Incremental Algorithms for the Minimum Cost Flow Problem

LAURA CIUPALĂ
Department of Computer Science
University Transilvania of Braşov
Iuliu Maniu Street 50, Braşov
ROMANIA
laura_ciupala@yahoo.com

Abstract: - Incremental algorithms may save computational time to solve different network flow problems. Let us consider a network in which we already established a minimum cost flow. We describe and solve the problem of establishing a minimum cost flow in this network after inserting a new arc and after deleting an existent arc. We focus on these problems because they arise in practice.

Key-Words: - Network flow; Network algorithms; Minimum cost flow problem; Incremental computation.

1 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 60 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.

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.

2 Minimum Cost Flow Problem

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

$$\text{minimize} \sum_{(i,j)\in A} b(i,j) f(i,j) \tag{1}$$

subject to

$$\sum_{j|(i,j)\in A} f(i,j) - \sum_{j|(j,i)\in A} f(j,i) = v(i), \forall i \in N \tag{2}$$

$$0 \le f(i,j) \le c(i,j), \forall (i,j) \in A. \tag{3}$$

where $\sum_{i\in N} v(i) = 0$.

A flow f satisfying the conditions (2) and (3) 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

$$B = \max\{b(i,j) \mid (i,j)\in A\}.$$

2.1 Solving Minimum Cost Flow Problem

For solving a minimum cost flow problem, 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. Most of these algorithms work on the residual network.

So, before describing them, we have to introduce some notions and some assumptions.

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 them 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.

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 1. ([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 2. (Negative Cycle Optimality Conditions) ([1]) A feasible solution f is an optimal solution of the minimum cost flow problem if and only if there is no negative cycle in the residual network $G(f)$.

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:

$$b^{\pi}(i, j) \geq 0 \text{ for every arc } (i, j) \text{ in the residual network } G(f).$$

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 cost and flow values satisfy the following complementary slackness optimality conditions for every arc $(i, j)\in A$:

$$\text{If } b^{\pi}(i, j) > 0, \text{ then } f(i, j) = 0 \quad (4)$$
$$\text{If } 0 < f(i, j) < c(i, j), \text{ then } b^{\pi}(i, j) = 0 \quad (5)$$
$$\text{If } b^{\pi}(i, j) < 0, \text{ then } f(i, j) = c(i, j) \quad (6)$$

A *pseudoflow* is a function $f : A \rightarrow \mathbf{R}_{+}$ satisfying the only conditions (3).

For any pseudoflow f, we define the *imbalance* of node i as

$$e(i) = v(i) + f(N, i) - f(i, N), \qquad \text{for all } i\in N.$$

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:

$$b^{\pi}(i,j) \geq 0 \text{ for every arc } (i,j) \text{ in the residual network } G(f^*).$$

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*:

$$\text{If } b^{\pi}(i,j) > \varepsilon, \text{ then } f(i,j) = 0 \quad (7)$$
$$\text{If } -\varepsilon \leq b^{\pi}(i,j) \leq \varepsilon, \text{ then } 0 \leq f(i,j) \leq c(i,j) \quad (8)$$
$$\text{If } b^{\pi}(i,j) < -\varepsilon, \text{ then } f(i,j) = c(i,j) \quad (9)$$

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

The algorithms for determining a minimum cost flow rely upon the optimality conditions stated by Theorems 2, 3 and 4.

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.

3 Incremental Algorithms for the Minimum Cost Flow Problem

In this section we describe incremental algorithms which update the solution of a minimum cost flow problem after inserting a new arc and after deleting an arc.

3.1 Inserting a new arc

Let $G=(N, A, c, b, v)$ be a network in which we already determined a minimum cost flow f^*. Let us insert a new arc (k, l) with capacity $c(k, l)$ and cost $b(k, l)$ into the network G, obtaining in this way the network $G' = (N, A', c', b', v)$, where:

$$A' = A \cup \{(k, l)\}$$
$$c'(i,j) = c(i,j), \forall (i\,j) \in A'$$
$$b'(i,j) = b(i,j), \forall (i\,j) \in A'$$

The new network G' can contain a minimum cost flow $f^{*}{}'$ with a smaller cost than f^* – the minimum cost flow in G.

In the network G' obtained by inserting a new arc (k, l) in G, it is possible that the optimality conditions are not fullfilled with respect to f^* which was a minimum cost flow in G. It is possible that the residual network $G'(f^*)$ contains a negative cycle, which means that the Negative cycle optimal conditions (from Theorem 2) are not satisfied. The incremental algorithm for determining a minimum cost flow in a network after inserting a new arc is based on these optimality conditions:

```
**Incremental Add Algorithm;**
**Begin**
    let $f$ be a minimum cost flow in the network $G$;
    determine the new network $G'$ obtained from $G$ by inserting a new arc $(k, l)$;
    **while** the residual network $G'(f)$ contains a negative cycle **do begin**
        determine a negative cycle $C$ in $G'(f)$;
        compute $r(C) = \min\{r(i,j) \mid (i,j) \in C\}$;
        send $r(C)$ units of flow along the cycle $C$;
        update the residual network $G'(f)$;
    end
end.
```

Theorem 5 *The incremental add algorithm computes correctly a minimum cost flow in the network G',* obtained from G by inserting a new arc(k, l).

Proof. The algorithm terminates when the residual network $G'(f)$ does not contain any negative cycles. By Theorem 2, it follows that the flow f is a minimum cost flow.

Theorem 6 *The incremental add algorithm runs in $O(nmc(k, l))$ time.*

Proof. Because f , the flow with which the algorithm starts, is a minimum cost flow in the network G, it follows that the new network G' obtained from G by inserting a new arc (k, l) could contain only negative cycles that contain the arc (k, l). At each iteration of the **while** loop, such a cycle C is determined in $O(nm)$ time. Sending $r(C)$ units of flow along the cycle C means reducing its residual capacity by $r(C)$. Accordingly to Assumption 2, all data are integer. It follows that $r(C)$ is also an integer number. Consequently, $r(C) \geq 1$. Thus, after at most $c(k, l)$ iterations, the network will contain no negative cycle. This implies that in $O(nmc(k, l))$ time the algorithm determines a minimum cost flow.

3.2 Deleting an existent arc

Let $G=(N, A, c, b, v)$ be a network in which we already determined a minimum cost flow f^*. Let (k, l) be an arbitrary arc of G. The arc (k, l) will be removed from the network G, obtaining in this way the network $G'= (N, A', c', b', v)$, where:

$$A' = A \setminus \{(k, l)\}$$
$$c'(i,j)= c(i,j),\ \forall(i\, j) \in A'$$
$$b'(i,j)= b(i,j),\ \forall(i\, j) \in A'$$

Let f^* be the minimum cost flow in G and let $f^{*}{}'$ be the minimum cost flow in the network G' obtained from G through deletion of the arc (k, l). The cost of the flow $f^{*}{}'$ might by greater than the cost of f^*.

In the network G' obtained by deleting the arc (k, l) from G, it is possible that the optimality conditions are not fullfilled with respect to f^* which was a minimum cost flow in G. More precisely, if in G the flow on the arc (k, l) was strictly positive, then in G' f^* will no longer satisfy the mass balance constraints (2) for both of the nodes k and l. This means that, in G', f^* is not a flow, but a pseudoflow. We will transform this pseudoflow into a minimum cost flow, by sending flow from the node k to the node l in the network G' using the following algorithm.

```
Incremental Delete Algorithm;
Begin
    let $f$ be a minimum cost flow in the network $G$;
    compute a set of optimal node potentials $\pi$ with respect to the minimum cost flow $f$;
    $e(k) = f(k, l)$;
    $e(l) = -f(k, l)$;
    determine the new network $G'$ obtained from $G$ by deleting the arc $(k, l)$;
    determine the residual network $G'(f)$;
    while $e(k) > 0$ do begin
        determine shortest path distances $d$ from $k$ to all other nodes in the residual network $G'(f)$ with respect to the reduced costs;
        let $P$ be a shortest path from $k$ to $l$;
        $\pi = \pi - d$;
        $r(P) = \min(e(k), \min\{r(i,j) \mid (i,j) \in P\})$;
        send $r(P)$ units of flow along the path $P$;
        update the residual network $G'(f)$ and the reduced costs;
    end
end.
```

Theorem 7 *The incremental delete algorithm computes correctly a minimum cost flow in the network G'.* obtained from G by deleting the arc(k, l).

Proof. The algorithm terminates when $e(k) = 0$, which means that k is a balanced node. Because all nodes excepting k and l are balanced at the beginning of the algorithm and remain balanced during the algorithm, it follows that at the end of the algotrithm also l is a balanced node. This implies that f is a flow. By Theorem 3, it follows that the flow f is a minimum cost flow.

Theorem 8 *The incremental add algorithm runs in $O(S(n, m)+f(k, l)S'(n, m))$ time, where $S(n, m)$ is the time needed to solve a shortest path problem with possible negative arc lengths and $S'(n, m)$ is the time needed to solve a shortest path problem with nonnegative arc lengths.*

Proof. For determining a set of optimal potentials, a shortest path algorithm is applied in the residual network $G(f)$. This network contains no negative cycle because f is a minimum cost flow in the network G, which must satisfy the negative cycle optimality conditions from Thorem 2. But it is not mandatory that all lengths in the residual network $G(f)$ to be nonnegative. Thus, the time complexity for determining a set of optimal potentials is $S(n, m)$.

Because f is a flow, all the nodes in G are balanced. After deleting the arc (k, l), if the flow on this arc was strictly positive, we have an excess of $f(k, l)$ units at node k and a deficit of $f(k, l)$ units at node l.

At each iteration of the **while** loop, we solve a shortest path problem from node k to all the other nodes in the residual network $G'(f)$ with respect to the reduced costs, which are nonnegative. Sending $r(P)$ units of flow along the shortest path P from k to l means reducing the excess of k by $r(P)$. Accordingly to Assumption 2, all data are integer. It follows that $r(P)$ is also an integer number. Consequently, $r(P) \geq 1$. Thus, after at most $f(k, l)$ iterations, the excess of the node k will be reduced to 0, which implies that the deficits of the node l will be also 0 and all the nodes will be balanced. Equivalently, f will be a flow. Moreover, it will be a minimum cost flow. Consequently, the **while** loop will take $O(f(k, l)S'(n, m))$ time and the algorithm will run in $O(S(n, m)+f(k, l)S'(n, m))$ time.

4 Conclusion

In this paper, we studied a problem that arises in practice in several applications. We consider that we need to modify a network in which a minimum cost flow is already determined. A modification of the network means that a new arc is added or an exiting arc is removed. After such a modification of the network, we need to reestablish a minimum cost flow. For solving these problems, incremental algorithms are used because they save computational time.

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] L. Ciupală, About Minimum Cost Flow Problem in Networks with Node Capacities, *Proceedings of the 13th WSEAS International Conference on Computers*, 2009, pp. 67-70

[4] L. Ciupală, A scaling out-of-kilter algorithm for minimum cost flow, *Control and Cybernetics* Vol.34, No.4, 2005, pp. 1169-1174.

[5] P.T. Sokkalingam, R. Ahuja and J.Orlin, New Poynomial-Time Cycle-Canceling Algorithms for Minimum Cost Flows, http://web.mit.edu/jorlin/www/papers.html, 2001

[6] K.D. Wayne, *A polynomial Combinatorial Algorithm for Generalized Minimum Cost Flow*, Math. Oper. Res., 445-459, 2002.