INTERNATIONAL JOURNAL OF COMPUTERS
Issue 3, Volume 2, 2008

About wave algorithms for network flows problems

Laura A. Ciupală

***Abstract*—** In this paper, we focus on using wave implementations in order to reduce the time complexity of some algorithms that solve the most important network flow problems. We describe wave implementations of known algorithms for the maximum flow problem, for the minimum cost flow problem and for the minimum flow problem.

First, we present a wave preflow algorithm for determining a minimum flow. This algorithm was developed by Ciupală and it is a special implementation of the generic preflow algorithm developed by Ciurea and Ciupală. The wave preflow algorithm is a hybrid between the FIFO preflow algorithm and the highest-label preflow algorithm for minimum flow. It examines the active nodes in nonincreasing order of their distance labels and the node examination terminates when either the node deficit becomes zero or the node is relabeled. The wave preflow algorithm for minimum flow runs in $O(n^3)$ time.

Next, we present the wave algorithm for the maximum flow problem developed by Tarjan. He described a preflow method that is simpler than Karzanov's first preflow method for finding a blocking flow. It is known that a maximum flow can be determined by computing $O(n)$ blocking flows. Consequently, by selecting the nodes in some specific order, wave algorithm developed by Tarjan computes a blocking flow in $O(n^2)$ time and a maximum flow in $O(n^3)$ time.

Finally, we describe the wave implementation of the cost scaling algorithm proposed by Goldberg and Tarjan. By examining the active nodes carefully, the wave implementation, developed also by Goldberg and Tarjan, improves the running time of the cost scaling algorithm from $O(n^2m \log(nB)$ to $O(n^3\log(nB))$.

***Keywords*—** Maximum flow, Minimum flow, Minimum cost flow, Network algorithms, Network flows.

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 50 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.

Manuscript received December 9, 2008: Revised version received December 9, 2008. This work was supported in part by the AT CNCSIS Grant nr. 6GR/2008.

L. A. Ciupală is from Transilvania University, Braşov, Romania. She is within the Department of Computer Science, phone: +40 268 414016; fax: +40 268 414016; e-mail: laura_ciupala@yahoo.com, laura.ciupala@unitbv.ro).

II. Minimum Flow Problem

Although it has its own applications, the minimum flow problem was not dealt so often as the maximum flow ([1], [2], [15], [16], [17], [18], [20], [21]) and the minimum cost flow problem ([1], [2], [5], [21], [22]).

There are many problems that occur in economy that can be reduced to minimum flow problems.

For instance, we present the machine setup problem. A job shop needs to perform p tasks on a particular day. It is known the start time $\pi(i)$ and the end time $\pi'(i)$ for each task i, $i = 1,..., p$. The workers must perform these tasks according to this schedule so that exactly one worker performs each task. A worker cannot work on two jobs at the same time. It is known the setup time $\pi_2(i, j)$ required for a worker to go from task i to task j. We wish to find the minimum number of workers to perform the tasks.

We can formulate this problem as a minimum flow problem in the network $G = (N, A, l, c, s, t)$, determined in the following way:

$$N = N_1 \cup N_2 \cup N_3 \cup N_4,$$
$$N_1 = \{s\},$$
$$N_2 = \{i \mid i=1,...,p\},$$
$$N_3 = \{i' \mid i'=1,...,p\},$$
$$N_4 = \{t\},$$
$$A = A_1 \cup A_2 \cup A_3 \cup A_4,$$
$$A_1 = \{(s, i) \mid i \in N_2\},$$
$$A_2 = \{(i, i') \mid i, i'=1,...,p\},$$
$$A_3 = \{(i', j) \mid \pi'(i')+\pi_2(i', j) \le \pi(j)\},$$
$$A_4 = \{(i', t) \mid i' \in N_3\},$$
$$l(s, i)=0,\ c(s, i)=1, \text{ for any } (s, i) \in A_1,$$
$$l(i, i')=1,\ c(i, i')=1, \text{ for any } (i, i') \in A_2,$$
$$l(i', j)=0,\ c(i', j)=1, \text{ for any } (i', j) \in A_3,$$
$$l(i', t)=0,\ c(i', t)=1, \text{ for any } (i', t) \in A_4.$$

We solve the minimum flow problem in the network $G=(N, A, l, c, s, t)$ and the value of the minimum flow is the minimum number of workers that can perform the tasks.

The minimum flow problem in a network can be solved in two phases:

1. establishing a feasible flow, if there is one
2. from a given feasible flow, establish the minimum flow.

The first phase, i.e. the problem of determining a feasible flow, can be reduced to a maximum flow problem (for details see [1]).

For the second phase of the minimum flow problem there are three approaches:

1. using decreasing path algorithms (see [12], [13])
2. using preflow algorithms (see [4], [7], [8], [9], [10], [12], [13])
3. using minimax algorithm (see [2], [11]).

The decreasing path algorithms maintain mass balance constraints at every node of the network other than the source node and the sink node. These algorithms decrease flow along paths from the source node to the sink node. By determining the decreasing paths with respect to different selection rules, different algorithms were developed.

The preflow algorithms allow that some nodes have deficits. These algorithms select nodes with deficits and reduce their deficit by sending flow from the node backward toward the source node or forward toward the sink node. By imposing different rules for selecting nodes with deficits, different preflow algorithms were obtained.

The third approach for solving the minimum flow problem consists in using the minimax algorithm which determines a minimum flow in a network by computing a maximum flow from the sink node to the source node in the residual network.

The wave algorithm for the minimum flow problem described in this section is part of the second class, which consists in the preflow algorithms that are more efficient and more versatile than the decreasing path algorithms.

A. Notation and Definitions

Given a capacitated network $G = (N, A, l, c, s, t)$ with a nonnegative capacity $c(i, j)$ and with a nonnegative lower bound $l(i, j)$ 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:

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

for some $v \geq 0$, where

$$f(i, N) = \Sigma_j f(i, j),\ i\in N$$

and

$$f(N, i) = \Sigma_j f(j, i),\ i\in N.$$

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. So, the objective is to send as little flow as possible through the network G from the source node s to the sink node t.

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

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

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

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

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

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

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.

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

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

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 $l(j, i) = 0$ and $c(j, i) = 0$. The residual capacity $r(i, j)$ of the arc (i, j) represents the maximum amount of flow from the node i to node j that can be canceled by modifying the flow on 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 preflow f).

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:

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

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

Theorem 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 the source node s to node i in the residual network.*

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

Proof. (a) Let $P = (s=i_1, i_2, \ldots, i_k, i_{k+1}=i)$ be any path of length k from node s to node i in the residual network. The validity conditions imply that:

$$d(i_2) \leq d(i_1) + 1 = d(s)+1 = 1$$
$$d(i_3) \leq d(i_2) + 1 \leq 2$$
$$d(i_4) \leq d(i_3) + 1 \leq 3$$
$$\ldots.$$
$$d(i_{k+1}) \leq d(i_k) + 1 \leq k.$$

(b) We proved that $d(t)$ is a lower bound on the length of the shortest path from the source node s to the sink node t in the residual network and we know that no directed path can contain more than $(n-1)$ arcs. Consequently, if $d(t) \geq n$, then the residual network contains no directed path from s to t.

We say that the distance labels are *exact* if for each node i, $d(i)$ equals the length of the shortest path from node s to node i in the residual network.

We refer to an arc (i, j) from the residual network as an *admissible arc* if $d(j) = d(i) + 1$; otherwise it is *inadmissible*.

We refer to a node i with $e(i) < 0$ as an *active* node. We adopt the convention that the source node and the sink node are never active.

B. The Generic Preflow Algorithm for Minimum Flow

This algorithm was developed by Ciurea and Ciupală in [13] and it begins with a feasible flow and sends back as much flow, as it is possible, from the sink node to the source node. Because the algorithm decreases the flow on individual arcs, it does not satisfy the mass balance constraint (1), (2), (3) at intermediate stages. In fact, it is possible that the flow entering in a node exceeds the flow leaving from it. Such a node is an active node because is has a strictly negative deficit.

The basic operation of the generic preflow algorithm is to select an active node and to send the flow entering in it back, closer to the source. For measuring closeness, the generic preflow algorithm for minimum flow uses the distance labels $d(\cdot)$. Suppose that j is a node with strictly negative deficit selected by the algorithm. If it exists an admissible arc (i, j), it pulls flow on this arc; otherwise it relabels the node j in order to create at least one admissible arc entering in the node j. The generic preflow algorithm for minimum flow repeats this process until the network contains no more active nodes, which means that the preflow is actually a flow. Moreover, it is a minimum flow.

The generic preflow algorithm for the minimum flow problem is the following:

Generic Preflow Algorithm;
Begin
let f be a feasible flow in network G;
compute the exact distance labels $d(\cdot)$ in the residual network G_f;
if t is not labeled **then**
f is a minimum flow
else
begin
for each arc $(i, t) \in A$ **do**
$f(i, t) := l(i, t)$;
$d(t) := n$;
while the network contains an active node **do**
begin
select an active node j;
pull/relabel(j);
end
end
end.

procedure *pull/relabel*(j);
begin
if the network contains an admissible arc (i, j) **then**
pull $g = \min(-e(j), r(i, j))$ units of flow from node j to node i;
else $d(j) := \min\{ d(i) \mid (i, j) \in A_f \} + 1$
end;

We refer to a pull of g units of flow on the admissible arc (i, j) as *canceling* if it deletes the arc (i, j) from the residual network; otherwise it is a *noncanceling pull*.

Theorem 2. ([11]) *If there is a feasible flow in the network $G = (N, A, l, c, s, t)$, the wave preflow algorithm computes correctly a minimum flow.*

Theorem 3. ([11]) *The generic preflow algorithm runs in $O(n^2m)$ time.*

The generic preflow algorithm for minimum flow does not specify any rule for selecting active nodes. By specifying different rules we can develop many different algorithms, which can have better running times then the generic preflow algorithm. For example, we could select active nodes in FIFO order, or we could always select the active node with the greatest distance label, or the active node with the minimum distance label, or the active node selected most recently or least recently, or the active node with the largest deficit or we could select any of active nodes with a sufficiently large deficit.

At an iteration, the generic preflow algorithm for minimum flow selects a node, say node j, and performs a canceling or a noncanceling pull, or relabels the node. If the algorithm performs a canceling pull, then node j might still be active, but, in the next iteration, the algorithm may select another active node for performing a pull or a relabel operation. We can establish the rule that whenever the algorithm selects an active node, it keeps pulling flow from that node until either its deficit becomes zero or the algorithm relabels the node. We refer to a sequence of canceling pulls followed either by a noncanceling pull or a relabel operation as a *node examination*.

C. The Wave Preflow Algorithm

The wave algorithm for minimum flow is a special implementation of the generic preflow algorithm for minimum flow.

The highest-label preflow algorithm for minimum flow examines (described in [7]) always an active node with the highest distance label. The FIFO preflow algorithm (developed in [11]) examines active nodes in FIFO order. The wave algorithm, described in this paragraph, is a hybrid between these two previous preflow algorithms and performs passes over active nodes. In each pass, it examines all the active nodes in nonincreasing order of their distance labels (like the highest-label preflow algorithm) and the node examination terminates when either the node deficit becomes zero or the node is relabeled (like in the FIFO preflow algorithm). In order to do this, it maintains two priority queues L and L_1, both with priority d. The nodes that become active during the initialization are added to L. The algorithm always selects the active node with the highest priority from L and pulls flow toward the source node, adding the newly active nodes in L_1. When L becomes empty, all active nodes from L_1 are moved in L. The algorithm repeats the same process until both L and L_1 become empty (i.e., until during a pass it relabels no node). Consequently, there are no active nodes and the preflow is a flow. Moreover, it is a minimum flow.

The wave preflow algorithm for the minimum flow problem is the following:

```
Wave Preflow Algorithm;
Begin
  let f be a feasible flow in network G;
  compute the exact distance labels d(·) in the residual network G_f;
  if t is not labeled then
  f is a minimum flow
  else
  begin
    L := Ø;
    for each arc (i, t)∈A do
    begin
      f(i, t) := l(i, t);
      if (e(i)<0) and (i ≠ s) then
        add i to the rear of L;
    end;
    d(t) := n;
    L1 := Ø;
    while (L ≠ Ø) and (L1 ≠ Ø) do
    begin
      if L = Ø then
      begin
        L := L1;
        L1 := Ø;
      end;
      remove the node j from the front of the queue L;
      pull/relabel(j);
    end
    end
end.

procedure pull/relabel(j);
begin
  select the first arc (i, j) that enters in node j;
  B := 1;
  repeat
   if (i, j) is an admissible arc then
   begin
     pull g = min(-e(j), r(i, j)) units of flow from node j to
        node i;
     if (i ∉ L1) and (i ≠ s) and (i ≠ t) then
       add i to the rear of L1;
   end;
   if e(j) < 0 then
     if (i, j) is not the last arc entering in node j then
       select the next arc (i, j) that enters in node j
     else
     begin
       d(j) := min{ d(i) | (i, j) ∈A_f }+1;
       B := 0;
     end;
until (e(j) = 0) or (B = 0);
if e(j) < 0 then
   add j to the rear of L1;
end;
```

Theorem 4. *If there is a feasible flow in the network $G = (N, A, l, c, s, t)$, the wave preflow algorithm computes correctly a minimum flow.*

Proof. The correctness of the wave preflow algorithm follows from the correctness of the generic preflow algorithm for minimum flow (for details see [11]).

Theorem 5. *The wave preflow algorithm runs in $O(n^3)$ time.*

Proof. This theorem can be proved in a manner similar to the proof of the complexity of the FIFO preflow algorithm for the minimum flow (for details see [11]).

III. Maximum Flow Problem

The maximum flow problem is one of the most 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 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.
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.

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:

$$f(s, N) - f(N, s) = v \quad (8)$$
$$f(i, N) - f(N, i) = 0, \; i \neq s,t \quad (9)$$
$$f(t, N) - f(N, t) = -v \quad (10)$$
$$0 \leq f(i, j) \leq c(i, j), \; (i, j) \in A \quad (11)$$

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:

$$f(i, N) - f(N, i) \geq 0, \; i \neq s,t \quad (12)$$
$$0 \leq f(i, j) \leq c(i, j), \; (i, j) \in A \quad (13)$$

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

$$e(i) = f(i, N) - f(N, i) \quad (14)$$

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.

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

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 6.([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.

A preflow is *blocking* if it saturates an arc on every path from s to t.

B. The Wave Algorithm

The wave method for maximum flow, developed by Tarjan in [21], finds a blocking preflow and gradually converts it into a blocking flow by balancing nodes, in successive forward and backward passes over the network.

Each node is in one of the two states: *unblocked* or *blocked*. An unblocked node can become blocked but not vice versa. We balance an unblocked node i by increasing the outgoing flow if i is unblocked and decreasing the incoming flow if i is blocked. More precisely, we balance an unblocked node i by repeating the following step until $e(i) = 0$ (the balancing succeeds) or there is no unsaturated arc (i, j) such that j is unblocked (the balancing fails):

INCREASING STEP. Let (i, j) be an unsaturated arc such that j is unblocked. Increase $f(i, j)$ by $\min\{c(i, j) - f(i, j), e(i)\}$.

We balance a blocked node i by repeating the following step until $e(i) = 0$ (such a balancing always succeeds):

DECREASING STEP. Let (k, i) be an arc of positive flow. Decrease $f(k, i)$ by $\min\{f(k, i), e(i)\}$.

To find a blocking flow, we begin with a preflow that saturates every arc out of s and is zero on all other arcs, make s blocked and every other node unblocked, and repeat *increase flow* followed by *decrease flow* until there are no unbalanced nodes.

Increase flow. Scan the nodes other than s and t in topological order, balancing each node i that is unbalanced and unblocked when it is scanned; if balancing fails, make i blocked.

Decrease flow. Scan the nodes other than s and t in reverse topological order, balancing each node that is unbalanced and blocked when it is scanned.

Theorem 7. The wave algorithm correctly computes a blocking flow in $O(n^2)$ time and a maximum flow in $O(n^3)$ time.

Proof. The method maintains the invariant that if i is blocked, every path from i to t contains a saturated arc. Since s is blocked initially, every preflow constructed by the algorithm is blocking. Scanning in topological order during increase flow guarantees that after such a step there are no unblocked, unbalanced nodes. Similarly each node blocked before a decrease flow step is balanced after the step and remains balanced during the next increase step, if any. Thus each increase flow step except the last blocks at least one node, and the method halts after at most n-1 iterations of *increase flow* and *decrease flow*, having balanced all nodes except s and t and thus having produced a blocking flow.

There are at most $(n-2)(n-1)$ balancings. The flow on an arc (i, j) first increases (while j is unblocked), the decreases (while j is blocked). Each increasing step either saturates an arc or terminates a balancing, each decreasing step either decreases the flow on an arc to zero or terminates a balancing. Thus there are at most $2m+(n-2)(n-1)$ increasing and decreasing steps.

To implement the method efficiently, we maintain for each node i the value of $e(i)$ and a bit indicating whether i is unblocked or blocked. To balance an unblocked node i, we examine the arcs out of i, beginning with the last arc previously examined and increase the flow on each arc to which the increasing step applies, until $e(i) = 0$ or we run out of arcs (the balancing fails). Balancing a blocked node is similar. With such an implementation the method takes $O(n^2)$ time to find a blocking flow, including the time to topologically order the nodes. Thus a maximum flow is found in $O(n^3)$ time.

When using the wave algorithm to find the maximum flow, we can use the layered structure of the level graphs to find each blocking flow in $O(m+k)$ time, where k is the number of balancings, eliminating the $O(n^2)$ overhead for scanning balanced nodes. This may give an improvement in practice, though the time bound is still $O(n^2)$ in the worst case.

IV. Minimum Cost Flow

The minimum cost flow problem, as well as one of its special cases which is the maximum flow problem, is one of the most 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 problem 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

$$\text{minimize} \quad \sum_{(i,j)\in A} b(i, j) f(i, j) \tag{15}$$

subject to

$$\sum_{j|(i,j)\in A} f(i, j) - \sum_{j|(j,i)\in A} f(j, i) = v(i), \forall i \in N \tag{16}$$

$$0 \le f(i, j) \le c(i, j), \forall (i, j) \in A. \tag{17}$$

A flow f satisfying the conditions (16) and (17) 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 section III.

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 8. ([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 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 $f(i, j) = 0$ (18)
If $0 < f(i, j) < c(i, j)$, then $b^{\pi}(i, j) = 0$ (19)
If $b^{\pi}(i, j) < 0$, then $f(i, j) = c(i, j)$ (20)

A *pseudoflow* is a function $f : A \rightarrow \mathbf{R}_+$ satisfying the only conditions (14).

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*.

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:

$b^{\pi}(i, j) \geq 0$ for every arc (i, j) 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*:

If $b^{\pi}(i, j) > \varepsilon$, then $f(i, j) = 0$ (21)
If $-\varepsilon \leq b^{\pi}(i, j) \leq \varepsilon$, then $0 \leq f(i, j) \leq c(i, j)$ (22)
If $b^{\pi}(i, j) < -\varepsilon$, then $f(i, j) = c(i, j)$ (23)

These conditions are relaxations of the (exact)

complementary slackness optimality conditions (18) - (20) and they reduce to complementary slackness optimality conditions when $\varepsilon = 0$.

B. The cost scaling algorithm

The cost scaling algorithm developed by Goldberg and Tarjan treats ε as a parameter and iteratively obtains ε-optimal flows for successively smaller values of ε. Initially, $\varepsilon = B$ and any feasible flow is ε-optimal. The algorithm then performs cost scaling phases by repeatedly applying an improve-approximation procedure that transforms an ε-optimal flow into an $\varepsilon/2$-optimal flow. After $1 + [\log(nB)]$ cost scaling phases, $\varepsilon < 1/n$ and the algorithm terminates with an optimal flow.

The cost scaling algorithm is the following:

```
Cost Scaling Algorithm;
Begin
   π := 0;
   ε = B;
   while ε ≥ 1/n do
   begin
      improve-approximation(ε, f, π);
      ε := ε/2;
   end;
end.

procedure improve-approximation(ε, f, π);
begin
   for(i, j)∈A do
    if b^π(i, j)> 0 then
      f(i, j) := 0;
    else if b^π(i, j) < 0 then
              f(i, j) := c(i, j);
    compute nodes imbalances;
    while the network contains an active node do
    begin
      select an active node i;
      push/relabel(j);
    end;
end;

procedure push/relabel(i);
begin
    if the residual network contains an admissible (i, j) then
              push g = min(e(i), r(i, j)) units of flow from node
      i to node j;
   else
      π(i) := π(i) + ε/2;
end;
```

We refer to a push of g units of flow on the admissible arc (i, j) as *saturating* if it saturates the arc (i, j); otherwise it is a *nonsaturating push*.

The *improve-approximation* procedure transforms an ε-optimal flow into an $\varepsilon/2$-optimal flow. This transformation consists in converting an ε-optimal flow into an $\varepsilon/2$-optimal pseudoflow and then gradually converting the pseudoflow into a flow while always maintaining $\varepsilon/2$-optimality of the solution.

We refer to a node i with $e(i) > 0$ as an *active node* and say that an arc (i, j) in the residual network is *admissible* if $-\varepsilon/2 \le b^{\pi}_{(i,\ j)} < 0$. The admissible network is a subgraph of the residual network consisting only in admissible arcs.

The basic operation in the *improve-approximation* procedure is to select an active node i and to perform pushes on admissible arcs (i, j) emanating from node i. When the network contains no admissible arc, the algorithm updates the node potential $\pi(i)$ in order to create new admissible arcs emanating from node i.

To identify admissible arcs emanating from node i, we use the following data structure: for each node i, we maintain a *current-arc* (i, j) which is the current candidate to test for admissibility. Initially, the current-arc of node i is the first arc in its arc list $A(i)$. To determine an admissible arc emanating from node i, the algorithm checks whether the node's current-arc is admissible, and if not, choose the next arc in the arc list as the current arc. Consequently, the algorithm passes through the arc list starting with the current-arc until it finds an admissible arc or it reaches the end of the arc list. If the algorithm reaches the end of the arc list without finding an admissible arc, it declares that the node has no admissible arc. At this point, it relabels node i and again sets its current-arc to the first arc in the arc list $A(i)$.

Theorem 11. ([1]) The cost scaling algorithm solves correctly the minimum cost flow problem in $O(n^2 m \log(nB))$ time.

The cost scaling algorithm starts with $\varepsilon = B$ and reduces ε by a factor of 2 in every scaling phase until $\varepsilon < 1/n$. As a consequence, ε could become nonintegral during the execution of the algorithm. By slightly modifying the algorithm, we can ensure that ε remains integral. We do so multiplying all the arc costs by n, by setting the initial value of ε equal to $2^{[\log(nB)]}$ and by terminating the algorithm when $\varepsilon < 1$. It is possible to show that the modified algorithm would yield an optimal flow for the minimum cost flow problem in the same computational time.

C. Wave implementation

In an iteration of the *improve-approximation* procedure, the algorithm selects a node, say node i, and either performs a saturating push or a nonsaturating push from this node, or relabels the node. If the algorithm performs a saturating push, node i might still be active, but the algorithm might select another node in the next iteration. We shall henceforth assume that whenever the algorithm selects a node, it keeps pushing flow from this node until either its excess becomes zero or the node becomes relabeled. If we adopt this node selection strategy, the algorithm will perform several saturating pushes from a particular node followed either by a nonsaturating push or a relabel operation. We refer to this sequence of operations as a *node examination*.

The wave implementation is a special implementation of the *improve-approximation* procedure that selects active nodes for node examinations in a specific order. The algorithm uses the fact that the admissible network is acyclic. Consequently, it is possible to put the nodes in topological order. For a given topological order, we define the *rank* of a node as n minus its number in the topological sequence.

Observe that each push carries flow from anode with higher rank to a node with lower rank. Also observe that pushes do not change the topological ordering of nodes since they do not create new admissible arcs. The relabel operations, however, might create new admissible arcs and consequently, might effect the topological ordering of nodes.

The wave implementation sequentially examines nodes in the topological order and if the node being examined is active, it performs push/relabel steps at the node until either the node becomes inactive or it becomes relabeled. When examined in this order, the active nodes push their excesses to nodes with lower rank, which in turn push their excesses to nodes with even lower rank and so on. A relabel operation changes the topological order; so after each relabel operation the algorithm modifies the topological order and again starts to examine nodes according to the new topological order. If within n consecutive node examinations, the algorithm performs no relabel operation, then at this point all the active nodes have discharged their excesses and the algorithm has obtained a flow. Since the algorithm performs $O(n^2)$ relabel operations, we immediately obtain a bound of $O(n^3)$ on the number of node examinations. Each node examination entails at most one nonsaturating push. Consequently, the wave algorithm performs $O(n^3)$ nonsaturating pushes per execution of *improve-approximation* procedure.

We need to describe a procedure for obtaining a topological order of nodes after each relabel operation. It is well known that we can determine a topological ordering of nodes in a network with n nodes and m arcs in $O(m)$ time (for details see [1]). So, we can use an $O(m)$ algorithm for obtaining an initial topological ordering of the nodes. Suppose that while examining node i, the algorithm relabels this node. At this point, the network contains no incoming admissible arc at node i. We claim that if we move node i from its present position to the first position in the previous topological order leaving all other nodes intact, we obtain a topological order of the new admissible network. This method works because: (1) after the relabeling, node i has no incoming admissible arc, so assigning it to the first place in the topological order is justified; (2) the relabeling, node i might create some new outgoing admissible arcs (i, j) but since node i is first in the topological order, any such arc satisfies the conditions of a topological ordering; and (3) the rest of the admissible network does not change, so the previous order remains valid. Therefore, the algorithm maintains an ordered set of nodes and examines nodes in this order. Whenever it ralabels a node i, the algorithm moves this node to the first place in the order and again examines nodes in order starting from node i.

We have established the following result:

Theorem 12. ([1]) The wave implementation of the cost scaling algorithm solves correctly the minimum cost flow problem in $O(n^3\log(nB))$ time.

Consequently, by examining the active nodes carefully and thereby reducing the number of nonsaturating pushes, the wave implementation improve the running time of the generic implementation of the *improve-approximation* procedure from $O(n^2m)$ to $O(n^3)$ and the running time of the cost scaling algorithm from $O(n^2m \log(nB)$ to $O(n^3\log(nB))$.

V. Conclusion

One of the major aims of any computer science researcher is to develop more and more efficient algorithms for solving certain problems. In the domain of network flow theory, first algorithm for maximum flow and first algorithm for minimum cost flow were developed more than a half of century ago. The problem of determining a minimum flow was formulated and first solved more recently. But since each of these three problems was formulated, researchers continuously designed more efficient algorithms for solving them. They demonstrated how the use of clever data structures and careful analysis can improve the theoretical performance of network algorithms. They have revealed the power of methods like scaling the problem data for improving algorithmic performance. The researchers have shown that, in some cases, new insights and simple algorithmic ideas can still produce better algorithms. In this paper, we used another powerful approach for improving the algorithmic performance of some network flow algorithms. We described wave implementations for algorithms that solve minimum flow problem, maximum flow problem and minimum cost flow problem. We showed that, by simply examining the nodes carefully, the wave implementation of some algorithms can improve their running time.

Ideas for further improvements should be: to develop wave implementations for other network flow algorithms and to combine the wave implementation with the use of some enhanced data structure in order to produce even more efficient algorithms.

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. 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.

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

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

[17] 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.

[18] 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.

[19] 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.

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

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

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