Improved Algorithm for Minimum Flows in Bipartite Networks with Unit Capacities

ELEONOR CIUREA
Transilvania University of Braşov
Theoretical Computer Science Departement
Braşov, Iuliu Maniu 50, cod 500091
ROMANIA
e.ciurea@unitbv.ro

ADRIAN DEACONU
Transilvania University of Braşov
Theoretical Computer Science Departement
Braşov, Iuliu Maniu 50, cod 500091
ROMANIA
a.deaconu@unitbv.ro

Abstract: The theory and applications of network flows is probabily the most important single tool for applications of digraphs and perhaps even of graphs as a whole. In this paper we study minimum flow an algorithm in bipartite networks with unit capacities combining an algorithm for minimum flow in bipartite networks with an algorithm for minimum flow in unit capacity networks. Finally, we present the applications of the minimum flow problem in bipartite networks with unit capacities.

Key–Words: Network flows, minimum flow problem, unit capacity networks, bipartite networks, maximum cut.

1 Introduction

From a theoretical point of view, flows are well understood as far as the basic questions, such as finding a maximum flow from a given source to a given sink or characterizing the size of such a flow, are concerned. However, the topic is still a very active research field and these are challenging open problems.

The computation of a flow in a network 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 the maximum flow problem, see [1], [12], [13], [16], [17], [18], [19]. The computation of a minimum flow in a network has been investigated by Ciurea and Ciupala, see, e.g., [4], [5]. By improving the running times of Ciurea and Ciupala the algorithms [4][5], Ciurea, Ciupala and Georgescu have developed the first specializations of minimum flow algorithms for bipartite netwrks [2], [3], [6] and [7]. Ciurea and Georgescu [8] showed that Ciurea and Ciupala algorithms solve the minimum flow problem in unit capacity networks in $O(min\{n^{2/3}m, m^{3/2}\})$, where n is the number of nodes and m is the number of arcs of the network. Other papers on the minimum flow problem are [9], [10], [14], [15].

Combining an algorithm for minimum flow in bipartite networks with an algorithm for minimum flow in unit capacity networks we obtain an efficient algorithm for minimum flow in bipartite networks with unit capacities.

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 minimum flows in bipartite networks. In Section 4 we present the minimum flows in unit capacity networks. In section 5 we present the applications of the minimum flows in bipartite networks with unit capacities.

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 [4] for minimum flow problem.

2 Terminology and Preliminaries

In this section we discuss some basic notations and results used in the rest of the 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 $l(x, y)$ 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.

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:

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

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

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

The minimum flow problem is to determine a flow f for which v is 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. We denote also:

$$[S,T] = (S,T) \cup (T,S)$$

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

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

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

The 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 minimum flow $\widehat{f}$.

The problem of determining a feasible flow consists in finding a function $f : A \to \mathrm{R}_+$ that satisfies the previous conditions (1.a) and (1.b). First, we transform this problem into a circulation problem by adding an arc (t,s) of infinite capacity and zero lower bound. This arc carries the flow sent from the source node s to the sink node t back to the source node s. Clearly, the minimum flow problem admits a feasible flow if and only if the circulation problem admits a feasible flow. Because these two problems are equivalent, we focus on finding a feasible circulation if it exists in the transformed network $G_1 = (N_1, A_1, l_1, c_1, s, t)$, where

$$N_1 = N$$
$$A = A \cup \{(t,s)\},$$
$$l_1(x,y) = l(x,y), \text{ for each arc}(x,y) \in A,$$
$$l_1(t,s) = 0,$$
$$c_1(x,y) = c(x,y), \text{ for each arc}(x,y) \in A,$$
$$c_1(t,s) = \infty.$$

The feasible circulation problem is to identify a flow f_1 satisfying the following constraints:

$$f_1(x,N_1) - f_1(N_1,x) = 0, \text{ for each node } x \in N,$$

$$l_1(x,y) \leq f_1(x,y) \leq c_1(x,y), \forall (x,y) \in A_1.$$

By replacing

$$f_1(x,y) = f_2(x,y) + l_1(x,y)$$

and

$$c_1(x,y) = c_2(x,y) + l_1(x,y)$$

we obtain the following transformed problem:

$$f_2(x,N_1) - f_2(N_1,x) = b_2(x), \text{ for each node } x \in N,$$

$$0 \leq f_2(x,y) \leq c_2(x,y), \text{ for each arc } (x,y) \in A_1,$$

where

$$b_2(x) = l_1(N,x) - l_1(x,N), \text{ for each node } x \in N.$$

Clearly,

$$\sum_N b_2(x) = 0.$$

We can solve this supply and demand problem by solving a maximum flow problem defined in the network $G_2 = (N_2, A_2, c_2, s_2, t_2)$, where

$$N_2 = N_1 \cup \{s_2, t_2\},$$
$$A_2 = \tilde{A}_2' \cup \tilde{A}_2'' \cup \tilde{A}_2''',$$
$$A_2' = \{(s_2,x) | x \in N_1, b_2(x) > 0\},$$
$$A_2'' = A_1,$$
$$A_2''' = \{(x,t_2) | x \in N_1, b_2(x) < 0\},$$
$$c_2(s_2,x) = b_2(x), (s_2,x) \in A_2',$$
$$c_2(x,y) = c_1(x,y), (x,y) \in A_2'',$$

$$c_2(x,t_2) = -b_2(x), (x,t_2) \in A_2'''.$$

If the maximum flow f_2 in this transformed network G_2 saturates all the source and the sink arcs ($f_2(s_2,x) = c_2(s_2,x), \forall (s_2,x) \in A_2'$ and $f_2(x,t_2) = c_2(x,t_2), \forall (x,t_2) \in A_2'''$), then the initial problem has a feasible solution f, which is the restriction function of $f_1 = f_2 + l_1$ for the set of arcs A.

We present the following two theorems (see [2] and [3]):

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

$$v = f[S,T] = f(S,T) - f(T,S) \qquad (3.a)$$

and therefore, in particular,

$$\widehat{c}[S,T] \leq v \qquad (3.b)$$

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

$$\widehat{v} = \widehat{c}[\widehat{S}, \widehat{T}] \qquad (4)$$

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

$$f(x,N) - f(N,x) \leq 0, \forall x \in N - \{s,t\} \qquad (5)$$

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

$$\widehat{e}(x) = f(x,N) - f(N,x), \forall x \in N. \qquad (6)$$

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

For the minimum flow problem, the *residual capacity* $\widehat{r}(x,y)$ of any arc $(x,y) \in A$, with respect to a given flow/preflow f, is given by

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

By convention, if $(x,y) \in A$ and $(y,x) \notin A$ then we add arc (y,x) to the set of arcs A and we set $l(y,x) = 0$ and $c(y,x) = 0$. The network $\widehat{G} = (N, \widehat{A})$ consisting only of the arcs with positive residual capacity is referred to as the *residual network*.

The arcs of the residual network $\hat{G}$ have a natural interpretation. If $(x,y) \in A$ and $l(x,y) = 2$, $f(x,y) = 5$ and $c(x,y) = 7$, then we may decrise $f(x,y)$ by up to two units on the arc (x,y) and we can also choose to increase $f(x,y)$ by up to 3 units along the arc (x,y). Note that an increase of flow along the arc (x,y) may also be thought of as sending flow in the opposite direction along the residual arc (y,x) and then canceling out.

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:

$$\widehat{d}(s) = 0 \qquad (7.a)$$

and

$$\widehat{d}(y) \leq \widehat{d}(x) + 1, \forall (x,y) \in \widehat{A} \qquad (7.b)$$

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*. A directed path from node s to node t in the residual network $\widehat{G}$ consisting entirely of admissible arcs is said to be an admissible directed path. 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 $\widehat{d}(x) = i, \forall i \in \{0, 1, ..., k\}$. 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 the minimum flow problem:

(1) using decreasing directed path algorithms from source node s to sink node t in residual network;

(2) using preflow-pull algorithms from sink node t to source node s in the residual network;

(3) using the augmentation directed path algorithms from sink node t to source node s or using preflow-push algorithms starting from t in the residual network $\tilde{G}$ (residual network for maximum flow).

We present the generic decreasing directed path algorithm. This algorithm is based on decreasing path theorem (see [4]). The algorithm begins with a feasible flow and proceeds by identifying decreasing directed paths and decreasing flows on these paths until the residual network $\hat{G}$ contains no such directed path. The generic decreasing directed path algorithm is the following:

(1)PROGRAM GDDP
(2)BEGIN
(3) Let f be a feasible flow in the network G;
(4) Determine the resiudal network $\hat{G}$;
(5) WHILE $\hat{G}$ contains a directed path $\hat{P}$ from the source node s to the sink node t DO
(6) BEGIN
(7) Identify a decreasing path $\hat{P}$ from the source node s to the sink node t;
(8) $\hat{r}(\hat{P}) = min\{r(x,y)|(x,y) \in \hat{P}\}$;
(9) Decrease $\hat{r}(\hat{P})$ units of the flow along $\hat{P}$;
(10) Update the residual network $\hat{G}$;
(11) END;
(12)END.

All the algorithms from the figure 1 are decreasing path algorithms, i.e., algorithms which determine decreasing directed path from source node s to sink node t (by different rules) in the residual network and then decreasing the flow along the coresponding path in G (the approache (1)). We have $n = |N|$, $m = |A|$ and $\bar{c} = max\{c(x,y)|(x,y) \in A\}$.

	Decreasing directed path algorithms	**Running time**
1	Generic decreasing path algorithm	O($nm\bar{c}$)
2	Ford-Fulkerson labeling algorithm	O($nm\bar{c}$)
3	Gabow bit scaling algorithm	O($nm \cdot log\bar{c}$)
4	Ahuja-Orlin maximum scaling algorithm	O($nm \cdot log\bar{c}$)
5	Edmonds-Karp shortest path algorithm	O(nm^2)
6	Ahuja-Orlin shortest path algorithm	O(n^2m)
7	Dinic layered networks algorithm	O(n^2m
8	Ahuja-Orlin layered networks algorithm	O(n^2m)

Figure 1: The running time of eight algorithms

Now we shall present the generic preflow-pull algorithm. This algorithm begins with a feasible flow and sends back as much flow as it is possible from the sink node t to the source node s. Because the algorithm decreases the flow on individual arcs, it does not satisfy the mass balance constraint (1.a) at intermediate stages. In fact, it is possible that the flow entering in a node exceeds the flow leaving from it. The basic operation of this algorithm is to select an active node and to send the flow entering in it back, closer to the source node s. For measuring closerness, we will use the distance labels $\hat{d}(x)$. Let y be a node with strictly negative deficit. If there exists an admissible arc (x,y), we pull the flow on this arc, otherwise we relabel the node y so that we create at least one admissible arc. The generic preflow-pull algorithm is the following:

(1)PROGRAM GPP
(2)BEGIN
(3)PREPROCESS;
(4)WHILE the residual network $\hat{G}$ 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 the network G;
(4)Determine the residual network $\hat{G}$;
(5)Compute the exact distance function $\hat{d}$ by
(6) breadth first search from s to t in $\hat{G}$;
(7)Pull $\hat{r}(x,t)$ units of flow on arc $(x,t) \in \hat{E}^-(t)$;
(8)$\hat{d}(t) := n$;
(9)END;

(1)PROCEDURE PULL/RELABEL(y);
(2)BEGIN
(3)IF the network $\hat{G}$ contains an admisible arc (x,y)
(4) THEN
(5) Pull $\hat{r}_1 = min\{-\hat{e}(y), \hat{r}(x,y)\}$; units from y to x;
(6) ELSE
(7) $\hat{d}(y) := min\{\hat{d}(x) + 1|(x,y) \in \hat{E}^-(y)\}$;
(8)END;

A pull of $\hat{r}_1$ units from node y to node x decreases both $\hat{e}(x)$ and $\hat{r}(x,y)$ by $\hat{r_1}$ units of flow and increases both $\hat{e}(y)$ and $\hat{r_1}(y,x)$ by $\hat{r}$ units of flow. We refer to the process of increasing the distance label of node y, $\hat{d}(y) = min\{\hat{d}(x) + 1|(x,y) \in \hat{E}^-(y)\}$ as a relabel operation. In this algorithm we have

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

for each node $y \in N$.

All the algorithms from the figure 2 are preflow-pull algorithms from sink node t to source node s in the residual network $\hat{G}$ (the approach (2)).

	Preflow-pull algorithms	**Running time**
1	Generic preflow-pull algorithm	O(n^2m)
2	Karzanov preflow-pull algorithm	O(n^3)
3	FIFO preflow-pull algorithm	O(n^3)
4	Higherst label preflow-pull algorithm	O($n^2m^{1/2}$)
5	Deficit scaling preflow-pull algorithm	O($nm + n^2log(\bar{c})$)

Figure 2: The running time of five algorithms

Actually, any algorithm terminates with optimal residual capacities. From these residual capacities we can determine a minimum flow by following expression:

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

3 Minimum flows in bipartite networks

In this section we shall present some minimum flow algorithms for bipartite networks.

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. We assume that the bipartite network is unbalanced. There are many applications of unbalanced networks.

The computation of a minimum flow in a bipartite network has been investigated in [2], [3], [6] and [7].

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 $p << n$.

For example, we consider Dinic's algorithm for the minimum flow problem. This algorithm constructs the layered network in $O(p)$ time and finds a blocking flow each time. 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 Dinic's algorithm is applied to unbalanced networks the running time improves from $O(n^2m)$ to $O(n_2^2m)$.

Figure 3 summarizes the improvements obtained for several algorithms using this approach (see [6], [7]).

	Original algorithms	**Running time, general network**	**Running time, bipartite network**
1	Dinic	$O(n^2m)$	$O(n_2^2m)$
2	Karzanov	$O(n^3)$	$O(n_2^2n)$
3	FIFO preflow	$O(n^3)$	$O(n_2^2n)$
4	Highest label preflow	$O(n^2m^{1/2})$	$O(n_2nm^{1/2})$
5	Deficit scaling	$O(nm+n^2 \log \bar{c})$	$O(n_2m+n_2n \log \bar{c})$

Figure 3: The running time for five algorithms

Specialization algorithm for minimum flow in bipartite network is a preflow algorithm and it 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 (x, y) and (u, x), where $x \in N_1$ and $y, u \in N_2$. It moves deficit from a node $y \in N_2$ to another node $u \in N_2$. 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. 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.

In the procedure PREPROCESS from the program GPP we replace $\widehat{d}(t) := n$ with $\widehat{d}(t) := 2n_2 + 1$. 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
$\widehat{d}(x) := \min\{\widehat{d}(u) + 1 | (u, x) \in \widehat{E}^{-}(x)\}$
(7) ELSE
$\widehat{d}(y) := \min\{\widehat{d}(x) + 1 | (x, y) \in \widehat{E}^{-}(y)\}$;
(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.

The idea of bipartite preflow algorithm we consider also apply in a straight forward manner to the Karzanov, FIFO preflow, highest label preflow and deficit scaling algorithms and yield algorithm improved worst case complexity.

Figure 4 summarizes the improvements obtained using this approach.

	Specialization algorithms	**Running time, general network**	**Running time, bipartite network**
1	Generic preflow	$O(n^2m)$	$O(n_2^2m)$
2	Karzanov	$O(n^3)$	$O(n_2m + n_2^3)$
3	FIFO preflow	$O(n^3)$	$O(n_2m + n_2^3)$
4	Highest label preflow	$O(n^2m^{1/2})$	$O(n_2m)$
5	Deficit scaling	$O(nm+n^2 \log \bar{c})$	$O(n_2m+n_2^2 \log \bar{c})$

Figure 4: The running time for five algorithms

The reader interested in further details is urged to consult the papers [6], [7].

4 Minimum flows in unit capacity networks

We present two algorithms for finding the minimum flows in unit capacity networks.

An *unit capacity network* is a network $G = (N, A, l, c, s, t)$ so that $l(x, y) = 0$ or 1 and $c(x, y) = 1$ for all arcs $(x, y) \in A$. The computation of a minimum flow in a bipartite network bas been investigated in [8].

In a unit capacity network, the Ford-Fulkerson labeling algorithm determines a minimum flow within n decreasion steps that requires $O(nm)$ time, because $\overline{c} = 1$. The Ahuja-Orlin shortest path algorithm also solves this problem in $O(nm)$ time since its bottleneck operation, which is a decreasion step, requires $O(nm)$ time.

The first unit capacity minimum flow algorithm is a two-phase algorithm. Let

$$\widehat{d}^* = min\{\left\lceil 2n^{2/3} \right\rceil, \left\lceil m^{1/2} \right\rceil\}$$

(P1) The algorithm applies the Ahuja-Orlin shortest path algorithm; this phase terminates whenever the distance label of the node t satisfies the condition $\widehat{d}(t) \geq \widehat{d}^*$.

(P2) The algorithm applies the Ford-Fulkerson labeling algorithm to convert the flow obtained in first phase into a minimum flow.

In the first phase the algorithm might terminate with a nonoptimal solution, the solution is probably nearly-optimal (its value is within d^* of the optimal flow value). The Ford-Fulkerson labeling algorithm converts this near-optimal flow into a minimum flow far more quickly than the Ahuja-Orlin shortest path algorithm.

The second unit capacity minimum flow algorithm is obtained by modifying the procedure BLOCKFLOW in Dinic layered network algorithm. Let $\widehat{G}' = (N, \widehat{A}')$ be the Dinic layered network and

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

We present now the new procedure in Dinic layered network algorithm.

(1)PROCEDURE BLOCKFLOWUNIT ($\widehat{G}', \widehat{f}'$);
(2)BEGIN
(3) $L := \Phi$;
(4) FOR $x \in N$ DO $\widehat{\rho}'_i(x) := 0$;
(5) FOR $(x, y) \in \widehat{A}'$ DO
(6) BEGIN
(7) $\widehat{f}'(x, y) := 0$; $\widehat{\rho}'_i(y) := \widehat{\rho}'_i(y) + 1$;
(8) END;
(9) REPEAT
(10) $y := t$;
(11) FOR $i := \widehat{d}^*$ DOWNTO 1 DO
(12) BEGIN
(13) choose an arc $(x, y) \in \widehat{A}'$;
(14) remove the arc (x, y) from $\widehat{A}'$;

```
(15)        ρ'_i(y) := ρ'_i(y) − 1; f'(x, y) := 1;
(16)        IF ρ'_i(y) = 0 THEN
(17)        BEGIN
(18)            append y to L;
(19)            WHILE L ≠ Φ DO
(20)            BEGIN
(21)              remove the first node u from L;
(22)              FOR (u, v) ∈ E'(u) DO
(23)              BEGIN
(24)                 remove (u, v) from A';
(25)                 ρ'_i(v) := ρ'_i(v) − 1;
(26)                 IF ρ'_i(v) = 0
(27)                    THEN append v to L;
(28)              END;
(29)            END;
(30)        END;
(32)        y := x;
(31)      END;
(33)  UNTIL ρ'_i(t) = 0;
(34) END.
```

Each unit capacity minimum flow algorithm solves a minimum flow problem on unit capacity networks in $O(min\{n^{2/3}m, m^{3/2}\})$ time (see [8]).

5 Minimum flows in bipartite networks with unit capacities

In this section we consider a special case of networks which ocurrs in applications and for which, due to their special structure, one can obtain a faster algorithm for finding a minimum flow.

Let $G = (N_1 \cup N_2, A, l, c, s, t)$ an unbalanced ($n_2 << n_1$) bipartite network with $l(x, y) = 0$ or 1 and $c(x, y) = 1$ for all arcs $(x, y) \in A$. There are many applications of bipartite networks with unit capacities.

For computation of a minimum flow in an unbalanced bipartite network with unit capacities we use the Dinic layered networks algorithm from section 3 with procedure BLOCKFLOWUNIT from section 4. We present now this algorithm (f_0 is a feasible flow in the network G):

```
(1)PROGRAM DLN;
(2)BEGIN
(3)   f := f_0;
(4)   d̂(s) := 0;
(5)   determine the residual network Ĝ;
(6)   WHILE d̂(s) < n DO
(7)   BEGIN
(8)      LAYEREDNETWORK(Ĝ, Ĝ', d̂);
(9)      BLOCKFLOWUNIT(Ĝ', f̂');
(10)     DECREASEFLOW(Ĝ, f, f̂');
(11)  END;
(12)END.
```

```
(1)PROCEDURE LAYEREDNETWORK(Ĝ, Ĝ', d̂);
(2)BEGIN
(3)   determine the exact distance labels d̂(x) in Ĝ;
(4)   determine the layered network Ĝ';
(5)END.
```

```
(1)PROCEDURE DECREASEFLOW(Ĝ, f, f̂');
(2)BEGIN
(3)   f := f − f̂';
(4)   update the residual network Ĝ;
(5)END.
```

Theorem 3 *The Dinic layered network algorithm for unbalanced bipartite network with unit capacities runs in* $O(min\{n_2^{2/3}m, m^{3/2}\})$.

Proof: The Dinic layered network algorithm for unbalanced bipartite network has a time complexity of $O(n_2^2 \cdot m)$ (see section 3). The modified Dinic algorithm in unit capcity networks has a time complexity of $O(min\{n^{2/3}m, m^{3/2}\})$ (see section 4). It results that the Dinic layered network algorithm for unbalanced bipartite network with unit capacities runs in $O(min\{n_2^{2/3}m, m^{3/2}\})$.

6 Applications

In this section we present the scheduling jobs on identical machines. This problem has many practical applications, where *machines* might be workers, tankers, airplanes, truks, processors etc. Three of such examples are treated here as subsections. Another interesting applications are presented in main works.

Let X be a set of jobs which are to be processed by a set of identical machines Y. Each job $x_i \in X$ is processed by one machine $y_j \in Y$. There is a fix schedule for the jobs, specifying that the job $x_i \in X$ must start at time $\tau(x_i)$ and finish at time $\tau'(x_i)$. Furthermore, there is a transition time $\tau''(x_i, y_j)$ required to set up a machine which has just performed the job x_i and will perform the job y_j. The goal is to find a

feasible schedule for the jobs which requires as few machines as possible.

We can formulate this problem as a minimum flow problem in a network G. This network contains a node for each job $x_i \in X$, $i = 1, ..., k$. We split each node x_i into two nodes x_i' and x_i'' and add the arc (x_i', x_i''). We also add a source node s and a sink node t. We connect the source node s to the nodes $x_i' \in X$, $i = 1, ..., k$ and each node $x_i'' \in X$, $i = 1, ..., k$ to the sink node t. If $\tau'(x_i') + \tau''(x_i'', x_j') \leq \tau(x_j')$ then we also add the arc (x_i'', x_j').

Thus, we can formulate this problem as a minimum flow problem in a network $G = (N, A, l, c, s, t)$ where

$$N = N_1 \cup N_2 \cup N_3 \cup N_4,$$
$$N_1 = \{s\},$$
$$N_2 = \{x_i' | i = 1, ..., k\},$$
$$N_3 = \{x_i'' | i = 1, ..., k\},$$
$$N_4 = \{t\},$$
$$A = A_1 \cup A_2 \cup A_3 \cup A_4,$$
$$A_1 = \{(s, x_i') | x_i' \in N_2\},$$
$$A_2 = \{(x_i', x_i'') | x_i' \in N_2, x_i'' \in N_3\},$$
$$A_3 = \{(x_i'', x_j') | \tau'(x_i') + \tau''(x_i'', x_j') \leq \tau(x_j')\},$$
$$A_4 = \{(x_i'', t) | x_i'' \in N_3\},$$
$$l(s, x_i') = 0, c(s, x_i') = 1 \text{ for each } (s, x_i') \in A_1,$$
$$l(x_i', x_i'') = 1, c(x_i', x_i'') = 1 \text{ for each } (x_i', x_i'') \in A_2,$$
$$l(x_i'', x_j') = 0, c(x_i'', x_j') = 1 \text{ for each } (x_i'', x_j') \in A_3,$$
$$l(x_i'', t) = 0, c(x_i'', t) = 1 \text{ for each } (x_i'', t) \in A_4.$$

We present now the airplain scheduling problem.

An airline company has contracted to perform k flights between several different origin-destination pairs. The starting time for flight x_i is $\tau(x_i)$ and the finishing time is $\tau'(x_i)$. The plane requires $\tau''(x_i, x_j)$ hours to return from the point of destination of the flight x_i to the point of origin of the flight x_j. The airline company wants to determine the minimum number of planes needed to perform these flights.

In order to illustrate a modeling approach for this problem, we consider an example with five flights. A plane requires

$$\tau''(x_i, x_j) = \tau'(x_j) - \tau(x_i)$$

hours to return from the point of destination of the flight x_i to the point of origin of the flight x_j. Each flight has the characteristics shown in the table in the figure 5.

Flight i	**Origin** $\tau(i)$	**Destination** $\tau'(i)$
1	Airport a_1 7.00	Airport a_3 9.00
2	Airport a_1 10.00	Airport a_3 12.15
3	Airport a_2 7.30	Airport a_4 8.30
4	Airport a_2 9.00	Airport a_4 12.00
5	Airport a_2 12.00	Airport a_5 14.15

Figure 5: Characteristic of flights

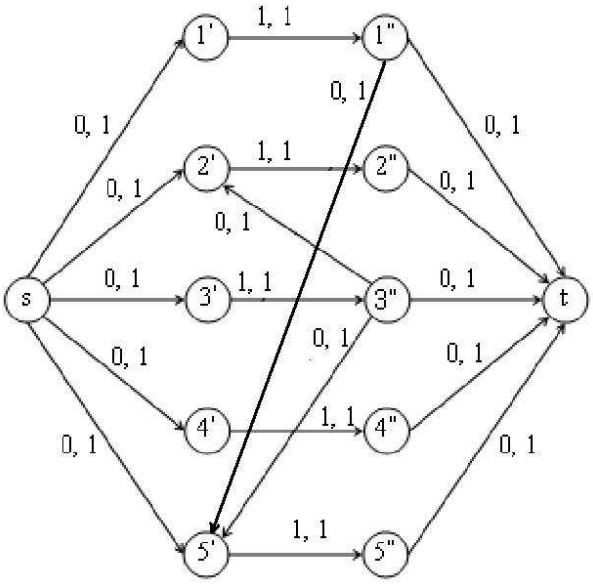

Figure 6: The airline scheduling network

Figure 6 shows the corresponding network $G = (N, A, l, c, s, t)$.

Figure 7 shows a minimum flow f with $v = 3$ in the network G.

The minimum number of planes to perform the flights is three. The first plane performs flights 1 and 5, the second plane performs flights 2 and 3 and the third plane performs flight 4.

A related problem could be a scheduling airplanes problem. This means that an airport has a certain number of runways that can be used for landing of airplanes. How would you schedule airplanes to use the minimum number of runways (in order to possi-

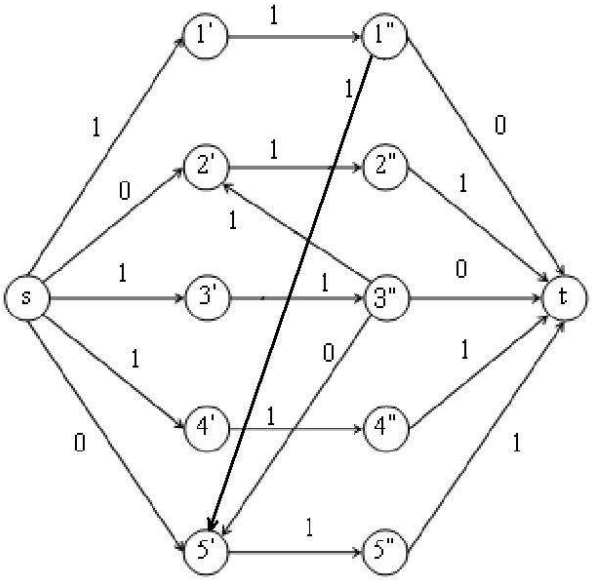

Figure 7: The minimum flow for airline scheduling network

bly have some spare ones permanently ready for emergency landings) if every use of a runway can be determined as fixed time interval? We can solve this problem in the same mode as the previous problems.

The machine setup problem is the following. A job shop needs to perform k jobs on a particular day. It is known the start time $\tau(x_i)$ and the end time $\tau'(x_i)$ for each job $x_i, i = 1, ..., k$. The workers must perform these jobs according to this schedule so that exactly one worker performs each job. A worker cannot work on two jobs at the same time. It is also known the setup time $\tau''(x_i, x_j)$ required for a worker to go from job x_i to job x_j. We wish to find the minimum number of workers to perform the given jobs.

The tanker scheduling problem is the following. A steamship company has contracted to deliver perishable goods between several different origin-destination pairs. Since the cargo is perishable, the customers have specified precise dates (in days) when the shipments must reach their destinations. The steamship company wants to determine the minimum number of ships needed to meet the delivery dates of the shiploads.

There are many other applications of bipartite networks with unit capacities.

Acknowledgements. The research was supported by University Transilvania of Braşov and by Grant IDEI 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] L. Ciupală and E. Ciurea, Sequential and parallel deficit scaling algorithms for minimum flow in bipartite networks, *WSEAS Transactions on Computers*, Issue 10, Vol. 7, 2008, pp. 1545–1554.

[3] L. Ciupală and E. Ciurea, A parallel algorithm for the minimum flow problem in bipartite networks, *Proceedings of the 12th WSEAS International Conference on Computers*, Crete, Greece, 2008, pp. 2003–2007.

[4] 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.

[5] E. Ciurea and L. Ciupală, Algorithms for minimum flows, *Computer Science of Moldova*, Vol. 9(3), 2001, pp. 275–290.

[6] E. Ciurea, O. Georgescu and D. Marinescu, Improved algorithms for minimum flows in bipartite networks, *International Journal of Computers*, Issue 4, Vol. 2, 2008, pp. 351–360.

[7] E. Ciurea, O. Georgescu and D. Marinescu, Minimum flows in bipartite networks, *Proceedings of the 10th WSEAS International Conference on Mathematical and Computational Methods in Science and Engineering*, Bucuresti, 2008, pp. 491–495.

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

[9] E. Ciurea, *An algorithm for minimal dynamic flow*, Korean Journal of Computational and Applied Mathematics, vol. 7(3) (2000), 259-269.

[10] E. Ciurea, A. Deaconu, *Inverse Minimum Flow Problem*, Journal of Applied Mathematics and Computing, vol. 23 (2007), 193-203.

[11] 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.

[12] J.R. Evans, E.Minieka, Optimization algorithms for networks, *Marcel Dekker Inc.*, New York, 1992.

[13] L.R. Ford, D.R. Fulkerson, Flows in networks, *Princeton University Press*, Princeton, New Yersey, 1962.

[14] 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.

[15] 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.

[16] F. Glover, D. Klingman and N.V. Philips, Network Models in Optimization and their Applications in Practice, *Wiley*, New York 1992.

[17] J. Gross, J. Yellen, Graph Theory and its Applications, *CRC Press*, New York 1999.

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

[19] D. Jungnickel, Graphs, Networks and Algorithms, *Springer*, Berlin 1999.

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

[21] S. Lin, H. Chang and 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.

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

[23] G. Ruhe, Algorithmic Aspects of Flows in Networks, *Kluwer Academic Publishers*, Dordrecht 1991.