Bulletin of the *Transilvania* University of Braşov • Vol 7(56), No. 1 - 2014
Series III: Mathematics, Informatics, Physics, 103-108

A GENERIC PREFLOW ALGORITHM FOR MAXIMUM FLOW IN SEMI-BIPARTITE NETWORKS

Laura CIUPALĂ[1]

Abstract

In this paper, we develop a generic algorithm for determining a maximum flow in a semi-bipartite network. This algorithm allows only nodes in N_1 to be active. For this reason, it performs pushes on individual arcs having both endpoints in N_1 or admissible paths of length 2 whose starting and ending nodes are both contained in N_1. The running time of this algorithm is $O(n_1^2 m)$.

2000 *Mathematics Subject Classification:* 90B10, 90C90.
Key words: network flow, maximum flow, bipartite network, semi-bipartite network.

1 Introduction

Network flow problems are a group of network optimization problems that are widely and intensively studied because of their widespread and diverse applications. The research focused on network flow started more than 60 years ago. Since then researchers have made continuous improvements to algorithms for solving several classes of problems, including maximum flow problem and minimimum cost flow problem. From the late 1940s through the 1950s, researchers designed many of the fundamental algorithms for network flow. 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.

Researchers were also interested in finding a balance between the generality and the specificity of their results.

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. Some of these can be modelled and solved as network flow problems in special networks. Using the particularities of

[1]Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: laura_ciupala@yahoo.com

these networks, one can develop network flow algorithms that are more efficient than the corresponding algorithms designed for regular networks.

In this paper, we develop a generic preflow algorithm for determining a maximum flow in a semi-bipartite network. This algorithm is a modified version of the generic preflow algorithm for regular networks([1]), but it is more efficient than this one because it uses the particular structure of a semi-bipartite network. The time complexity of our algorithm is $O(n_1^2 m)$ time.

2 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 $(x, y) \in A$ has a nonnegative capacity $c(x, y)$. In the directed network $G = (N, A, c, s, t)$, two special nodes are specified: s is the source node and t is the sink node.

Let X and Y be two subsets of the node set N. We define the set of arcs $(X, Y) = \{(x, y) | (x, y) \in A,\ x \in X, y \in Y\}$.

For any function $g : N \times N \to \mathbb{R}^+$ and for any function $h : N \to \mathbb{R}^+$ we define

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

and

$$h(X) = \sum_{X} h(x).$$

If $X = \{x\}$ or $Y = \{y\}$ then we will use $g(x, Y)$ or $g(X, y)$ instead of $g(X, Y)$.

A *flow* from the source node s to the sink node t in the directed network $G = (N, A, c, s, t)$ is a function $f : A \to \mathbb{R}^+$ which meets the follwing conditions:

$$f(x, N) - f(N, x) = \begin{cases} v, x = s \\ 0, x \neq s, t \\ -v, x = t \end{cases} \tag{1}$$

$$0 \leq f(x, y) \leq c(x, y), \qquad \forall (x, y) \in A. \tag{2}$$

We refer to v as the *value* of the flow f. A flow whose value is maximum is called a *maximum flow*.

A *preflow* is a function $f : A \to \mathbb{R}^+$ satisfying relations (2) and the next conditions:

$$f(x, N) - f(N, x) \geq 0, \qquad \forall x \in N \backslash \{s, t\}. \tag{3}$$

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

$$e(x) = f(x, N) - f(N, x)$$

Thus, for any preflow f, we have $e(x) \geq 0, \forall x \in N \backslash \{s,t\}$. We say that a node $x \in N \backslash \{s,t\}$ is active if $e(x) > 0$ and balanced if $e(x) = 0$. A preflow f for which $e(x) = 0, \forall x \in N \backslash \{s,t\}$ is a flow. Consequently, a flow is a particular case of preflow.

Let f be a flow from the source node s to the sink node t in the directed network $G = (N, A, c, s, t)$. The *residual capacity* of the arc (x,y) corresponding to the flow f is defined as $r(x,y) = c(x,y) - f(x,y) + f(y,x)$ and it is the maximum amount of additional flow that can be sent from x to y using both arcs (x,y) and (y,x). By convention, if an arbitrary arc $(x,y) \notin A$, then we can add (x,y) to A and we will consider that $c(x,y) = 0$.

The *residual network* $G(f) = (N, A(f))$ corresponding to flow f contains all those arcs with strictly positive residual capacity.

A network $G = (N, A)$ is called *bipartite* if its node set N can be partitioned into two subsets N_1 and N_2, such that all arcs have one endpoint in N_1 and the other in N_2.

A network $G = (N, A)$ is called *semi-bipartite* if its node set N can be partitioned into two subsets N_1 and N_2, such that no arc has both its endpoints in N_2. Thus, a semi-bipartite network can contain arcs having both their endpoints in N_1. Consequently, the notion of semi-bipartite network is less restrictive than the notion of bipartite network.

We consider a semi-bipartite capacitated network $G = (N, A, c, s, t)$. We distinguish two special nodes in the network G: a source node s and a sink node t. We assume without loss of generality that $s \in N_2$. If $s \in N_1$, then we could create a new source node $s\prime \in N_2$ and add a new arc $(s\prime, s)$ with sufficiently large capacity.

Let $n = |N|$, $n_1 = |N_1|$, $n_2 = |N_2|$, $m = |A|$ and $C = \max\{c(i,j) | (i,j) \in A\}$.

In the residual network Gf, the *distance function* $d : N \rightarrow \mathcal{N}$ 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 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.

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*.

Let $G = (N, A, c, s, t)$ be a semi-bipartite directed network, $N = N_1 \cup N_2$. Any path in the network G or in the residual network $G(f)$, that is also a semi-bipartite network, can have at most $2n_1$ arcs. Consequently, if we set $d(s) = 2n_1 + 1$ then the residual network will never contain an admissible directed path from the source node s to the sink node t.

Lemma 1. *[1] In the bipartite directed network $G = (N, A, c, s, t)$, for any node $i \in N$, $d(i) < 4n_1 + 1$.*

A straight consequence of this lemma is the following:

Lemma 2. *In the semi-bipartite directed network $G = (N, A, c, s, t)$, for any node $i \in N$, $d(i) < 4n_1 + 1$.*

For determining a maximum flow in regular networks, several algorithms were developed in the last decades. These algorithms can be divided into two classes:

1. augmenting path algorithms

2. preflow algorithms.

Algorithms from both classes can be modified in order to determine a maximum flow in a semi-bipartite network by using the particularities of this type of network. In the next section we will develop a generic preflow algorithm for determining a maximum flow in semi-bipartite networks. This algorithm is obtained from the generic preflow algorithm for maximum flow in bipartite networks (see [1]). Our algorithm allows only the nodes in N_1 to become active. In order to do this, it pushes flow on individual admissible arcs or along paths consisting of two admissible arcs. The generic preflow algorithm for maximum flow in semi-bipartite networks runs in $O(n_1^2 m)$ time.

3 The generic preflow algorithm for maximum flow in semi-bipartite networks

The generic preflow algorithm for maximum flow in semi-bipartite networks is based on the same idea as the generic preflow algorithm for maximum flow in regular networks. This means that it creates excesses at intermediate nodes during its execution. An important feature of this algorithm is that it allows only nodes in N_1 to become active, obtaining in this way a better running time than the generic preflow algorithm for maximum flow in regular networks.

Because the generic preflow algorithm for maximum flow in semi-bipartite networks is a generic algorithm, its basic step consists of selecting, *without a specified rule*, an active node and trying to eliminate its excess by pushing flow to its neighbors which are closer to the sink node and which are also contained in N_1. In order to do this it pushes the flow on individual admissible arcs or along paths consisting of two admissible arcs. Like any preflow algorithm, it ends when all the intermediate nodes have no excess, which means that a flow was obtained. Moreover, this is a maximum flow.

The generic preflow algorithm for the maximum flow problem in semi-bipartite networks is the following:

```
Generic Preflow Algorithm;
Begin
    let $f = 0$;
    determine the residual network $G(f)$;
    compute the exact distance labels $d$ in the residual network $G(f)$;
    for each arc $(s, i) \in A$ do
        $f(s, i) = c(s, i)$;
    $d(s) = 2n_1 + 1$;
    while the residual network $G(f)$ contains an active node do
     begin
        select an active node $i$;
        push/relabel($i$);
    end
end.

procedure push/relabel($i$);
begin
    if there is an admissible arc $(i, j)$ in the residual network then
        if  $j \in N_1$  then
            push $g$ =min$\{e(i), r(i, j)\}$ units of flow on the arc $(i, j)$;
        else
            if there is an admissible arc $(j, k)$ in the residual network then
            push $g$ =min$\{e(i), r(i, j), r(k, j)\}$ units of flow along the path
            $i - j - k$;
            else $d(j)$ =min$\{d(k)|(j, k) \in A(f)\} + 1$;
    else  $d(i)$ =min$\{d(j)|(i, j) \in A(f)\} + 1$;
end;
```

Theorem 1. *(Correctness theorem) The generic preflow algorithm computes correctly a maximum flow in the semi-bipartite network G = (N, A, c, s, t).*

Proof. The correctness of the generic preflow algorithm for maximum flow in semi-bipartite networks is a straight consequence of the correctness of the generic preflow algorithm for maximum flow in regular networks, proved in [1]. □

Theorem 2. *The generic preflow algorithm determines a maximum flow in a semi-bipartite network in $O(n_1^2 m)$ time.*

Proof. The algorithm performs three types of operations: saturating pushes, nonsaturating pushes and node relabelings. As in the generic algorithm for maximum flow in regular networks, the bottleneck operations are the nonsaturating pushes. To determine an upper bound of the number of nonsaturating pushes performed by the algorithm, we will use the potential function $\Phi = \sum_{i \in I} d(i)$, where I is the set of active nodes. The initial value of Φ is at most $4n_1^2$, because the algorithm allows only nodes in N_1 to be active and $d(i) \leq 4n_1$ for all $i \in N_1$, from Lemma 2. During the execution of a push/relabel procedure, one of the following 5 cases might appear:

1. The algorithm increases the distance label of node $i \in N_1$. In this case Φ increases, but the total increase in Φ caused by all these relabeling operations is, considering Lemma 2, $O(n_1^2)$.

2. The algorithm increases the distance label of node $j \in N_2$. In this case Φ doesn't change because the nodes in N_2 are never active.

3. The algorithm pushes flow on arc (i, j) saturating this arc. In this case Φ increases by at most $4n_1$ because a new node, j, from N_1 might become active. But the total increase in Φ caused by all these operations is $O(n_1^2 m)$, considering Lemma 2 and the fact that the total number of saturating pushes is $O(n_1 m)$.

4. The algorithm pushes flow along the path $i - j - k$ saturating one of its arcs. In this case Φ increases by at most $4n_1$ because a new node, k, from N_1 might become active. Lemma 2 and the fact that the total number of saturating pushes is $O(n_1 m)$ imply that the total increase in Φ caused by all these operations is $O(n_1^2 m)$.

5. The algorithm performs a nonsaturating push. In this case the value of Φ decreases by at least 1, because node i becomes inactive, but a new node in N_1 or N_2 might become active.

Because at the end of algorithm Φ=0, we have that the algorithm performs $O(n_1^2 m)$ nonsaturating pushes. Consequently, it runs in $O(n_1^2 m)$ time. $\square$

References

[1] Ahuja, R., Magnanti, T., Orlin, J., *Network Flow. Theory, Algorithms and Applications*, Prentice Hall, New Jersey, 1993.

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

[3] Ciupală, L., *Wave Algorithm for Maximum Flow in Bipartite Networks*, Bulletin of the *Transilvania* University of Braşov **5(54)** (2013), 133-138.