STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

MINIMUM FLOW ALGORITHMS. DYNAMIC TREE IMPLEMENTATIONS

ELEONOR CIUREA, OANA GEORGESCU, AND MIHAI IOLU

Abstract. We present augmenting path algorithms for minimum flows with dynamic tree implementations. The time bounds for augmenting path for minimum flows automatically improve when the algorithms are implemented with dynamic tree.

Key words: Graph algorithms, Dynamic tree, Minimum flow problems

1. Introduction

The computation of a maximum flow in a graph has been an important and well studied problem, both in the fields of computer science and operations research. Many efficient algorithms have been developed to solve this problem, see, e.g., [1], [3]. Sleator and Tarjan [7] developed the dynamic tree data structure and used it to improve the worst-case complexity of Dinic's algorithm from $O(n^2m)$ to $O(nmlogn)$. Since then, researchers have used this data structure on many occasions to improve the performance of a range of network flow algorithms. Using the dynamic tree data structure, Goldberg and Tarjan [6] improved the complexity of the FIFO preflow-push algorithm from $O(n^3)$ to $O(nmlog(n^2/m))$, and Ahuja, Orlin and Tarjan [2] improved the complexity of the excess scalling algorithm and several of its variants.

The computation of a minimum flow in a network has been investigated by Ciurea and Ciupala [4]. The algorithms for minimum flows with dynamic tree implementations were not treated. In this paper we present the augmenting path algorithms for minimum flows with dynamic tree implementations.

In the presentation to follow, we assume familiarity with flow problems and we omit many details. The reader interested in further details is urged

Received by the editors: May 1, 2008.

2000 *Mathematics Subject Classification.* 90B10, 90C35, 05C35, 68R10.

1998 *CR Categories and Descriptors.* G.2.2 [**Graph Theory**]: Subtopic – *Network problems*.

to consult the books [1, 3] for maximum flow problem and the paper [4] for minimum flow problem.

2. Terminology and Preliminaries

We consider a capacitated network $G = (N, A, b, c, s, t)$ with a nonnegative capacity $c(x, y)$ and with a nonnegative lower bound $l(x, y)$ 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 subset X, Y of the nodes set N of a network 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 $f : A \to R+$, satisfying the next conditions:

$$f(x, N) - f(N, x) = \begin{cases} v, & x = s \\ 0, & x \neq s, t \\ -v, & x = t \end{cases} \tag{2.1a}$$

$$l(x, y) \leq f(x, y) \leq c(x, y), \forall (x, y) \in A \tag{2.1b}$$

for some v. We refer to v as the value of the flow f. The maximum (minimum) flow problem is to determine a flow $\tilde{f}(\hat{f})$ for which v is maximized (minimized). A cut is a partition of the node set N into two subset S and $T = N - S$; we represent this cut using the notation $[S, T]$. We refer to a cut $[S, T]$ as a $s - t$ 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 backward arcs.

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

$$\tilde{c}[S, T] = c(S, T) - l(T, S) \tag{2.2}$$

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

$$\hat{c}[S, T] = l(S, T) - c(T, S) \tag{2.3}$$

We refer to a $s-t$ cut where capacity $\tilde{c}[S,T]$ (capacity $\hat{c}[S,T]$) is the minimum (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:

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

$(P2)$ from a given feasible flow f, establishing the maximum flow $\tilde{f}$ (minimum flow $\hat{f}$)

Theorem 2.1. *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*

$$v=f[S,T]=f(S,T)-f(T,S) \tag{2.4a}$$

and therefore, in particular,

$$\hat{c}[S,T]\le v\le \tilde{c}[S,T] \tag{2.4b}$$

Theorem 2.2. *Let $G=(N,A,l,c,s,t)$ be a network, $[\tilde{S},\tilde{T}]$ a minimum $s-t$ cut and $[\hat{S},\hat{T}]$ a maximum $s-t$ cut. Denote the values of a maximum flow $\tilde{f}$ and a minimum flow $\hat{f}$ by $\tilde{v}$ and $\hat{v}$, respectively. Then*

$$\tilde{v}=\tilde{c}[\tilde{S},\tilde{T}] \tag{2.5a}$$

and

$$\hat{v}=\hat{c}[\hat{S},\hat{T}] \tag{2.5b}$$

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 $c(y,x)=0$. For the maximum (minimum) flow problem, the residual capacity $\tilde{r}(x,y)(\hat{r}(x,y))$ of any arc $(x,y)\in A$, with respect to a given flow f, is given by $\tilde{r}(x,y)=c(x,y)-f(x,y)+f(y,x)-l(y,x)$ $(\hat{r}(x,y)=c(y,x)-f(y,x)+f(x,y)-l(x,y))$. The network $\tilde{G}(f)=(N,\hat{A})$ $(\hat{G}(f)=(N,\hat{A}))$ consisting only of the arcs with $\tilde{r}(x,y)>0$ $(\hat{r}(x,y)>0)$ is referred to as the residual network with respect to flow f for maximum (minimum) flow problem.

There are two approaches for solving maximum flow problem [1]:

$(\tilde{1})$ using augmenting directed path algorithms from source node s to sink node t in residual network $\tilde{G}(f)$;

$(\tilde{2})$ using preflow-push algorithms starting from source node s in residual network $\tilde{G}(f)$.

There are three approaches for solving minimum flow problem [4]:

($\hat{1}$) using decreasing directed path algorithms from source node s to sink node t in residual network $\hat{G}(f)$;
($\hat{2}$) using preflow-pull algorithms starting from sink node t in residual network $\hat{G}(f)$;
($\hat{3}$) using augmenting directed path algorithms from sink node t to source node s or using preflow-push algorithms starting from sink node t in residual network $\tilde{G}(f)$.

In this paper we present the augmenting directed path algorithms from sink node t to source nodes s with dynamic tree implementations for solving minimum flow problems.

All the algorithms from Table 2.1 are augmenting path algorithms, i.e. algorithms which determine augmenting directed path from source node s to sink node t (by different rules) in residual network $\tilde{G}(f)$ and then augment flows along these paths.

We have $n = |N|$, $m = |A|$, $\bar{c} = max\{c(x,y)|(x,y) \in A\}$.

Augmenting path algorithms	Running time
General augmenting path algorithm	$O(nm\bar{c})$
Ford-Fulkerson labelling algorithm	$O(nm\bar{c})$
Gabow bit scaling algorithm	$O(nmlog(\bar{c}))$
Ahuja-Orlin maximum scaling algorithm	$O(nmlog(\bar{c}))$
Edmonds-Karp shortest path algorithm	$O(nm^2)$
Ahuja-Orlin shortest path algorithm	$O(n^2m)$
Dinic layered networks algorithm	$O(n^2m)$
Ahuja-Orlin layered networks algorithm	

TABLE 2.1. Running times for augmenting path algorithms

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

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

3. Augmenting path algorithms for minimum flows. Dynamic tree implementations

A dynamic tree is an important data structure that researchers have used extensively to improve the worst-case complexity of several network algorithms. In this section we describe the use of this data structure for augmenting direct path algorithms from sink node t to source node s.

The dynamic tree data structure maintains a collection T of node-disjoint rooted trees, each arc with an associated value. Each rooted tree is a directed in-tree with a unique root. Each node x (except the root node) has a unique predecessor, which is the next node on the unique path in the tree from that node to the root. We store the predecessor of node x using a predecessor index $p(x)$. If $y = p(x)$, we say that node y is the predecessor of node x and node x is a successor of node y. These predecessor indices uniquely define a rooted tree and also allow us to trace out the unique direct path from any node back to the root. The descendants of a node x consist of the node itself, its successors, successors of its successors and so on. We say that a node is an ancestor of all of its descendants. Notice that, according to our definitions, each node is its own ancestor and descendant.

This data structure supports the six operations obtained by performing the following six procedures:

$ROOT(x)$: finds the root of the tree containing node x;
$VALUE(x)$: finds the value of the tree arc leaving node x;
if node x is a root node, then it returns the value ∞;
$ANCES(x)$: finds the ancestor u of x with the minimum value $VALUE(u)$;
in case of a tie, chooses the node u closest to the tree root;
$CHANGE(x, \overline{w})$: adds a real number $\overline{w}$ to the value of every arc along the directed path from node x to $ROOT(x)$;
$LINK(x, y, w)$: combines the tree containing tree root x and tree containing node y the predecessor of node x and giving arc (x, y) the value w;
$DELET(x)$: break the tree containing node x into two trees by deleting the arc joining node x to its predecessor; we perform this operation when x is not a tree root;

The following important result lies at the heart of the efficiency of the dynamic tree data structure[1].

Theorem 3.1. *If q is the maximum number of nodes in any tree, a sequence of k tree operations, starting with an initial collection of singleton trees, requires a total of $O(klog(k + q))$ time.*

The dynamic tree implementation stores the values of tree only implicitly. Storing the values implicitly allows us to update the values in only $O(logq)$ time.

Before describing the augmenting directed path algorithms for sink node t to source node s with dynamic tree implementation for minimum flow problem, we introduce some definitions. In the residual network $\tilde{G}(f)$, the distance function with respect to a given flow f is a function $\tilde{d}' : N \to N$ from the set of nodes N to the nonnegative integers N. We say that a distance function $\tilde{d}'$ is valid if it satisfies the following conditions:

$$\tilde{d}'(s) = 0 \tag{3.1a}$$

$$\tilde{d}' \leq \tilde{d}'(y) + 1 \text{ for every arc } (x, y) \in \tilde{A} \tag{3.1b}$$

We refer to $\tilde{d}'$ as the distance label of node x. We say that distance labels are exact if for each node x, $\tilde{d}'(x)$ equals the length of the shortest directed path from node x to source node s in the residual network $\tilde{G}(f)$. If $\tilde{d}'(x) = \tilde{d}'(y)+1$ we refer to arc (x, y) as an admissible arc, otherwise inadmissible. We refer to a directed path from sink node t to source node s consisting entirely of admissible arcs as an admissible directed path.

The following results are very important:

Theorem 3.2. *An admissible directed path in the residual network $\tilde{G}(f)$ is a shortest directed path from the sink node t to the source node s.*

Theorem 3.3. *If $\tilde{d}'(t) \geq n$, the residual network $\tilde{G}(f)$ contains no directed path from the sink node t to the source node s.*

These theorems can be proved in a manner similar to the proof of distance function $\tilde{d}$ for maximum flow problem [1] or similar to the proof of distance function $\hat{d}$ for minimum flow problem [4].

We can determine exact distance labels $\tilde{d}'(x)$ for all nodes in $O(m)$ time by performing a backward breadth first search of the residual network $\tilde{G}(f)$ from node source s to node sink t.

How we might use the dynamic tree data structure to improve the computational performance of augmenting directed path algorithms for sink node t to source node s? Let us use the variant of Ahuja-Orlin shortest path algorithm as an illustration. The following basic idea underlies the algorithmic speed-up. In the dynamic tree implementation, each arc in the rooted tree is an admissible arc. The value of an arc is its residual capacity.

Figures 3.1 and 3.2 give a formal statement of the algorithm.

The first two procedures, TADV and TRET, are straight forward, but the TAUG procedure requires some explanation. If node u is an ancestor of sink node t with the minimum value of $VALUE(u)$ and among such nodes in the

```
ALGORITHM TAP;
BEGIN
  let f be a feasible flow in network G;
  determine the residual network $\tilde{G}(f)$;
  compute the exact distance labels $\tilde{d}'(x)$ in $\tilde{G}(f)$;
  let $\tilde{T}$ be the collection of all singleton nodes;
  x:=t;
  WHILE $\tilde{d}'(t) < n$ DO
    BEGIN
      IF exists admissibile arc (x,y) in $\tilde{G}(f)$
        THEN TADV(x)
        ELSE TRET(x);
      IF x=s
        THEN TAUG;
    END;
END.
```

FIGURE 3.1. Dynamic tree implementation for the variant of Ahuja-Orlin shortest path algorithm.

directed path, it is closest to the source node s, then value $VALUE(u)$ gives the residual capacity of the augmenting path.

The procedure $CHANGE(t, -w)$ implicitly updates the residual capacities of all the arcs in the augmenting directed path. This augmentation might cause the capacity of move than one arc in the directed path to become zero. The WHILE loop identifies all such arcs, one by one and deletes them from the collection of rooted trees T.

Theorem 3.4. *The TAP algorithm correctly computes a minimum flow.*

Proof. The TAP algorithm is same as the variant of Ahuja-Orlin shortest path algorithm except that it performs the procedures TADV, TRET and TAUG differently using trees. Notice that, knowing a feasible flow, we determine a minimum flow from the source node s to the sink node t by establishing a maximum flow from the sink node t to the source node s [4]. □

Theorem 3.5. *The TAP algorithm solves the minimum flow problem in $O(nmlogn)$ time.*

Proof. Using simple augments, we can show that the algorithm performs each of the six tree operations $O(nm)$ times. It performs each tree operations on a tree of maximum size n. The use of Theorem 3.1 establishes the result. □

```
PROCEDURE TADV(x);
BEGIN
  LINK(x,y,$\tilde{r}(x,y)$);
  x:=ROOT(y);
END;
```

```
PROCEDURE TRET(x);
BEGIN
  $\tilde{d}'(x) := min\{\tilde{d}'(y)+1|(x,y)\in\tilde{A}\}$;
  FOR $(z,x)\ \in\ \tilde{T}$ DO
    DELET(z);
  x:=ROOT(t);
END;
```

```
PROCEDURE TAUG;
BEGIN
  u:=ANCES(t);
  w:=VALUE(u);
  CHANGE(t,-w);
  WHILE VALUE(u)=0 DO
    BEGIN
      DELET(u);
      u:=ANCES(t);
    END;
  x:=ROOT(t);
  update the residual network $\tilde{G}(f)$;
END;
```

FIGURE 3.2. Procedures of TAP algorithm

4. EXAMPLE

Consider the network $G = (N, A, l, f, c, s, t)$ given in figure 4.1 with $s = 1$ and $t = 4$, which shows the lower bounds, feasible flows and capacities next to the arcs. The residual network $\tilde{G}(f)$ is shown in figure 4.2, which represents the distance labels next to the nodes and residual capacities next to the arcs.

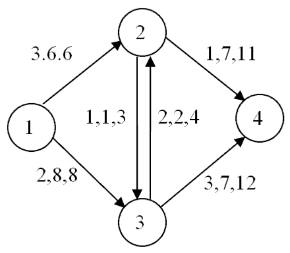

FIGURE 4.1

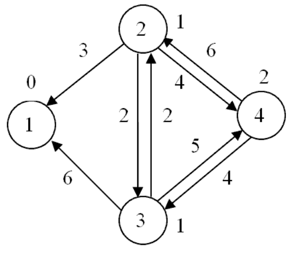

FIGURE 4.2

Figure 4.3 shows the collection of singleton nodes $\tilde{T}$. The algorithm starts with the singleton tree containing only the sink node 4. Suppose that algorithm selects admissible arc $(4, 2)$ and it performs the procedure $TADV(x)$ for $x = 4$, $x = 2$. The algorithm obtains a rooted tree that contains both the sink and source nodes (see figure 4.4).

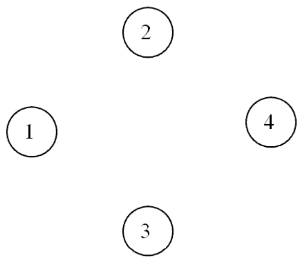

Figure 4.3

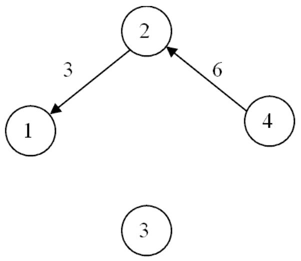

Figure 4.4

The procedure $TAUG$ determines the collection $\tilde{T}$ of node-disjoint rooted trees from figure 4.5, $x = 2$ and residual network $\tilde{G}(f)$ from figure 4.6. But node 2 has no outgoing admissible arc; so the algorithm performs the procedure $TRET(2)$, which increases the distance label of node 2 to 2 ($\tilde{d}' = (0, 2, 1, 2)$), determines the rooted trees $\tilde{T}$ from figure 4.3 and $x = 4$.

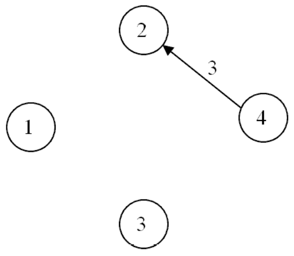

Figure 4.5

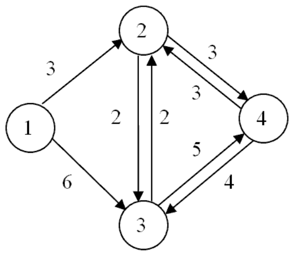

Figure 4.6

Figure 4.7 shows the last residual network $\tilde{G}(f)$ and figure 4.8 shows the lower bounds, minimum flows and capacities next to the arcs.

References

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

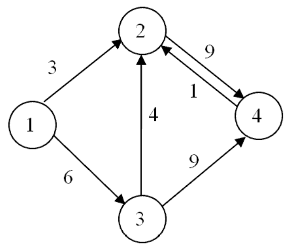

FIGURE 4.7

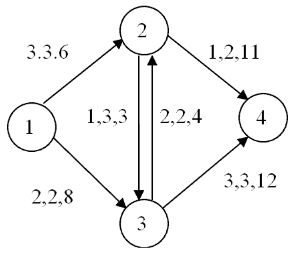

FIGURE 4.8

[2] R. Ahuja, J. Orlin and R. Tarjan, *Improved time bounds for the maximum flow problem*, SIAM Journal on Computing, 18(5) (1989), pp. 939-954.

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

[4] E. Ciurea, L. Ciupala, *Sequential and parallel algorithms for minimum flow*, Journal of Applied Mathematics and Computing, 15 1-2 (2004), pp. 53-75.

[5] E. Ciurea, O. Georgescu, *Minimum flows in unit capacity networks*, Analele Universitatii Bucuresti, XLV (2006), pp. 11-20.

[6] A. Goldberg, R. Tarjan, *A new approach to the maximum flow problem*, Journal of ACM, 35 (1988), pp. 921-940.

[7] D. Sleator, R. Tarjan, *A data structure for dynamic trees*, Journal of Computer and System Sciences, 24 (1983), pp. 362-391.

DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF MATHEMATICS AND INFORMATICS, TRANSILVANIA UNIVERSITY OF BRAŞOV, STR. IULIU MANIU NR. 50, 500091, BRAŞOV RO

E-mail address: {`e.ciurea, o.georgescu`}`@unitbv.ro, mihaiiolu@yahoo.com`