INTERNATIONAL JOURNAL OF COMPUTERS
Issue 4, Volume 2, 2008

Improved algorithms for minimum flows in bipartite networks

Eleonor Ciurea, Oana Georgescu, Daniela Marinescu

***Abstract*—In this paper we study minimum flow algorithms for bipartite networks. We present two classes of algorithms for finding minimum flow in bipartite networks. The time bounds for several minimum flow algorithms automatically improve when the algorithms are applied without modification on bipartite networks. We obtain further running time improvements by modifying the algorithms. This modification applies only to preflow algorithms. In the final part of the paper we present an example for one of these algorithms.**

***Index Terms*—bipartite networks, minimum flow problem, network algorithms, network flow**

I. Introduction

THE theory of flows is one of the most important parts of Combinatorial Optimization.

The computation of a maximum flow in a graph has been an important and well studied problem, both in the field of computer science and operations research. Many efficient algorithms have been developed to solve this problem, see, e.g., [1]. By improving the running times of Dinic's [8] and Karzanov's [12] algorithms, Gusfield, Martel and Fernandez-Baca [11] have developed the first specializations of maximum flow algorithms for bipartite networks. Ahuja, Orlin, Stein and Tarjan [2] provided further improvements and proved that it is possible to obtain new time bounds for bipartite networks. The paper by Gusfield, Martel and Fernandez-Baca [11] describes several problems which can be solved using network flow in a bipartite graph.

The computation of a minimum flow in a network has been investigated by the authors in [3], [4], [5], [6], [7], [9], [10]. The minimum flow problem in bipartite network was not treated by other authors.

The brief outline of the paper is as follows: in Section 2 we discuss some basic notions and results used in the rest of the paper. Section 3 deals with original algorithms for minimum flow in bipartite networks. In Section 4 we present specialization algorithms for minimum flow in bipartite networks. Section 5 deals with an example for one of these algorithms.

In the next presentation we assume familiarity with preflow algorithms and we omit many details, since they are straightforward modifications of known results. The reader interested in further details is urged to consult the book [1] for maximum flow problem and the paper [5] for minimum flow problem.

E. Ciurea, O. Georgescu and D. Marinescu are with the Department of Theoretical Computer Science, *Transivania* University of Brasov, Str. Iuliu Maniu nr. 50, 500091, Brasov, ROMANIA, e-mail: {e.ciurea, o.georgescu, mdaniela}@unitbv.ro.

Manuscript received December 10, 2008; revised .

II. Terminology and Preliminaries

In this section we discuss some basic notions and results used in the rest of the paper.

For the interest of this paper, we consider a capacitated network $G = (N, A, l, c, s, t)$ with a nonnegative capacity $c(x, y)$ and with a nonnegative lower bounds function $l(x, y)$ associated with each arc $(x, y) \in A$. We distinguish two special nodes in the network G: a source node s and a sink node t. We further assume, without loss of generality, that the network contains no parallel edges.

For a given pair of not necessarily disjoint subsets X, Y of the nodes set N of a network G we use the notation:

$$(X, Y) = \{(x, y) | (x, y) \in A, x \in X, y \in Y\}$$

and for a given function f on arcs set A we use the notation:

$$f(X, Y) = \sum_{(X,Y)} f(x, y)$$

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

$$f(x, N) - f(N, x) = \begin{cases} v, & \text{if } x = s \\ 0, & \text{if } x \neq s, t \\ -v, & \text{if } x = t \end{cases} \qquad (1.a)$$

$$l(x, y) \leq f(x, y) \leq c(x, y), \forall (x, y) \in A, \qquad (1.b)$$

for some $v \geq 0$. We refer to v as the *value of the flow f*.

The maximum (minimum) flow problem is to determine a flow f for which v is maximized (minimized).

A *cut* is a partition of the nodes set N into two subsets S and $T = N$-S. We represent this cut using notation

$[S,T]$. We refer to a cut $[S,T]$ as an *s-t cut* if $s \in S$ and $t \in T$. We refer to an arc (x,y) with $x \in S$ and $y \in T$ as a *forward arc* of the cut and an arc (x,y) with $x \in T$ and $y \in S$ as a *backward arc* of the cut. Let (S,T) denote the set of forward arcs in the cut and let (T,S) denote the set of backward arcs.

For the maximum flow problem, we define the *capacity* $\widetilde{c}[S,T]$ of a s-t cut $[S,T]$ as

$$\widetilde{c}[S,T] = c(S,T) - l(T,S). \tag{2}$$

and for the minimum flow problem, we define the *capacity* $\widehat{c}[S,T]$ of a s-t cut $[S,T]$ as

$$\widehat{c}[S,T] = l(S,T) - c(T,S). \tag{3}$$

We refer to an s-t cut whose capacity $\widetilde{c}[S,T]$ is the minimum ($\widehat{c}[S,T]$ is the maximum) among all s-t cuts as a *minimum (maximum) cut*.

The maximum (minimum) flow problem in a network $G = (N, A, l, c, s, t)$ can be solved in two phases:

(P1) establish a feasible flow f, if it exists;

(P2) from a given feasible flow f, establish the maximum flow $\widetilde{f}$ (minimum flow $\widehat{f}$).

Theorem 1: Let $G = (N, A, l, c, s, t)$ be a network, $[S,T]$ a s-t cut and f a feasible flow with value v. Then

$$v = f[S,T] = f(S,T) - f(T,S) \tag{4.a}$$

and therefore, in particular,

$$\widehat{c}[S,T] \le v \le \widetilde{c}[S,T] \tag{4.b}$$

Theorem 2: Let $G = (N, A, l, c, s, t)$ be a network, $[\widetilde{S},\widetilde{T}]$ a minimum s-t cut and $[\widehat{S},\widehat{T}]$ a maximum s-t cut. We denote the values of a maximum flow $\widetilde{f}$ and minimum flow $\widehat{f}$ by $\widetilde{v}$ and $\widehat{v}$, respectively. Then

$$\widetilde{v} = \widetilde{c}[\widetilde{S},\widetilde{T}] \tag{5.a}$$

and

$$\widehat{v} = \widehat{c}[\widehat{S},\widehat{T}] \tag{5.b}$$

A *preflow* f is a function $f : A \to \mathbb{R}_+$ that satisfies (1.b) and

$$f(N,x) - f(x,N) \ge 0, \forall x \in N - \{s,t\} \tag{6.a}$$

for maximum flow problem and

$$f(x,N) - f(N,x) \le 0, \forall x \in N - \{s,t\} \tag{6.b}$$

for minimum flow problem.

For any preflow f we define the *excess*, respectively *deficit* of node x as

$$\widetilde{e}(x) = f(N,x) - f(x,N), \forall x \in N. \tag{7.a}$$

respectively

$$\widehat{e}(x) = f(x,N) - f(N,x), \forall x \in N. \tag{7.b}$$

for maximum and minimum flow problem respectively.

We refer to a node x with $\widetilde{e}(x) = 0$ ($\widehat{e}(x) = 0$) as a *balanced node*. A preflow f satisfying the condition $\widetilde{e}(x) = 0$ ($\widehat{e}(x) = 0$), for each node $x \in N - \{s,t\}$ is a *flow*. Thus, a flow is a particular case of preflow.

For the maximum (minimum) flow problem, the *residual capacity* $\widetilde{r}(x,y)$ ($\widehat{r}(x,y)$) of any arc $(x,y) \in A$, with respect to a given flow/preflow f, is given by $\widetilde{r}(x,y) = c(x,y) - f(x,y) + f(y,x) - l(y,x)$ ($\widehat{r}(x,y) = c(y,x) - f(y,x) + f(x,y) - l(x,y)$). By convention, if $(x,y) \in A$ and $(y,x) \notin A$ then we add the arc (y,x) to the set of arcs A and we set $l(y,x) = 0$ and $c(y,x) = 0$.

The network $\widetilde{G} = (N, \widetilde{A})$ ($\widehat{G} = (N, \widehat{A})$) consisting only of the arcs with positive residual capacity is referred to as the *residual network* (with respect to the flow/preflow f).

In the residual network $\widehat{G} = (N, \widehat{A})$, the *distance function* is a function $\widehat{d} : N \to \mathbb{N}$. We say that a distance function is *valid* if it satisfies the following conditions:

$$\widehat{d}(s) = 0 \tag{8.a}$$

and

$$\widehat{d}(y) \le \widehat{d}(x) + 1, \forall (x,y) \in \widehat{A} \tag{8.b}$$

We refer to $\widehat{d}(x)$ as the *distance label of node* x and to an arc $(x,y) \in \widehat{A}$ as an *admissible* arc if $\widehat{d}(y) = \widehat{d}(x)+1$; otherwise it is *inadmissible*. We refer to a path from node s to node t consisting entirely of admissible arcs as an *admissible path*. We say that the distance labels are *exact* if for each node x, $\widehat{d}(x)$ equals the length of the shortest directed path from node s to node x in the residual network $\widehat{G}$. We refer to a path in G from the source node s to the sink node t as a *decreasing path* if it consists only of arcs with positive residual capacity. Clearly, there is an one to one correspondence between set of decreasing paths in G and the set of directed paths from s to t in $\widehat{G}$.

We define the *layered network* $\widehat{G}' = (N, \widehat{A}', \widehat{r})$ as follows: the nodes set N is partitioned into layers $N_0, \ldots, N_k$, where layer N_i contains the nodes x whose exact distance labels equal i, so that $\widehat{d}(x) = i$. Furthermore, for each arc (x,y) in the layerd network, $x \in N_i$ and $y \in N_{i+1}$ for some i. We say that $\widehat{f}'$ is a *blocking flow* if the layered network $\widehat{G}'$ contains no decreasing directed path.

There are three approaches for solving minimum flow problem:

(1) using decreasing path algorithms;

(2) using preflow algorithms;

(3) using minmax algorithm, see [5].

In this paper we refer to some algorithms in approaches (1) and (2).

The generic algorithm using decreasing path algorithm for the minimum flow problem is as following:

(1)PROGRAM GADP;
(2)BEGIN
(3) let f be a feasible flow in network G;
(4) determine the residual network $\widehat{G}$;
(5) WHILE $\widehat{G}$ contains a directed path $\widehat{D}_{s,t}$ DO
(6) BEGIN
(7) identify a directed path $\widehat{D}_{s,t}$;
(8) $\widehat{r} := min\{\widehat{r}(x,y) \,|\, (x,y) \in \widehat{D}_{s,t}\}$;
(9) identify a path $P_{s,t}$ in G corresponding to $\widehat{D}_{s,t}$;
(10) decrease $\widehat{r}$ units of flow along $P_{s,t}$;
(11) update the residual network $\widehat{G}$;
(12) END;
(13)END.

A node $x \in N - \{s,t\}$ with $\widehat{e}(x) < 0$ is called an *active node*.

The generic preflow algorithm for the minimum flow problem is as follows:

(1)PROGRAM GAP;
(2)BEGIN
(3) PREPROCESS;
(4) WHILE the network contains an active node DO
(5) BEGIN
(6) select an active node y;
(7) PULL/RELABEL(y);
(8) END;
(9)END.

(1)PROCEDURE PREPROCESS;
(2)BEGIN
(3) let f be a feasible flow in network G;
(4) compute the exact distance function $\widehat{d}$
by breadth first searches from s to t
in network $\widehat{G}$;
(5) pull $\widehat{r}(x,t)$ units of flow on each arc
$(x,t) \in \widehat{E}^{-}(t)$;
(6) $\widehat{d}(t) := n$;
(7)END;

(1)PROCEDURE PULL/RELABEL(y);
(2)BEGIN
(3) IF network $\widehat{G}$ contains an admissible arc (x,y)
(4) THEN
pull $\widehat{r}_1 := \min\{-\widehat{e}(y), \widehat{r}(x,y)\}$ units of flow
from node y to node x
(5) ELSE
compute $\widehat{d}(y) :=$
$\min\{\widehat{d}(x) + 1 | (x,y) \in \widehat{E}^{-}(y), \widehat{r}(x,y) > 0\}$;
(6)END;

In this algorithm we have

$$\widehat{E}^{-}(y) = \{(x,y) | (x,y) \in \widehat{A}\}$$

for each node y in N.

A *pull* of $\widehat{r}_1$ units of flow from node y to node x increases both $\widehat{e}(y)$ and $\widehat{r}(y,x)$ by $\widehat{r}_1$ units and decreases both $\widehat{e}(x)$ and $\widehat{r}(x,y)$ by $\widehat{r}_1$ units.

The operation of decreasing the flow on an arc is called *a pull* through the arc. We say that a pull of $\widehat{r}_1$ units of flow on arc (x,y) is a saturating pull if $\widehat{r}_1 = \widehat{r}(x,y)$ and nonsaturating otherwise. A nonsaturating pull at node y reduces $\widehat{e}(y)$ to zero.

We refer to the process of increasing the distance label of a node as a *relabel* operation. The purpose of the relabel operation is to create at least one admissible arc on which the algorithm can perform further pulls.

Using the residual capacities we evaluate the flow with the formula:

$$f(x,y) = l(x,y) + \max\{\widehat{r}(x,y) - c(y,x) + l(y,x), 0\}$$

In Figure 1 we briefly present five algorithms for minimum flow problem.

Algorithm	**Features**
Dinic	1. A special implementation of the GADP. 2. The blocking flows in layered networks $\widehat{G}'$ are determined using decreasing path algorithm.
Karzanov	1. A special implementation of the both GADP and GAP. 2. The blocking flows in layered networks $\widehat{G}'$ are determined using preflow algorithm.
FIFO preflow	1. A special implementation of the GAP. 2. Examines active nodes in the FIFO order.
Highest label preflow	1. A special implementation of the GAP. 2. Examines active nodes with the highest distance label.
Deficit scaling	1. A special implementation of the GAP. 2. Performs pull/relabel operations at the nodes with sufficiently large deficits and, among these nodes, selects a node with the smallest distance label.

Fig. 1. Five algorithms for minimum flow problem

A *bipartite network* is a network $G = (N, A, l, c, s, t)$ with a node set N partitioned into two subsets N_1 and N_2 so that for each arc $(x, y) \in A$, either $x \in N_1$ and $y \in N_2$ or $x \in N_2$ and $y \in N_1$. We often represent a bipartite network using the notation $G = (N_1 \cup N_2, A, l, c, s, t)$.

Let n_1=$|N_1|$ and n_2=$|N_2|$. Without any loss of generality, we assume that $n_2 \leq n_1$. We also assume that $s \in N_2$ and $t \in N_1$. A bipartite network is called *unbalanced* if $n_2 << n_1$ and *balanced* otherwise.

III. Original algorithms for minimum flow in bipartite networks

The time bound for several minimum flow algorithms automatically improves when the algorithms are applied without modification to unbalanced networks.

A careful analyse of the running times of these algorithms reveals that the worst case bound depends on the number of arcs in the longest vertex simple path of the network. We denote this length by p. For general network, $p \leq n - 1$ and for a bipartite network $p \leq 2n_2 + 1$. Hence, for unbalanced bipartite network $p << n$.

We consider Dinic's algorithm for the minimum flow problem. This algorithm constructs $O(p)$ layered networks and finds a blocking flow in each one. Each blocking flow computation performs $O(m)$ decreases and each decrease takes $O(p)$ time. Therefore, the running time of Dinic's algorithm is $O(p^2m)$. Consequently, when the Dinic's algorithm is applied to unbalanced networks, the running time improves from $O(n^2m)$ to $O(n_2^2m)$.

We show that a slightly modified version of the generic preflow algorithm requires loss than $O(n^2m)$ time to solve problems defined on bipartite networks. To establish this result, we change the PROCEDURE PREPROCESS by setting $\widehat{d}(t) = 2n_2 + 1$ instead of $\widehat{d}(t) = n$. The modification stems from the observation that any path in the residual network can have at most $2n_2 + 1$ arcs since every alternate node in the directed path must be in N_2 (because the residual network is also bipartite) and no directed path can repeat a node in N_2. Therefore, if we set $\widehat{d}(t) = 2n_2 + 1$ then the residual network will never contain a directed path from node s to node t and the algorithm will terminate with a minimum flow.

We present the following two theorems (see [1], [2], [5]).

Theorem 3: The generic preflow algorithm maintains valid distance at each step. Moreover, each relabeling of a node x strictly increases $\widehat{d}(x)$.

Theorem 4: At any time during the generic preflow algorithm, for each active node x there is a directed path from node t to node x in the residual network.

We can gain the necessary results for the specific structure of bipartite networks.

Theorem 5: For each active node x, $\widehat{d}(x) < 4n_2 + 1$.

Proof. When a node x is relabeled, it has negative deficit and hence, the residual network has a directed path $P = (x_0, x_1, \ldots, x_k)$ from node $t = x_0$ to node $x = x_k$. Since the nodes on directed path P are alternately in N_1 and N_2 we have $k < 2n_2$. Because $\widehat{d}(t) = 2n_2 + 1$ and $\widehat{d}(x_k) \leq \widehat{d}(x_{k-1}) + 1 \leq \ldots \leq \widehat{d}(x_0) + k$ it follows $\widehat{d}(x) < 4n_2 + 1$. □

Theorem 6:

(a) The number of relabel operations is $O(n_2n)$.

(b) The number of saturating pulls is $O(n_2m)$.

Proof.

(a) From Theorem 5 follows that each distance label increases at most $O(n_2)$ times. Consequently, the number of relabel operations is $O(n_2n)$.

(b) Between two consecutive saturating pulls on an arc (x, y), both $\widehat{d}(x)$ and $\widehat{d}(y)$ must increase by at least 2 units (see [1]). By Theorem 5, only $O(n_2)$ saturating pulls can be on an arc (x, y). Therefore, summing over all m arcs the number of saturating pulls is $O(n_2m)$. □

Theorem 7: The generic preflow algorithm performs $O(n_1^2m)$ nonsaturating pulls.

Proof. Let N_a denote the set of active nodes. We consider the potential function $F = \sum_{N_a} \widehat{d}(x)$. Since we allow only the nodes in N_2 to be active and $\widehat{d}(x) \leq 4n_2$ for all x in N_2, the initial value of F is at most $4n_2^2$. At the end of the algorithm, F is zero.

During the pull/relabel(y) operation, one of the following two cases must apply.

(1) The algorithm is unable to find an admissible arc along which it can pull flow. In this case the distance label of node y increases by $k \geq 1$ units. This operation increases F by at most k units. Since the total increase in $\widehat{d}(y)$ for each node y throughout the execution of the algorithm is bounded by $4n_2$, the total increase in F due to increases in distance labels is bounded by $4n_2^2$.

(2) The algorithm is able to identify an arc on which it can pull flow, so it performs a saturating pull or a nonsaturating pull. A saturating pull on arc (x, y) might create a new deficit at node x, thereby increasing the number of active nodes by 1 and increasing F by $\widehat{d}(x)$, which could be as much as $4n_2$ per saturating pull, and so $4n_2^2m$ over all saturating pulls. Note that a nonsaturating pull on arc (x, y) does not increase $|N_a|$. The nonsaturating pull will decrease F by $\widehat{d}(y)$ since y becomes inactive, but it simultaneously increases F by $\widehat{d}(x) = \widehat{d}(y) - 1$ if the pull causes node x to become active, the total decrease in F being of value 1. If node x was active before the pull, F decreases by an amount

$\widehat{d}(y)$. Consequently, net decrease in F is at least 1 unit per nonsaturating pull.

We summarize these facts. The initial value of F is at most $4n_2^2$ and the maximum possible increase in F is $4n_2^2 + 4n_2^2 m$. Each nonsaturating pull decrease F by at least 1 unit and F always remains nonnegative. Consequently, the algorithm can perform at most $4n_2^2 + 4n_2^2 + 4n_2^2 m = O(n_2^2 m)$ nonsaturating pulls, proving the theorem. □

Figure 2 summarizes the improvements obtained for the next algorithms using this approach, where the value $\bar{c} = \max\{c(x,y)|(x,y) \in A\}$.

Algorithm	**Running time, general network**	**Running time, bipartite network**
Dinic	$n^2 m$	$n_2^2 m$
Karzanov	n^3	$n_2^2 n$
FIFO preflow	n^3	$n_2^2 n$
Highest label preflow	$n^2 m^{1/2}$	$n_2 n m^{1/2}$
Deficit scaling	$nm + n^2 \log \bar{c}$	$n_2 m + n_2 n \log \bar{c}$

Fig. 2. The running time for five algorithms

IV. Specialization algorithms for minimum flow in bipartite networks

All minimum flow algorithms described in this section are preflow algorithms, i.e. algorithms that maintain a preflow at every stage. They work by examining active nodes and pulling deficit from these nodes to nodes estimated to be closer to source node s. If the source node s is not reachable, however, an attempt is made to pull the deficit back to the sink node t. Eventually, there will be no deficit on any node $x \in N - \{s,t\}$. At this point the preflow is a flow and, moreover, it is a minimum flow [5]. The algorithms use exact distance labels to measure the closeness of a node to the source node s or the sink node t.

Specialization algorithm for minimum flow in bipartite network is called *bipartite preflow algorithm*. The basic idea behind the bipartite preflow algorithm is to perform bipull from nodes N_2. A *bipull* is a pull over two consecutive admissible arcs. It moves deficit from a node in N_2 to another node in N_2. This approach ensures that no node in N_1 ever has any deficit.

The bipartite preflow algorithm is a simple generalization of the generic preflow algorithm given in Section II. We change the preprocess operation by setting $\widehat{d}(t) := 2n_2 + 1$ instead of $\widehat{d}(t) := n$. The modification stems from the observation that any directed path in the residual network can have at most $2n_2$ arcs since every alternate node in the directed path must be in N_2 (because the residual network is also bipartite) and no directed path can repeat a node in N_2. Therefore, if we set $\widehat{d}(t) := 2n_2 + 1$, the residual network will never contain a directed path from the source node s to the sink node t and the algorithm will terminate with a minimum flow. Consequently, the PROCEDURE PULL/RELABEL is replaced with the PROCEDURE BIPULL/RELABEL, as follows.

(1)PROCEDURE BIPULL/RELABEL(y);
(2)BEGIN
(3) IF network $\widehat{G}$ contains an admissible arc (x,y)
(4) THEN
  IF network $\widehat{G}$ contains an admissible arc (u,x)
(5) THEN pull $\widehat{r}_1 := \min\{-\widehat{e}(y), \widehat{r}(x,y), \widehat{r}(u,x)\}$
  units of flow over back path (y,x,u)
(6) ELSE
  compute $\widehat{d}(x) :=$
  $\min\{\widehat{d}(u) + 1|(u,x) \in \widehat{E}^-(x), \widehat{r}(u,x) > 0\}$
(7) ELSE
  compute $\widehat{d}(y) :=$
  $\min\{\widehat{d}(x) + 1|(x,y) \in \widehat{E}^-(y), \widehat{r}(x,y) > 0\}$;
(8)END;

We call a pull of $\widehat{r}_1$ units on the back path (y,x,u) a *bipull*. The bipull is *saturated* if $\widehat{r}_1 = \widehat{r}(x,y)$ or $\widehat{r}_1 = \widehat{r}(u,x)$ and *unsaturated* otherwise. Note that an unsaturated bipull reduces the deficit at vertex y to zero.

As in the generic preflow algorithm, the bipartite preflow algorithm always pulls flow on admissible arcs and relabels a node only if there are no admissible arcs. Hence, Theorem 3 holds for this algorithm too. Theorem 4, Theorem 5, Theorem 6 also hold.

Theorem 8: During the execution of the bipartite preflow algorithm all active nodes remain in N_2.

Proof. First of all, the algorithm has to saturate all arcs $(x,t) \in \widehat{E}^-(t)$. Since the node $t \in N_1$, the claim is true immediately after this step. All the other pulls in the algorithm are done using the PROCEDURE BIPULL/RELABEL, which pulls from a node in N_2 through a node in N_1 to another node in N_2, never leaving any deficit on a node in N_1. No other operations create deficit at any node. □

Theorem 9: The bipartite preflow algorithm runs in $O(n_2^2 m)$ time.

Proof. Let N_a denote the set of active nodes. We consider the potential function $F = \sum_{N_a} \widehat{d}(x)$. Since we allow only the nodes in N_2 to be active and $\widehat{d}(x) \leq 4n_2$ for all x in N_2, the initial value of F is at most $4n_2^2$. The procedure bipull/relabel(y) yields one of the following four cases:

(1) it increases the distance label of node x;

(2) it increases the distance label of node y in N_2;

(3) it pulls flow over the arcs (x, y) and (u, x), saturating one of these two arcs;

(4) it performs a nonsaturating pull.

In case (1) the potential function F increases, but the total increase over all such iterations is only $O(n_2^2)$.

In case (2) the function F remains unchanged.

In case (3) the function F can increase by as much as $4n_2 + 1$ units since a new node might become active. Theorem 6 shows that the total increase over all iterations is $O(n_2^2 m)$.

In case (4) a nonsaturating pull decreases the potential function F by at least 2 units since it makes node y inactive and it can make node u newly active node with $\widehat{d}(u) = \widehat{d}(y) - 2$.

This fact, in view of the preceding arguments, implies that the algorithm performs $O(n_2^2 m)$ nonsaturating pulls. Since all the other operations, such as the relabel operations and finding admissible arcs require only $O(n_2 m)$ time, we have established the theorem. □

We also consider the ideas of the bipartite preflow algorithm beeing applied in a straightforward manner to the Karzanov, FIFO preflow, highest label preflow and deficit scaling algorithms. It yields that the algorithm improved worst case complexity.

The FIFO preflow algorithm examines active nodes in first-in, first-out (FIFO) order. The algorithm maintains a queue Q of nodes. It selects a node y from the front of Q, performs pulls from this node and adds newly active nodes to the rear of Q. The algorithm examines node y until either it becomes inactive or it is relabeled. In the latter case we add node y to the rear of Q. The algorithm terminates when the queue Q of active nodes is empty.

To analyze the complexity of bipartite FIFO preflow algorithm, we partition the total number of node examinations into different phases. The first phase consists of node examinations for those nodes that become active during the preprocess operation. For $k \geq 2$, the k^{th} phase consists of examining all nodes that have been added to Q during the $(k-1)^{th}$ phase.

Theorem 10: The number of phases over Q is $O(n_2^2)$.

Proof. We consider the total change in the potential function $F = \max\{\widehat{d}(x) | x \text{ is active node}\}$ over an entire phase. The initial value of F is at most $4n_2$. We consider two cases:

(1) The algorithm performs no relabel operation during a phase. In this case the deficit of every node in N_2 that was active at the beginning of the phase moves to nodes with smaller distance labels. Consequently, F decreases by at least 2 units.

(2) The algorithm performs at least one relabel operation during a phase. In this case the potential function F can increase or remain the same. In such a case the increase in F, then F might increase by as much as the maximum increase in any distance label. Hence, by Theorem 5, the total increase in F over all the phases is at most $4n_2^2$.

Combining cases (1) and (2), we find that the total number of phases is $O(n_2^2)$. □

Theorem 11: The bipartite FIFO preflow algorithm runs in $O(n_2 m + n_2^2)$ time.

Proof. This theorem is a direct consequence of Theorem 10. □

We note that bound is also achieved by Karzanov's algorithm if it is implemented using the two-arcs pull rule.

The highest label preflow algorithm always pulls from an active node with highest distance label. The nonsaturating bipulls performed by the algorithm can be divided into phases. A phase consists of all bipulls that occur between two consecutive relabel steps of nodes in N_2. Within a phase, nodes in N_1 can possibly be relabeled several times.

We remark that, in this algorithm, the deficits that are most distant from the source are pulled down two levels at a time. Consequently, if the algorithm does not relabel any node during n_2 consecutive node examinations, the total deficit reaches the source and the algorithm terminates. Since the algorithm performs $O(n_2^2)$ relabel operations on nodes in N_2, we immediately obtain a bound of $O(n_2^3)$ on the number of node examinations. As each node examination entails at most one nonsaturating bipull, this gives a bound of $O(n_2^3)$ on the number of nonsaturating bipulls and a bound of $O(n_1 m + n_1^3)$ on the running time of the algorithm.

The bound of $O(n_2^3)$ on the number of nonsaturating bipulls performed by the algorithm is rather loose and can be improved by a more clever analysis.

Theorem 12: The bipartite highest label preflow algorithm performs $O(n_2 m)$ nonsaturating bipulls and runs in the same time.

Proof. The proof is long, complex and we omitted it here. □

The deficit scaling preflow algorithm incorporates scaling of the deficit into the generic preflow algorithm.

Theorem 13: The deficit scaling preflow algorithm performes $O(n_2^2 \log \bar{c})$ nonsaturating bipulls and runs

$O(n_2 m + n_2^2 \log \bar{c})$ time.

Proof. The proof is long, complex and we omitted it here. □

Figure 3 summarizes the improvements obtained using this approach.

Algorithm	**Running time, general network**	**Running time, modified version**
Karzanov	n^3	$n_2 m + n_2^3$
FIFO preflow	n^3	$n_2 m + n_2^3$
Highest label preflow	$n^2 m^{1/2}$	$n_2 m$
Deficit scaling	$nm + n^2 \log \bar{c}$	$n_2 m + n_2^2 \log \bar{c}$

Fig. 3. The running time for four algorithms

V. EXAMPLE

We represent in Figure 4 the initial bipartite network $G' = (N_1' \cup N_2', A', l', c')$ with $N_1' = \{2, 3, 4\}$ and $N_2' = \{5, 6\}$.

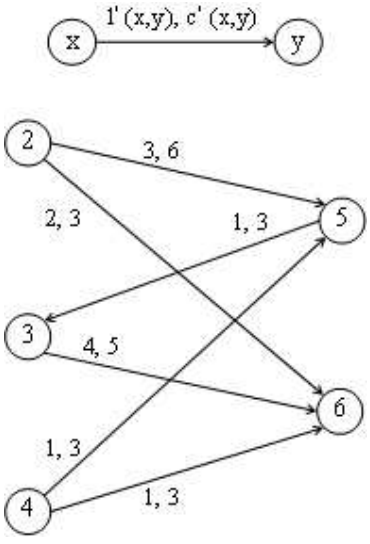

Fig. 4. The initial bipartite network

The extended network is $G = (N_1 \cup N_2, A, l, c, s, t)$ with the source node $s = 1$, the sink node $t = 7$, $N_1 = \{2, 3, 4, 7\}$ and $N_2 = \{1, 5, 6\}$. This network is represented in Figure 5.

The network flow is in Figure 6 and the value of the admissible flow is $v = 15$.

The residual network $\widehat{G}(f)$ is in Figure 7.

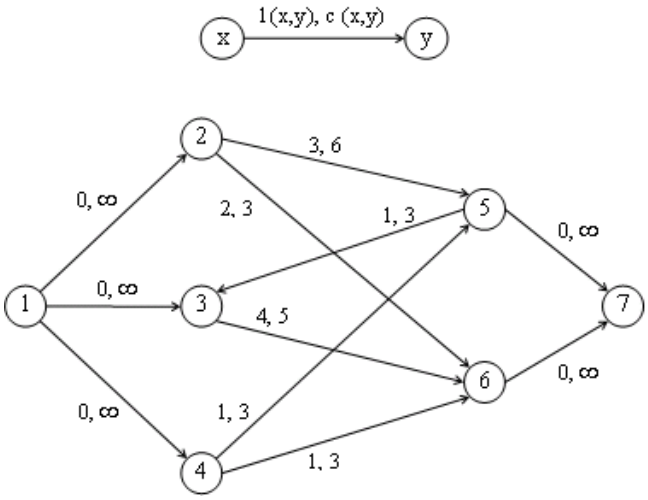

Fig. 5. The extended network

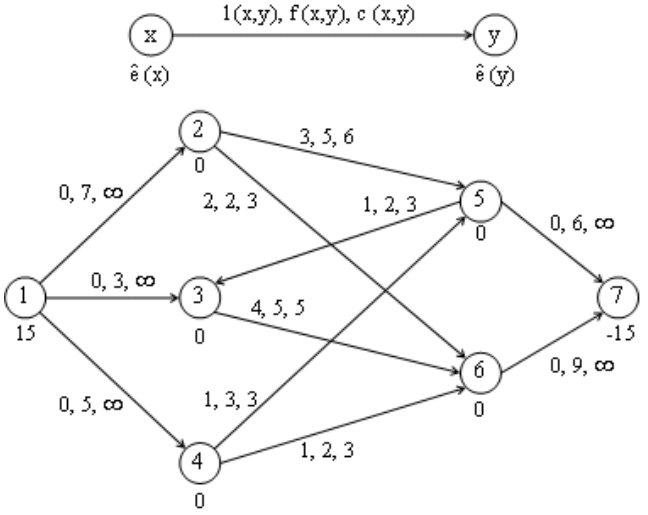

Fig. 6. The network flow

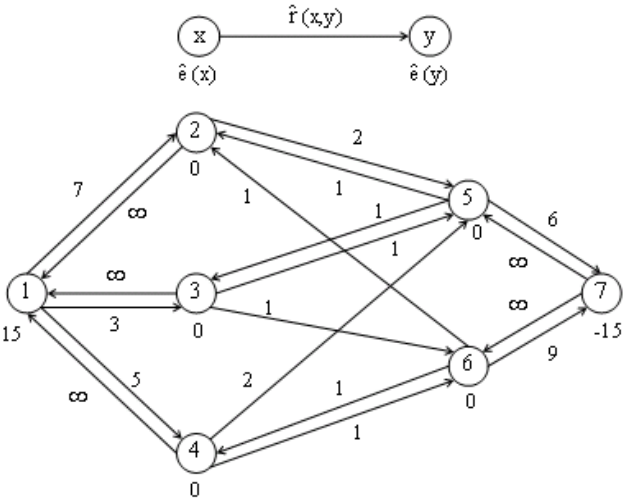

Fig. 7. The initial residual network

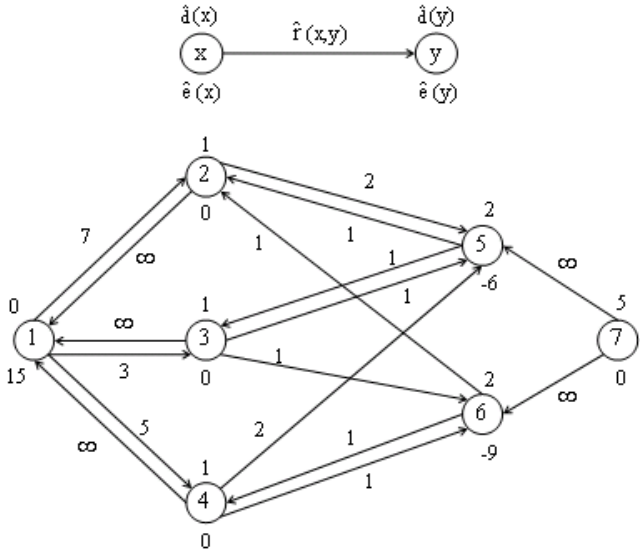

Fig. 8. The residual network after the PREPROCESS

By applying the PREPROCESS procedure we obtain the network represented in Figure 8.

We apply the BIPULL/RELABEL procedure. We select $y = 5$, $(x, y) = (2, 5)$, $(u, x) = (1, 2)$, $\widehat{r}_1 = \min\{6, 2, 7\} = 2$ and $(y, x, u) = (5, 2, 1)$. We obtain the residual network represented in Figure 9.

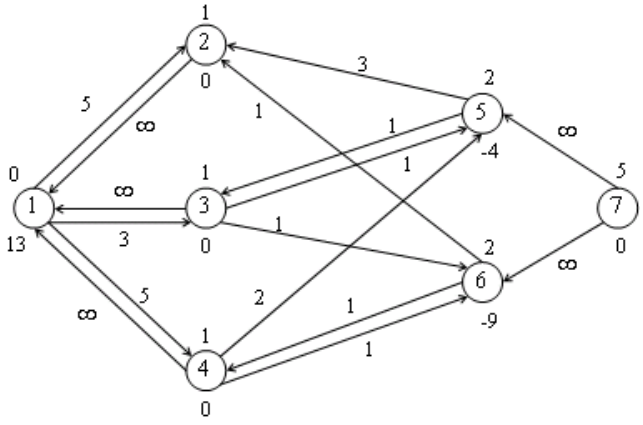

Fig. 9. The residual network

For the next step we have $y = 5$, $(x, y) = (3, 5)$, $(u, x) = (1, 3)$, $\widehat{r}_1 = \min\{4, 1, 3\} = 1$ and $(y, x, u) = (5, 3, 1)$. Now, the residual network is represented in Figure 10.

For the next step we have $y = 5$, $(x, y) = (4, 5)$, $(u, x) = (1, 4)$, $\widehat{r}_1 = \min\{3, 2, 5\} = 2$ and $(y, x, u) = (5, 4, 1)$. Now, the residual network is represented in Figure 11.

It follows that $y = 5$ and there is no admissible arc $(x, 5)$. Hence, $\widehat{d}(5) = \min\{5 + 1\} = 6$. We select $y = 6$, $(x, y) = (3, 6)$, $(u, x) = (1, 3)$, $\widehat{r}_1 = \min\{9, 1, 2\} = 1$

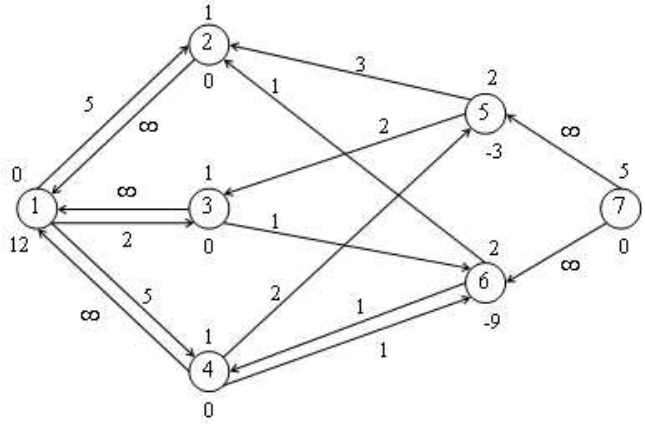

Fig. 10. The residual network

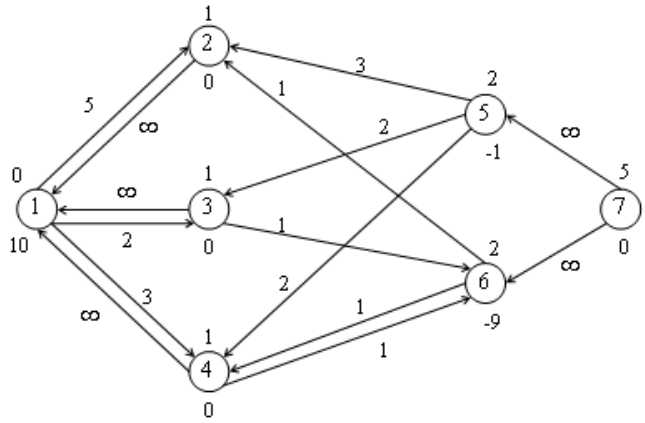

Fig. 11. The residual network

and $(y, x, u) = (6, 3, 1)$. Now, the residual network is represented in Figure 12.

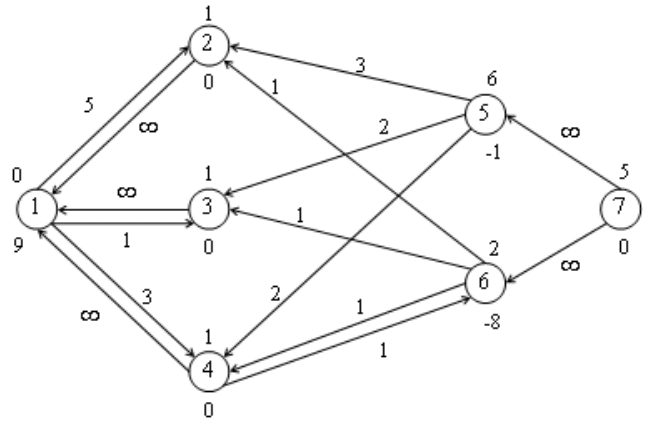

Fig. 12. The residual network

For the next step we have $y = 6$, $(x, y) = (4, 5)$, $(u, x) = (1, 4)$, $\widehat{r}_1 = \min\{8, 1, 3\} = 1$ and $(y, x, u) = (6, 4, 1)$.

Now, the residual network is represented in Figure 13.

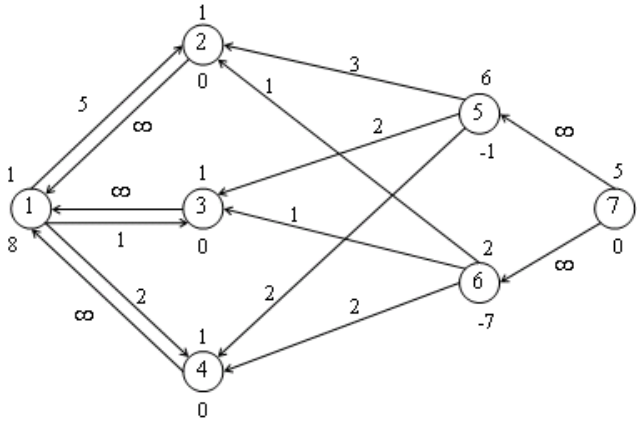

Fig. 13. The residual network

It follows that $y = 6$ and there is no admissible arc $(x, 6)$. Hence, $\widehat{d}(6) = \min\{5 + 1\} = 6$.

Obviously, the source node $s = 1$ cannot be reached from the active nodes $y = 5$, $y = 6$ using reverse paths from y to s. In this case the deficite of these nodes will be sent back to the node $y = 7$. The residual network is now represented in Figure 14.

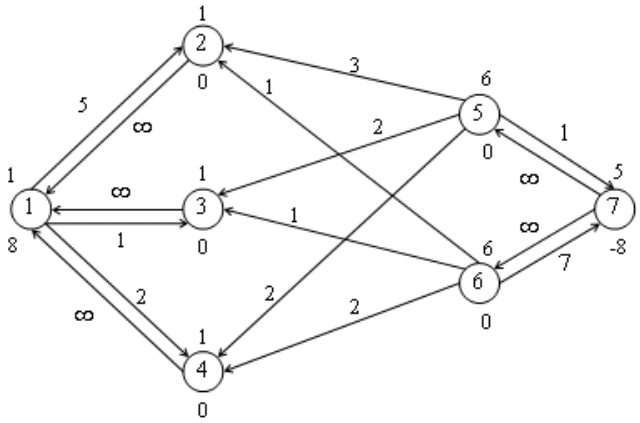

Fig. 14. The residual network

There are no anymore active nodes and the algorithm stops. It pulls 7 units of flow from the sink node $t = 7$ to the source node $s = 1$. The minimum flow in the extended network is in Figure 15.

The value of minimum flow is $\widehat{v} = 8$ and the maximum cut is $[\widehat{S}, \widehat{T}] = (\widehat{S}, \widehat{T}) \cup (\widehat{T}, \widehat{S}) = \{(2,5),(2,6),(3,6),(4,5),(4,6)\} \cup \{(5,3)\}$ with $l(\widehat{S}, \widehat{T}) - c(\widehat{T}, \widehat{S}) = 3 + 2 + 4 + 1 + 1 - 3 = 8$. Obviously, $\widehat{v} = l(\widehat{S}, \widehat{T}) - c(\widehat{T}, \widehat{S})$.

The minimum flow in the initial bipartite network from N_1' to N_2' is represented in Figure 16.

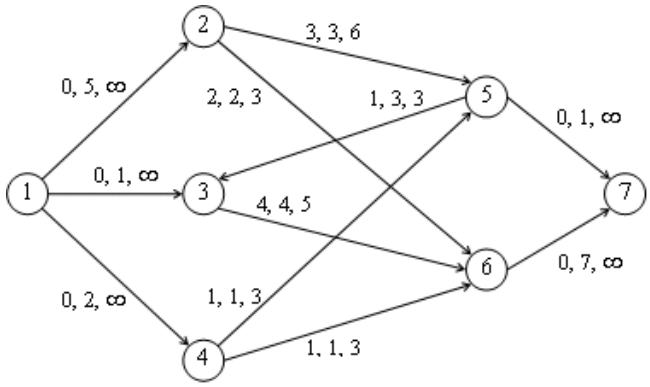

Fig. 15. The minimum flow network

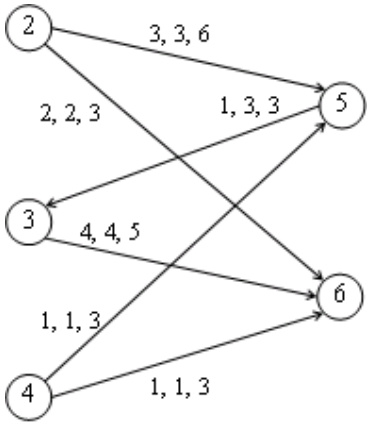

Fig. 16. The final network flow

ACKNOWLEDGMENT

The research was supported by the Transilvania University of Braşov and, in the case of the first and the third authors, it was also supported by the Grant IDEI no. 134/2007.

REFERENCES

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

[2] R. Ahuja, J. Orlin, C. Stein and R. Tarjan, Improved algorithms for bipartite network flows, *SIAM Journal of Computing*, Vol. 23, 1994, pp. 906–933.

[3] L. Ciupală and E. Ciurea, A highest-label preflow algorithm for the minimum flow problem, *Proceedings of the* 11^{th} *WSEAS International Conference on Computers*, Crete, Greece, 2007, pp. 167–170.

[4] L. Ciupală and E. Ciurea, About preflow algorithms for the minimum flow problem, *WSEAS Transactions on Computer Research*, Vol. 3(1), January 2008, pp. 35–42.

[5] E. Ciurea and L. Ciupală, Sequential and parallel algorithms for minimum flows, *Journal of Applied Mathematics and Computing*, Vol. 15, No. 1-2, 2004, pp. 53–75.

[6] E. Ciurea and O. Georgescu, Minimum flows in unit capacity networks, *Analele Universităţii Bucureşti*, XLV, 2006, pp. 11–20.

[7] E. Ciurea, O. Georgescu and M. Iolu, Minimum Flow Algorithms. Dynamic Tree Implementations, *Studia Univ. Babeş Bolyai*, *Informatica*, LIII(1), 2008, pp. 73–82.

[8] E. Dinic, Algorithm for solution of a problem of maximum flow in network with power estimation, *Soviet Mathematics Doklady*, Vol. 11, 1970, pp. 1277–1280.

[9] O. Georgescu and E. Ciurea, Decreasing Path Algorithm for minimum flows. Dynamic Tree Implementations, *Proceedings of the* 12^{th} *WSEAS International Conference on Computers*, Crete, Greece, 2008, pp. 235–240.

[10] O. Georgescu, Minimum Flow in Network using Dynamic Tree Implementation, *Proceedings of* 3^{rd} *Annual South-East European Doctoral Student Conference*, Thessaloniki, Greece, Vol. 2, 2008, pp. 192–204.

[11] D. Gusfield, C. Martel and D. Fernandez-Baca, Fast algorithms for bipartite network flow, *SIAM Journal of Computing*, Vol. 16, 1987, pp. 237–251.

[12] A. Karzanov, Determining the maximal flow in a network by the method of preflows, *Soviet Mathematics Doklady*, Vol. 15, 1974, pp. 434–437.

[13] S.–S. Lin, H. Chang and C.–C. Kuo, An Implementation of the Parallel Algorithm for Solving Nonlinear Multi-commodity Network Flow Problem, *WSEAS Transactions on Systems*, Vol. 5(8), August 2006, pp. 1853–1860.

[14] E. Milkova, Combinatorial Optimization: Mutual Relations among Graph Algorithms, *WSEAS Transactions on Mathematics*, Vol. 7(5), May 2008, pp. 293–302.