Bulletin of the *Transilvania* University of Braşov • Vol 2(51) - 2009
Series III: Mathematics, Informatics, Physics, 241-248

THE COST SCALING ALGORITHM FOR BIPARTITE NETWORKS

Laura CIUPALĂ[1]

Abstract

In this paper, we describe the cost scaling algorithm for the minimum cost flow in bipartite networks. The cost scaling algorithm for minimum cost flow problem in general networks, developed by Goldberg and Tarjan, is based on ϵ-optimality conditions.

The basic idea behind the cost scaling algorithm for the minimum cost flow in a bipartite network is to perform bipushes from nodes in N_1. A bipush is a push over two consecutive admissible arcs. Consequently, a bipush moves the excess from a node in N_1 to another node in N_1. This approach has all the advantages of the scaling cost algorithm for the minimum cost flow in general networks. Moreover, it has an additional advantage that leads to an improved running time. This additional advantage consists in the fact that, using bipushes instead of pushes, all the nodes in N_2 are maintained balanced. Consequently, the running time of the cost scaling algorithm for the minimum cost flow is reduced from $O(n^2 m \log(nB))$ to $O(n_1^2 m \log(nB))$ when it is applied on bipartite networks.

2000 *Mathematics Subject Classification:* 90B10, 90C90.
Key words: network flow, minimum cost flow, bipartite network.

1 The minimum cost flow problem

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.

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 capacity $c(x, y)$ and a cost $b(x, y)$. We associate with each node $x \in N$ a number $v(x)$ which indicates its supply or demand depending on whether $v(x) > 0$ or $v(x) < 0$. In the directed network $G = (N, A, c, b, v)$, the minimum cost flow problem is to determine the flow $f(x, y)$ on each arc $(x, y) \in A$ which

$$minimize \sum_{(x,y)\in A} b(x, y)f(x, y) \tag{1}$$

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

subject to

$$\sum_{y|(x,y)\in A} f(x,y) - \sum_{y|(y,x)\in A} f(y,x) = v(x), \qquad \forall x \in N \tag{2}$$

$$0 \le f(x,y) \le c(x,y), \qquad \forall (x,y) \in A. \tag{3}$$

A flow f satisfying the last two conditions is a feasible flow.

Let $n = |N|$ and $m = |A|$.

Let C denote the largest magnitude of any supply/demand or finite arc capacity, that is

$$C = \max(\max\{v(x)|x \in N\}, \max\{c(x,y)|(x,y) \in A, c(x,y) < \infty\}).$$

Let B denote the largest magnitude of any arc cost, that is

$$B = \max\{b(x,y)|(x,y) \in A\}.$$

The residual network $G(f) = (N, A(f))$ corresponding to a flow f is defined as follows. We replace each arc $(x,y) \in A$ by two arcs (x,y) and (y,x). The arc (x,y) has cost $b(x,y)$ and residual capacity $r(x,y) = c(x,y) - f(x,y)$ and the arc (y,x) has cost $b(y,x) = -b(x,y)$ and residual capacity $r(y,x) = f(x,y)$. The residual network consists only of arcs with positive residual capacity.

We shall assume that the minimum cost flow problem satisfies the following assumptions:

1. All data (cost, supply/demand and capacity) are integral.
2. The network contains no directed negative cost cycle of infinite capacity.
3. All arc costs are nonnegative.
4. The supplies/demands at the nodes satisfy the condition $\sum_{x\in N} v(x) = 0$ and the minimum cost flow problem has a feasible solution.
5. The network contains an uncapacitated directed path (i.e. each arc in the path has infinite capacity) between every pair of nodes.

All these assumptions can be made without any loss of generality (for details see [1]).

We associate a real number $\pi(x)$ with each node $x \in N$. We refer to $\pi(x)$ as the potential of node x. For a given set of node potentials π, we define the reduced cost of an arc (x,y) as

$$b^{\pi}(x,y) = b(x,y) - \pi(x) + \pi(y).$$

The reduced costs are applicable to the residual network as well as to the original network.

Theorem 1. *[1](a) For any directed path P from node w to node z we have*
$\sum_{(x,y)\in P} b^{\pi}(x,y) = \sum_{(x,y)\in P} b(x,y) - \pi(w) + \pi(z)$
(b) For any directed cycle W we have
$\sum_{(x,y)\in W} b^{\pi}(x,y) = \sum_{(x,y)\in W} b(x,y)$

Theorem 2. *(Negative Cycle Optimality Conditions) [1] A feasible solution f is an optimal solution of the minimum cost flow problem if and only if it satisfies the following negative cycle optimality conditions:*
the residual network $G(f)$ contains no negative cost directed cycle.

Theorem 3. *(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:*
$b^{\pi}(x,y) \geq 0$ *for every arc (x,y) in $G(f)$*

Theorem 4. *(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 costs and flow values satisfy the following complementary slackness optimality conditions for every arc $(x,y) \in A$:*

$$\text{If } b^{\pi}(x,y) > 0, \text{ then } f(x,y) = 0 \tag{4}$$

$$\text{If } 0 < f(x,y) < c(x,y), \text{ then } b^{\pi}(x,y) = 0 \tag{5}$$

$$\text{If } b^{\pi}(x,y) < 0, \text{ then } f(x,y) = c(x,y) \tag{6}$$

A pseudoflow is a function $f : A \to \mathbb{R}^{+}$ satisfying only conditions (3). For any pseudoflow f, we define the imbalance of node x as
$e(x) = v(x) + f(N,x) - f(x,N)$, for all $x \in N$.

If $e(x) > 0$ for some node x, we refer to $e(x)$ as the excess of node x; if $e(x) < 0$, we refer to $-e(x)$ as the deficit of node x. If $e(x) = 0$ for some node x, we refer to node x 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.

We refer to a flow or a pseudoflow f as ϵ-optimal for some $\epsilon > 0$ if for some node potentials π , the pair (f,π) satisfies the following ϵ-optimality conditions:

$$\text{If } b^{\pi}(x,y) > \epsilon, \text{then } f(x,y) = 0 \tag{7}$$

$$\text{If } -\epsilon \leq b^{\pi}(x,y) \leq \epsilon, \text{then } 0 \leq f(x,y) \leq c(x,y) \tag{8}$$

$$\text{If } b^{\pi}(x,y) < -\epsilon, \text{then } f(x,y) = c(x,y) \tag{9}$$

These conditions are relaxations of the (exact) complementary slackness optimality conditions (4) - (6) and they reduce to complementary slackness optimality conditions when $\epsilon = 0$.

For solving a minimum cost flow problem based on these optimality conditions, several algorithms were developed from the primal-dual algorithm proposed by Ford and Fulkerson in 1962 to the polynomial-time cycle-canceling algorithms described by Sokkalingam, Ahuja and Orlin in 2001.

The basic algorithms for minimum cost flow can be divided into two classes: those that maintain feasible solutions and strive toward optimality and those that maintain infeasible solutions that satisfy optimality conditions and strive toward feasibility. Algorithms from the first class are: the cycle-canceling algorithm and the out-of-kilter algorithm. The cycle-canceling algorithm maintains a feasible flow at every iteration, augments flow along negative cycle in the residual network and terminates when there is no more negative cycle in the residual network, which means (from Theorem 2) that the flow is a minimum cost flow. The out-of-kilter algorithm maintains a feasible flow at every iteration and augments flow along shortest path in order to satisfy the optimality conditions. Algorithms from the second class are: the successive shortest path algorithm and primal-dual algorithm. The successive shortest path algorithm maintains a pseudoflow that satisfies the optimality conditions and augments flow along shortest path from excess nodes to deficit nodes in the residual network in order to convert the pseudoflow into an optimal flow. The primal-dual algorithm also maintains a pseudoflow that satisfies the optimality conditions and solves maximum flow problems in order to convert the pseudoflow into an optimal flow.

Starting from the basic algorithms for minimum cost flow, several polynomial-time algorithms were developed. Most of them were obtained by using the scaling technique. By capacity scaling, by cost scaling or by capacity and cost scaling, the following polynomial-time algorithms were developed: capacity scaling algorithm, cost scaling algorithm, double scaling algorithm, repeated capacity scaling algorithm and enhanced capacity scaling algorithm.

Another approach for obtaining polynomial-time algorithms is to select carefully the negative cycles in the cycle-canceling algorithm.

2 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, $\epsilon = B$ 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 $\epsilon/2$-optimal flow. After $1 + [log(nB)]$ cost scaling phases, $\epsilon < 1/n$ and the algorithm terminates with an optimal flow. The cost scaling algorithm is the following:

Cost Scaling Algorithm;
Begin
$\pi = 0$;

```
  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(x, y) ∈ A do
    if b^π(x, y) > 0 then
      f(x, y) = 0;
    else if b^π(x, y) < 0 then
      f(x, y) := c(x, y);
  compute nodes imbalances;
  while the network contains an active node do
  begin
    select an active node x;
    push/relabel(x);
  end;
end;

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

We refer to a push of g units of flow on the admissible arc (x, y) as saturating if it saturates the arc (x, y); otherwise it is a nonsaturating push. The improve-approximation procedure transforms an ϵ-optimal flow into an $\epsilon/2$-optimal flow. This transformation consists in converting an ϵ-optimal flow into an $\epsilon/2$-optimal pseudoflow and then gradually converting the pseudoflow into a flow while always maintaining $\epsilon/2$-optimality of the solution. We refer to a node x with $e(x) > 0$ as an active node and say that an arc (x, y) in the residual network is admissible if $-\epsilon/2 \leq b^{\pi}(x, y) < 0$. 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 x and to perform pushes on admissible arcs (x, y) emanating from node x. When the network contains no admissible arc, the algorithm updates the node potential $\pi(x)$ in order to create new admissible arcs emanating from node x.

To identify admissible arcs emanating from node x, we use the following data structure: for each node x, we maintain a current-arc (x, y) which is the current candidate to test for admissibility. Initially, the current-arc of node x is the first arc in its arc list $A(x)$. To determine an admissible arc emanating from node x, 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 x and again sets its current-arc to the first arc in the arc list $A(x)$.

Theorem 5. *[1] The cost scaling algorithm solves correctly the minimum cost flow problem in* $O(n^2 m \log(nB))$ *time.*

3 The cost scaling algorithm for the minimum cost flow problem in bipartite networks

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

We consider a bipartite capacitated network $G = (N_1, N_2, A, c, b)$ with a nonnegative capacity $c(x, y)$ and with a cost $b(x, y)$ associated with each arc $(x, y) \in A$.

Let $n_1 = |N_1|$ and $n_2 = |N_2|$.

When applied on bipartite networks, the cost scaling algorithm can be improved by imposing the rule that we push flow on two adjacent arcs in order to maintain all the nodes in N_2 balanced, only nodes in N_1 could have excesses. We refer to these operations as bipushes. The replacement of the push/relabel procedure with the bipush/relabel procedure described below ensures that only nodes in N_1 can have excesses. Consequently, the running time of the cost scaling algorithm for the minimum cost flow is reduced from $O(n^2 m \log(nB))$ to $O(n_1^2 m \log(nB))$ when it is applied on bipartite networks.

The basic idea behind the cost scaling algorithm for the minimum cost flow in a bipartite network is to perform bipushes from nodes in N_1. A bipush is a push over two consecutive admissible arcs. Consequently, a bipush moves the excess from a node in N_1 to another node in N_1. This approach has all the advantages of the scaling cost algorithm for the minimum cost flow in regular networks. Moreover, it has an additional advantage that leads to an improved running time. This additional advantage consists of the fact that, using bipushes instead of pushes, all the nodes in N_2 are maintained balanced. We refer to a bipush along the path $x - y - z$ as saturating if after it at least one of the arcs (x, y) and (y, z) is dropped from the residual network; otherwise the bipush is nonsaturating. Obviously, after a nonsaturating bipush along the path $x - y - z$, the excess of the node x becomes 0. Since all the excesses are at the nodes in N_1, it is sufficient to account for the nonsaturating bipushes from the nodes in N_1. Since $|N_1| < |N|$, the number of nonsaturating bipushes is reduced.

For determining a minimum cost flow in a bipartite network, we can use the cost scaling algorithm modified by replacing the procedure push/relabel with the procedure

bipush/relabel, which is described below.

procedure bipush/relabel(x);
begin
 if the residual network contains an admissible (x, y) **then**
  if the residual network contains an admissible (y, z) **then**
   push $g = \min(e(x), r(x, y), r(y, z))$ units of flow from node x to node z;
  else
   $\pi(y) := \pi(y) + \epsilon/2$;
 else
  $\pi(x) := \pi(x) + \epsilon/2$;
end;

Theorem 6. *The cost scaling algorithm for the minimum cost flow problem in a bipartite network runs in* $O(n_1^2 m \log(nB))$ *time.*

Proof. The proof of Theorem 5 shows that the time-complexity of the cost scaling algorithm for the minimum cost flow problem in general networks is $O(n^2 m \log(nB))$ in the following manner:

1. the improve-approximation procedure is called $1 + \lceil \log(nB) \rceil$ times

2. the complexity of the improve-approximation procedure is $O(n^2 m)$ because it requires $O(n^2 m)$ time to perform nonsaturating pushes and $O(nm)$ time to perform saturating pushes.

Because the cost scaling algorithm for the minimum cost flow in bipartite networks pushes flow on two adjacent arcs, it maintains all the nodes in N_2 balanced, only nodes in N_1 could have excesses. Consequently, the number of nonsaturating bipushes is $O(n_1^2 m)$ and the number of saturating bipushes is $O(n_1 m)$ in an improve-approximation procedure execution. Since the procedure is called $1 + \lceil \log(nB) \rceil$ times, it follows the the cost scaling algorithm for the minimum cost flow in bipartite networks runs in $O(n_1^2 m \log(nB))$ time. □

4 Conclusions

In the improve-approximation procedure from the cost scaling algorithm, we replaced a push a flow on a single admissible arc with a push on two consecutive admissible arcs. In this manner, we obtained the cost scaling algorithm for the minimum cost flow in bipartite networks. This algorithm maintains all the nodes in N_2 balanced, only nodes in N_1 could have excesses, because each bipush moves the excess from a node in N_1 to another node in N_1. Consequently, the running time of the modified cost scaling algorithm for the minimum cost flow in bipartite networks is $O(n_1^2 m \log(nB))$.

References

[1] Ahuja, R., Magnanti, T.,Orlin, J., *Network Flow. Theory, Algorithms and Applications*, Prentice Hall, New Jersey, 1999.

[2] Ciupală, L., *A Scaling out-of-Kilter Algorithm for Minimum Cost Flow*, Control and Cybernetics **34** (2005), 1169-1174.

[3] Ciupală, L., Ciurea, E., *Sequential and parallel deficit scaling algorithms for minimum flow in bipartite networks*, WSEAS Transactions on Computer Research **7** (2008), 1545-1554.