum flow in a network after inserting a new arc.

Let $G=(N, A, c, b, v)$ be a network in which we already determined a maximum flow f. Let us insert a new arc (k, l) with capacity $c(k, l)$ into the network G, obtaining in this way the network $G'= (N, A', c', b', v)$, where:

$A' = A \cup \{(k, l)\}$

$c'(i, j)= c(i, j), \forall (i\, j) \in A'$

The new network G' can contain a maximum flow f' with a greater value than f – the maximum flow in G. This is possible because when this new arc (k, l) is inserted in the network G there may appear new augmenting paths through which additional flow can be sent. We will refer to a node that is contained in at least one of these new augmenting paths as an *affected node*. The set of all affected nodes will be denoted by AN.

The affected nodes are contained in the augmenting paths from the source node s to the node k and in the augmenting paths from the node l to the sink node t. Consequently, they can be determined by applying a modified Backward Breadth First Search algorithm from node k to node s and by applying a modified Breadth First Search algorithm starting from node l to node t.

The modified Backward Breadth First Search algorithm from node k to node s will explore only those nodes y for which the distance label $d(y) \geq n$ because, from Theorem 2, only those nodes are the candidates to the set AN.

The modified Backward Breadth First Search (BBFS) algorithm is the following:

BBFS(v, u) Algorithm;
Begin
$W = \{v\}$;
$AN = \{v\}$;
while $W \neq \emptyset$ **do**
begin
remove a node x from W;
if $x = v$ **then** break;
for each $(y, x) \in A$ **do**
if $y \notin AN$ **and** $r(y, x) >0$ **and** $d(y) \geq n$ **then**
begin
$W = W \cup \{y\}$;
$AN = AN \cup \{y\}$;
end;
end
end.

The modified Breadth First Search algorithm from node l to node t will explore only those nodes y for which the distance label $d(y) < n$ because, from Theorem 2, only those nodes are the candidates to the set AN.

The modified Breadth First Search (BFS) algorithm is the following:

BFS(v, u) Algorithm;
Begin
$W = \{v\}$;
$AN = \{v\}$;
while $W \neq \emptyset$ **do**
begin
remove a node x from W;
if $x = u$ **then** break;
for each $(x, y) \in A$ **do**
if $y \notin AN$ **and** $r(x, y) >0$ **and** $d(y) < n$ **then**
begin
$W = W \cup \{y\}$;
$AN = AN \cup \{y\}$;
end;
end
end.

The incremental algorithm for updating the solution of a maximum flow in a network after inserting a new arc follows the approach of the generic preflow algorithm ([1]) developed by Goldberg and Tarjan.

Incremental Add Max Flow Algorithm;
Begin
let f be a maximum flow in the network G;
determine the new network G' obtained from G by inserting a new arc (k, l);
determine the residual network $G'(f)$;
if $d(k) \geq n$ **and** $d(l) < n$ **then**
begin
call BBFS(k, s) to determine the set $AN1$ of the affected nodes that lie on the augmenting paths from the source node s to the node k;
call BFS(l, t) to determine the set $AN2$ of the affected nodes that lie on the augmenting paths from the node l to the sink node t;
$AN = AN1 \cup AN2$;
compute the exact distance labels for all the nodes $i \in AN$;
for each arc $(s, j) \in A'(f)$ **do**
if $j \in AN$ **then**
$f(s, j) = c'(s, j)$;
$d(s) = n$;
while the network contains an active node **do**
begin
select an active node i;
push/relabel(i);
end
end
end.

procedure *push/relabel*(i);
begin

if the network contains an admissible arc (i, j) **and** $j \in AN$ **then**
push $g = \min(e(i), r(i, j))$ units of flow from node i to node j;
else $d(i) = \min\{ d(j) \mid (i, j) \in A'(f)$ and $j \in AN \} + 1$
end;

A push of g units of flow from node i to node j means that:
$e(i) = e(i) - g$
$e(j) = e(j) + g$
$r(i, j) = r(i, j) - g$
$r(j, i) = r(j, i) + g$.

Theorem 3.([19]) *The incremental add max flow algorithm computes correctly a maximum flow in the network G', obtained from G by inserting a new arc* (k, l).

Theorem 4.([19]) *The incremental add max flow algorithm runs in* $O(|AN|^2 m)$ *time.*

Now we will study the problem of determining a maximum flow in a network after deleting an arc.

Let $G=(N, A, c, b, v)$ be a network in which we already determined a maximum flow f. Let (k, l) be an arbitrary arc of G. The arc (k, l) will be removed from the network G, obtaining in this way the network $G'= (N, A', c', b', v)$, where:

$A' = A \setminus \{(k, l)\}$
$c'(i, j) = c(i, j), \forall (i j) \in A'$

Let f be a maximum flow in G and let f' be a maximum flow in the network G' obtained from G through deletion of the arc (k, l). The value of the flow f' might by smaller than the value of f.

In the network G' obtained by deleting the arc (k, l) from G, it is possible that f is no longer a flow, but a pseudoflow. More precisely, if in G the flow on the arc (k, l) was strictly positive, then, in G', f will no longer satisfy the mass balance constraints (2) for both of the nodes k and l. This means that, in G', node k will have a strictly positive excess. The incremental algorithm will try to push this excess toward the sink node t using alternate augmenting path in the modified network G'. The incremental algorithm for determining a maximum flow in a network after removing an arc is the following:

Incremental Del Max Flow Algorithm;
Begin
let f be a maximum flow in the network G;
determine the residual network $G'(f)$;
if $f(k, l) > 0$ **then**
begin
call BFS(l, t) to determine the set $AN1$ of the affected nodes that lie on the augmenting paths from the node l to the sink node t;
determine the inverse network $G^{-1}(f) = (N, A^{-1}(f))$ by reversing the arcs in the residual network $G(f)$;
for each node $j \in AN1 \setminus \{t\}$ **do**
$d^{-1}(j) = 0$;
$d^{-1}(t) = |AN1|$;
call *push/relabel*(l) in the inverse network $G^{-1}(f)$;
determine the new network G' obtained from G by deleting the arc (k, l);
determine the residual network $G'(f)$;
call **B**BFS(k, s) to determine the set $AN2$ of the affected nodes that lie on the augmenting paths from the source node s to the node k;
for each node $j \in AN2$ **do**
$d(j) = 0$;
call *push/relabel*(l) in the residual network $G'(f)$;
call *push/relabel*(k) in the residual network $G'(f)$;
end
end.

Theorem 5.([19]) *The incremental del max flow algorithm computes correctly a maximum flow in the network G', obtained from G by deleting the arc* (k, l).

Theorem 6.([19]) *The incremental del max flow algorithm runs in* $O(|AN|^2 m)$ *time*.

III. Minimum Cost Flows

The minimum cost flow problem, as well as one of its special cases which is the maximum flow problem, is one of the fundamental problems in network flow theory and it was studied extensively. The importance of the minimum cost flow problem is also due to the fact that it arises in almost all industries, including agriculture, communications, defense, education, energy, health care, medicine, manufacturing, retailing and transportation. Indeed, minimum cost flow problems are pervasive in practice.

A. Notation and Definitions

Let $G = (N, A)$ be a directed graph, defined by a set N of n nodes and a set A of m arcs. Each arc $(i, j) \in A$ has a capacity $c(i, j)$ and a cost $b(i, j)$. We associate with each node $i \in N$ a number $v(i)$ which indicates its supply or demand depending on whether $v(i) > 0$ or $v(i) < 0$. In the directed network $G = (N, A, c, b, v)$, the minimum cost flow problem is to determine the flow $f(i, j)$ on each arc $(i, j) \in A$ which

$$\text{minimize} \quad \sum_{(i,j)\in A} b(i, j) f(i, j) \tag{8}$$

subject to

$$\sum_{j|(i,j)\in A} f(i, j) - \sum_{j|(j,i)\in A} f(j, i) = v(i), \forall i \in N \tag{9}$$

$$0 \le f(i, j) \le c(i, j), \forall (i, j) \in A\,. \tag{10}$$

A flow f satisfying the conditions (9) and (10) is referred to as a *feasible flow*.

Let C denote the largest magnitude of any supply/demand or finite arc capacity, that is

$C = \max(\max\{v(i) \mid i \in N\}, \max\{c(i, j) \mid (i, j) \in A, c(i, j) < \infty\})$

and let B denote the largest magnitude of any arc cost, that is

$$B = \max\{b(i, j) \mid (i, j) \in A\}.$$

The *arc adjacency list* or, shortly, the *arc list* of a node i is the set of arcs emanating from that node, that is:

$$A(i) = \{(i, j) \mid (i, j) \in A\}.$$

The residual network $G(f) = (N, A(f))$ corresponding to a flow f is defined as follows. We replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i). The arc (i, j) has the cost $b(i, j)$ and the residual capacity $r(i, j) = c(i, j) - f(i, j)$ and the arc (j, i) has the cost $b(j, i) = -b(i, j)$ and the residual capacity $r(j, i) = f(i, j)$. The residual network consists only of arcs with positive residual capacity.

We shall assume that the minimum cost flow problem satisfies the following assumptions:

Assumption 1. The network is directed.

This assumption can be made without any loss of generality. In [1] it is shown that we can always fulfil this assumption by transforming any undirected network into a directed network.

Assumption 2. All data (cost, supply/demand and capacity) are integral.

This assumption is not really restrictive in practice because computers work with rational numbers which we can convert into integer numbers by multiplying by a suitably large number.

Assumption 3. The network contains no directed negative cost cycle of infinite capacity.

If the network contains any such cycles, there are flows with arbitrarily small costs.

Assumption 4. All arc costs are nonnegative.

This assumption imposes no loss of generality since the arc reversal transformation described in [1] converts a minimum cost flow problem with negative arc costs to one with nonnegative arc costs. This transformation can be done if the network contains no directed negative cost cycle of infinite capacity.

Assumption 5. The supplies/demands at the nodes satisfy the condition $\sum_{i \in N} v(i) = 0$ and the minimum cost flow problem has a feasible solution.

Assumption 6. The network contains an uncapacitated directed path (i.e. each arc in the path has infinite capacity) between every pair of nodes.

We impose this condition by adding artificial arcs $(1, i)$ and $(i, 1)$ for each $i \in N$ and assigning a large cost and infinite capacity to each of these arcs. No such arc would appear in a minimum cost solution unless the problem contains no feasible solution without artificial arcs.

We associate a real number $\pi(i)$ with each node $i \in N$. We refer to $\pi(i)$ as the *potential* of node i. These node potentials are generalizations of the concept of distance labels that we used in previous section.

For a given set of node potentials π, we define the reduced cost of an arc (i, j) as

$$b^{\pi}(i, j) = b(i, j) - \pi(i) + \pi(j).$$

The reduced costs are applicable to the residual network as well as to the original network.

Theorem 7. ([1]) (a) For any directed path P from node h to node k we have

$$\sum_{(i,j) \in P} b^{\pi}(i, j) = \sum_{(i,j) \in P} b(i, j) - \pi(h) + \pi(k)$$

(b) For any directed cycle W we have

$$\sum_{(i,j) \in W} b^{\pi}(i, j) = \sum_{(i,j) \in W} b(i, j).$$

Theorem 8. (**Negative Cycle Optimality Conditions)** ([1]) A feasible solution f is an optimal solution of the minimum cost flow problem if and only if the residual network $G(f)$ contains no negative directed cycle.

Theorem 9. (**Reduced Costs Optimality Conditions)** ([1]) A feasible solution f is an optimal solution of the minimum cost flow problem if and only if some set of node potentials π satisfy the following reduced cost optimality conditions:

$b^{\pi}(i, j) \geq 0$ for every arc (i, j) in the residual network $G(f)$.

Theorem 10.(Complementary Slackness Optimality Conditions) ([1]) A feasible solution f is an optimal solution of the minimum cost flow problem if and only if for some set of node potentials π, the reduced cost and flow values satisfy the following complementary slackness optimality conditions for every arc $(i, j) \in A$:

If $b^{\pi}(i, j) > 0$, then $f(i, j) = 0$ (11)
If $0 < f(i, j) < c(i, j)$, then $b^{\pi}(i, j) = 0$ (12)
If $b^{\pi}(i, j) < 0$, then $f(i, j) = c(i, j)$ (13)

The residual network corresponding to a pseudoflow is defined in the same way that we define the residual network for a flow.

The optimality conditions can be extended for pseudoflows. A pseudoflow f^* is optimal if there are some set of node

potentials π such that the following reduced cost optimality conditions are satisfied:

$$b^{\pi}(i,j) \geq 0 \text{ for every arc } (i,j) \text{ in the residual network } G(f^*).$$

We refer to a flow or a pseudoflow f as ε-*optimal* for some $\varepsilon>0$ if for some node potentials π, the pair (f, π) satisfies the following ε-*optimality conditions*:

$$\text{If } b^{\pi}(i,j) > \varepsilon, \text{ then } f(i,j) = 0 \quad (14)$$
$$\text{If } -\varepsilon \leq b^{\pi}(i,j) \leq \varepsilon, \text{ then } 0 \leq f(i,j) \leq c(i,j) \quad (15)$$
$$\text{If } b^{\pi}(i,j) < -\varepsilon, \text{ then } f(i,j) = c(i,j) \quad (16)$$

These conditions are relaxations of the (exact) complementary slackness optimality conditions (11) - (13) and they reduce to complementary slackness optimality conditions when $\varepsilon = 0$.

The algorithms for determining a minimum cost flow rely upon the optimality conditions stated by Theorems 8, 9 and 10.

The basic algorithms for minimum cost flow can be divided into two classes: those that maintain feasible solutions and strive toward optimality and those that maintain infeasible solutions that satisfy optimality conditions and strive toward feasibility. Algorithms from the first class are: the cycle-canceling algorithm and the out-of-kilter algorithm. The cycle-canceling algorithm maintains a feasible flow at every iteration, augments flow along negative cycle in the residual network and terminates when there is no more negative cycle in the residual network, which means (from Theorem 8) that the flow is a minimum cost flow. The out-of-kilter algorithm maintains a feasible flow at each iteration and augments flow along shortest path in order to satisfy the optimality conditions. Algorithms from the second class are: the successive shortest path algorithm and primal-dual algorithm. The successive shortest path algorithm maintains a pseudoflow that satisfies the optimality conditions and augments flow along shortest path from excess nodes to deficit nodes in the residual network in order to convert the pseudoflow into an optimal flow. The primal-dual algorithm also maintains a pseudoflow that satisfies the optimality conditions and solves maximum flow problems in order to convert the pseudoflow into an optimal flow.

Starting from the basic algorithms for minimum cost flow, several polynomial-time algorithms were developed. Most of them were obtained by using the scaling technique. By capacity scaling, by cost scaling or by capacity and cost scaling, the following polynomial-time algorithms were developed: capacity scaling algorithm, cost scaling algorithm, double scaling algorithm, repeated capacity scaling algorithm and enhanced capacity scaling algorithm.

Another approach for obtaining polynomial-time algorithms is to select carefully the negative cycles in the cycle-canceling algorithm.

B. Incremental algorithms for the minimum cost flow problem

In this subsection we describe incremental algorithms which update the solution of a minimum cost flow problem after inserting a new arc and after deleting an arc.

First, we will study the problem of determining a minimum cost flow in a network after inserting a new arc.

Let $G=(N, A, c, b, v)$ be a network in which we already determined a minimum cost flow f. Let us insert a new arc (k, l) with capacity $c(k, l)$ and cost $b(k, l)$ into the network G, obtaining in this way the network $G'= (N, A', c', b', v)$, where:

$A' = A \cup \{(k, l)\}$
$c'(i,j)= c(i,j), \forall (i\,j) \in A'$
$b'(i,j)= b(i,j), \forall (i\,j) \in A'$

The new network G' can contain a minimum cost flow f' with a smaller cost than f – the minimum cost flow in G.

In the network G' obtained by inserting a new arc (k, l) in G, it is possible that the optimality conditions are not fulfilled with respect to f which was a minimum cost flow in G. It is possible that the residual network $G'(f)$ contains a negative cycle, which means that the Negative cycle optimal conditions from Theorem 8 are not satisfied. The incremental algorithm for determining a minimum cost flow in a network after inserting a new arc is based on these optimality conditions:

Incremental Add Algorithm;
Begin
let f be a minimum cost flow in the network G;
determine the new network G' obtained from G by inserting a new arc (k, l);
while the residual network $G'(f)$ contains a negative cycle **do begin**
determine a negative cycle C in $G'(f)$;
compute $r(C) = \min\{r(i,j) \mid (i,j) \in C\}$;
send $r(C)$ units of flow along the cycle C;
update the residual network $G'(f)$;
end
end.

Theorem 11. *The incremental add algorithm computes correctly a minimum cost flow in the network G'*, obtained from G by inserting a new arc(k, l).

Proof. The algorithm terminates when the residual network $G'(f)$ does not contain any negative cycles. By Theorem 8, it follows that the flow f is a minimum cost flow.

Theorem 12. *The incremental add algorithm runs in* $O(nmc(k, l))$ *time.*

Proof. Because f, the flow with which the algorithm starts, is a minimum cost flow in the network G, it follows that the new network G' obtained from G by inserting a new arc (k, l) could contain only negative cycles that contain the arc (k, l).

At each iteration of the **while** loop, such a cycle C is determined in $O(nm)$ time. Sending $r(C)$ units of flow along the cycle C means reducing its residual capacity by $r(C)$. Accordingly to Assumption 2, all data are integer. It follows that $r(C)$ is also an integer number. Consequently, $r(C) \geq 1$. Thus, after at most $c(k, l)$ iterations, the network will contain no negative cycle. This implies that in $O(nmc(k, l))$ time the algorithm determines a minimum cost flow.

Now we will study the problem of determining a minimum cost flow in a network after removing an arc.

Let $G=(N, A, c, b, v)$ be a network in which we already determined a minimum cost flow f. Let (k, l) be an arbitrary arc of G. The arc (k, l) will be removed from the network G, obtaining in this way the network $G'= (N, A', c', b', v)$, where:

$A' = A \setminus \{(k, l)\}$

$c'(i, j)= c(i, j), \forall (i j) \in A'$

$b'(i, j)= b(i, j), \forall (i j) \in A'$

Let f be the minimum cost flow in G and let f' be the minimum cost flow in the network G' obtained from G through deletion of the arc (k, l). The cost of the flow f' might by greater than the cost of f.

In the network G' obtained by deleting the arc (k, l) from G, it is possible that the optimality conditions are not fulfilled with respect to f which was a minimum cost flow in G. More precisely, if in G the flow on the arc (k, l) was strictly positive, then in G' f will no longer satisfy the mass balance constraints (2) for both of the nodes k and l. This means that, in G', f is not a flow, but a pseudoflow. We will transform this pseudoflow into a minimum cost flow, by sending flow from the node k to the node l in the network G' using the following algorithm.

Incremental Delete Algorithm;
Begin
let f be a minimum cost flow in the network G;
compute a set of optimal node potentials π with respect to the minimum cost flow f;
$e(k) = f(k, l)$;
$e(l) = -f(k, l)$;
determine the new network G' obtained from G by deleting the arc (k, l);
determine the residual network $G'(f)$;
while $e(k) > 0$ **do begin**
determine shortest path distances d from k to all other nodes in the residual network $G'(f)$ with respect to the reduced costs;
let P be a shortest path from k to l;
$\pi = \pi - d$;
$r(P) = \min(e(k), \min\{r(i, j) \mid (i, j) \in P\})$;
send $r(P)$ units of flow along the path P;
update the residual network $G'(f)$ and the reduced costs;
end
end.

Theorem 13. *The incremental delete algorithm computes correctly a minimum cost flow in the network G', obtained from G by deleting the arc* (k, l).

Proof. The algorithm terminates when $e(k) = 0$, which means that k is a balanced node. Because all nodes excepting k and l are balanced at the beginning of the algorithm and remain balanced during the algorithm, it follows that at the end of the algorithm also l is a balanced node. This implies that f is a flow. By Theorem 9, it follows that the flow f is a minimum cost flow.

Theorem 14. *The incremental delete algorithm runs in* $O(S(n, m)+f(k, l)S'(n, m))$ *time, where* $S(n, m)$ *is the time needed to solve a shortest path problem with possible negative arc lengths and* $S'(n, m)$ *is the time needed to solve a shortest path problem with nonnegative arc lengths.*

Proof. For determining a set of optimal potentials, a shortest path algorithm is applied in the residual network $G(f)$. This network contains no negative cycle because f is a minimum cost flow in the network G, which must satisfy the negative cycle optimality conditions from Theorem 8. But it is not mandatory that all lengths in the residual network $G(f)$ to be nonnegative. Thus, the time complexity for determining a set of optimal potentials is $S(n, m)$.

Because f is a flow, all the nodes in G are balanced. After deleting the arc (k, l), if the flow on this arc was strictly positive, we have an excess of $f(k, l)$ units at node k and a deficit of $f(k, l)$ units at node l.

At each iteration of the **while** loop, we solve a shortest path problem from node k to all the other nodes in the residual network $G'(f)$ with respect to the reduced costs, which are nonnegative. Sending $r(P)$ units of flow along the shortest path P from k to l means reducing the excess of k by $r(P)$. Accordingly to Assumption 2, all data are integer. It follows that $r(P)$ is also an integer number. Consequently, $r(P) \geq 1$. Thus, after at most $f(k, l)$ iterations, the excess of the node k will be reduced to 0, which implies that the deficits of the node l will be also 0 and all the nodes will be balanced. Equivalently, f will be a flow. Moreover, it will be a minimum cost flow. Consequently, the **while** loop will take $O(f(k, l)S'(n, m))$ time and the algorithm will run in $O(S(n, m)+f(k, l)S'(n, m))$ time.

IV. Conclusion

In this paper, we focused on updating a given solution of an optimal flow problem, which could be a maximum flow problem or a minimum cost flow problem, after the network is modified. We studied two possible modifications of the network: the first one implies inserting a new arc and the second one consists in deleting an existent arc. We described incremental algorithms for updating the solution of a maximum flow problem and of a minimum cost flow problem after both modifications of the network. These incremental algorithms save important computational time.

We only studied the problems of updating a solution of an

optimal flow problem (maximum flow problem and minimum cost flow problem) after inserting or deleting an arc. But, the problems of updating a solution after increasing or decreasing the capacity of an arc can be reduced to one of these studied problems.

REFERENCES

[1] R. Ahuja, T. Magnanti and J. Orlin, *Network flows. Theory, algorithms and applications*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1993.

[2] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, London, 2001.

[3] J. Barros, S.D. Servetto, "Network Information Flow with Correlated Sources", *IEEE Transactions on Information Theory*, vol. 52(1), pp. 155-170, 2006.

[4] L. Ciupală, "The wave preflow algorithm for the minimum flow problem", *Proceedings of the 10th WSEAS International Conference on Mathematical and Computational Methods in Science and Engineering*, 2008, pp. 473-476.

[5] L. Ciupală, "A deficit scaling algorithm for the minimum flow problem", *Sadhana* Vol.31, No. 3, pp.1169-1174, 2006.

[6] L. Ciupală, "A scaling out-of-kilter algorithm for minimum cost flow", *Control and Cybernetics* Vol.34, No.4, pp. 1169-1174, 2005.

[7] L. Ciupală and E. Ciurea, "A highest-label preflow algorithm for the minimum flow problem", *Proceedings of the 11th WSEAS International Conference on Computers*, 2007, pp. 565-569.

[8] L. Ciupală and E. Ciurea, "About preflow algorithms for the minimum flow problem", *WSEAS Transactions on Computer Research* vol. 3 nr.1, pp. 35-41, January 2008.

[9] L. Ciupală and E. Ciurea, "Sequential and parallel deficit scaling algorithms for the minimum flow in bipartite networks", *WSEAS Transactions on Computer Research* vol. 7, pp. 1545-1554, October 2008.

[10] L. Ciupală and E. Ciurea, "An algorithm for the minimum flow problem", *The Sixth International Conference of Economic Informatics*, 2003, pp. 167-170.

[11] L. Ciupală and E. Ciurea, "An approach of the minimum flow problem", *The Fifth International Symposium of Economic Informatics*, 2001, pp. 786-790.

[12] E. Ciurea and L. Ciupală, "Sequential and parallel algorithms for minimum flows", *Journal of Applied Mathematics and Computing* Vol.15, No.1-2, pp. 53-78, 2004.

[13] E. Ciurea and L. Ciupală, "Algorithms for minimum flows", *Computer Science Journal of Moldova* Vol.9, No.3(27), pp. 275-290, 2001.

[14] A. Deaconu. E. Ciurea and C. Marinescu, "A Study on the Feasibility of the Inverse Maximum Flow Problems and Flow Modification Techniques in the Case of Non-Feasibility", *WSEAS Transactions on Computers*, Issue 10, Volume 9, pp. 1098-1107, 2010.

[15] A. Deshpande, S. Patkar and H. Narayanan, "Submodular Theory Based Approaches For Hypergraph Partitioning", *WSEAS Transactions on Circuit and Systems*, Issue 6, Volume 4, pp. 647-655, 2005.

[16] V. Goldberg and R. E. Tarjan, "A New Approach to the Maximum Flow Problem", *Journal of ACM* Vol.35, pp. 921-940, 1988.

[17] S. Fujishige, "A maximum flow algorithm using MA ordering", *Operation Research Letters* 31, No. 3, pp. 176-178, 2003.

[18] S. Fujishige and S. Isotani, "New maximum flow algorithms by MA orderings and scaling", *Journal of the Operational Research Society of Japan* 46, No. 3, pp. 243-250, 2003.

[19] S. Kumar and P. Gupta, "An incremental algorithm for the maximum flow problem", *Journal of Mathematical Modelling and Algorithms* 2, No.1, pp. 1-16, 2003.

[20] S. Patkar, H. Sharma and H. Narayanan, "Efficient Network Flow based Ratio-cut Netlist Hypergraph Partitioning", *WSEAS Transactions on Circuits and Systems* vol. 3, no. 1, pp. 47-53, January 2004.

[21] A. Schrijver, "On the history of the transportation and maximum flow problems", *Mathematical Programming* 91, No.3, pp. 437-445, 2002.

[22] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphiaa, Pennsylvania, 1983.

[23] K.D. Wayne, "A polynomial Combinatorial Algorithm for Generalized Minimum Cost Flow", *Mathematics of Operations Research*, pp. 445-459, 2002.

[24] L. Zhou and J.Zheng, "A New Immune Clone Algorithm to solve the constrained optimization problems", *WSEAS Transactions on Computers*, Issue 4, Volume 10, pp. 105-114, 2011.