Journal of Advanced College of Engineering and Management, Vol. 6, 2021

MAXIMUM NETWORK FLOW ALGORITHMS

Mohan Chandra Adhikari[1], Urmila Pyakurel[2]

[1, 2]Central Department of Mathematics, IOST, Tribhuvan University, Kathmandu, Nepal

Abstract

The aim of the maximum network flow problem is to push as much flow as possible between two special vertices, the source and the sink satisfying the capacity constraints. For the solution of the maximum flow problem, there exists a number of algorithms. The existing algorithms can be divided into two families. First, augmenting path algorithms that satisfy the conservation constraints at intermediate vertices and the second preflow push relabel algorithms that violates the conservation constraints at the intermediate vertices resulting incoming flow more than outgoing flow.In this paper, we study different algorithms that determine the maximum flow in the static and dynamic networks.

Keywords: *Maximum flow, network flow, augmenting path algorithm, push relabel algorithm*

1. Introduction

To solve a maximum flow problem, we have to determine the maximum amount of flow that can be sent from the source to the sink in a flow network.The fundamental algorithms that solvethe maximum flow problems are used to solve various real life problems in the network flow theory as presented by Ahuja et al.[1] and Dhamala et al. [5]. These algorithms are classified into two groups. Augmenting path algorithms based on Ford and Fulkerson [8] and preflow push relabel algorithms based on Goldberg and Tarjan [10].

In a flow network, augmenting path algorithm is a two-step procedure: finding$s-t$ paths and then augment flow along such paths. If augmenting paths are chosen arbitrarily, algorithm runs in $O(mnU)$ time complexity which is not a polynomial. If flow is augmentedalong the shortest path (in terms of number of arcs) and maximum capacity path,the algorithm runs in polynomial time$O(nm^2)$ and $O(m^2\log(U))$ [7]. In Dinic layer graph [6], the algorithm has time complexity$O(mn^2)$.Moreover, if the total capacity of the arcs leaving the source is greater than the minimum cut capacity of the network having intermediate nodes with capacities at least equal to the incoming arcs, then the maximum flow with intermediate storage is solved in polynomial time complexity by Pyakurel and Dempe [14], Pyakurel and Adhikari [13] and Pyakurel et al. [15].

Preflow push relabel algorithm of Goldberg and Tarjan [10] uses the concept of preflows developed by Karzanoz [11]. The algorithm violates the conservation constraints at the intermediate vertices as it saturates all the arcs outgoing from the source. The algorithm performs two basic operations: push and relabel until there exist active vertices in the residual network. Number of non-saturating push operationsdetermine the complexity. Push relabel algorithm has the time complexity$O(mn^2)$. Further,it is reduced to $O(n^3)$ by examining the active vertices in the first in first out order. Authors in [3] and [4] have obtained $O(n^2\sqrt{m})$ time bound with highest label active vertex selection rule.

The paper is organized as follows: Basic network ideas, flow models and some definitions are given in Section 2. Different static maximum flow algorithms andtheir efficiency criteria is discussed in Section 3. Section 4 discusses about time expanded graph and dynamic flow. The paper is concluded in Section 5.

2. Preliminaries

The procedure of transshipment of flow from the fixed location (source) $s$to the fixed destination (sink) t through the intermediate locations can be represented by a network. Let V be the set of n vertices (locations), and A be the set of m arcs then $e = (i,j) \in A$ is the transshipment route that joins any two locations $i \in V$ and $j \in V$.The set of incoming and the outgoing arcs for any vertex i are represented as $A(i) = \{e \in A: e = (j,i), j \in V\}$ and $B(i) = \{e \in A: e = (i,j), j \in V\}$. In general, no arcs enter the source and exit from the sink so $A(i) = \emptyset$ and $B(i) = \emptyset$. The function $u: A \to Z^+$ determines the maximum amount of flow that can be transshipped from the vertex i to vertex j using the arc (i,j) and denoted by $u(i,j)$. With these parameters the network is $N = (V, A, s, t, u)$.

An s-t flow is a non - negative real valued function $f: A \to R^+$such that it satisfies:

a. Capacity constraints: $f(i,j) \leq u(i,j)$ for all $(i,j) \in A$
b. Antisymmetry: $f(i,j) = -f(j,i)$ for all $(i,j) \in A$
c. Conservation constraints: Total amount of flow incoming to a vertex $i \in V$ is equal to the total amount of flow outgoing from $i \in V$

$$\sum_{A(i)} f(j,i) = \sum_{B(i)} f(i,j).$$

Value of the flow$f$ is

$$|f| = \sum_{B(s)} f(s,j) = \sum_{A(i)} f(i,d).$$

Preflows are like flows where conservation constraints are not satisfied. As a result some intermediate vertices will have incoming flows more than the outgoing flows. The difference is called an excess and the vertex is an active vertex. The excess is defined as

$$e(i) = \sum_{A(i)} f(j,i) - \sum_{B(i)} f(i,j) > 0.$$

The residual network N_f of N with respect to the flow f contains the set of arcs$e = (i,j) \in A$ known as forward arcs for which $f(i,j) < u(i,j)$ and $e = (j,i) \in A$ known as backward arcs for which $f(i,j) > 0$. The residual capacity of the arc is defined as $u_f(i,j) = u(i,j) - f(i,j)$, and $u_f(j,i) = f(i,j)$. A flow f^* in N_f defines a flow f' in N such that $f'(i,j) = f(i,j) + f^*(i,j)$ and $f'(j,i) = f(j,i) - f^*(i,j)$. An arc $(i,j) \in A$ is said to be saturated if $u_f(i,j)$ is zero otherwise unsaturated.

An augmenting path is a path from the source to the sink. Let P be a simple $s-t$ path in N_f. The bottleneck capacity of a path is the minimum residual capacity of any arc on the path P. A cut is a partition of vertex set V into two sets S and T such that $S \cap T = \emptyset$, and $S \cup T = V$. An arc $(i,j) \in A$ is said to be forward arc if $i \in S$ and $t \in T$ and backward arc if $j \in S$ and $i \in T$. The cut is an $s-t$ cut if $s \in S$, $t \in T$ and its capacity is defined as

$$u[S,T] = \sum_{(i,j) \in [S,T]} u(i,j)$$

A distance function $d: N \to Z^+ \cup \{0\}$ with respect to the residual capacity $u_f(i,j)$ is said to be valid with respect to the flow f if

a. $d(t) = 0$

b. $d(i) \leq d(j) + 1$ for each $(i,j) \in N_f$. Here, $d(i)$ is the distance label of vertex i. If the distance labels are valid, $d(i)$ is a lower bound on the length of the shortest path from the vertex i to t in N_f and if $d(s) \geq n$ then there does not exist an $s-t$ path in N_f.

3. Maximum flow algorithms: In this section, we study two classes of the maximum flow algorithms: augmenting path and preflow push relabel. We begin with the first maximum flow algorithm that came to the existence due to Ford and Fulkerson [8]

Augmenting Path Algorithm

1. $f \leftarrow 0$
2. Determine an augmenting path Pin N_f
3. Augmentδ units flow along path P
4. Update $f: f(i,j) = \begin{cases} f(i,j) + \delta \; if \; (i,j) \in P \\ f(i,j) - \delta \; if \; (j,i) \in P \\ f(i,j) \quad otherwise \end{cases}$

   Where δ is the residual capacity in N_f

Augmenting path algorithm is based on the concept of augmenting paths, residual network and cut. In worst case, it requires$O(m)$ time to determine the paths. If such $s-t$ paths are not found in the residual network, algorithm terminates and the $s-t$flow is a maximum flow which is equal to the minimum cut capacity. If all arc capacities are integer values with an upper bound U, the capacity of an $s-t$ cut is at most $(n-1)U$. So complexity of the algorithm is $O(mnU)$ which is not a polynomial. Some polynomial time augmenting path algorithms are discussed below.

Capacity scaling [7]: We introduce a parameter $\Delta = 2^{\lfloor \log U \rfloor} \geq \frac{U}{2}$ and then augment flow along the path having residual capacity at least Δ. At the start of the Δ scaling phase, value of the maximum flow in the residual network N_f is $2m\Delta$. Initially, $\Delta = 2^{\lfloor \log U \rfloor}$ and at each Δ scaling phase divided by 2 until it is less than 1. Hence, there are at most $O(\log_2 U)$ scaling phase and $O(m)$ iterations per Δ scaling phase implying at most $O(m \log_2 U)$iterations.

Maximum capacity path[7]: Flow is augmented along the path with the maximum capacity. Let $f$be any flow and f^* be a maximum flow in a network N. Then, $f^* - f$is the maximum flow in the residual network N_f and residual capacity of the path is at least $\frac{1}{m}(f^* - f)$. We consider a sequence of $2m$ consecutive maximum capacity augmentations each starting with flow f. If each $2m$ augmentations carry at least $\frac{1}{2m}(f^* - f)$ units of flow, then we require at most $2m$ augmentations to obtain the maximum flow. If at least one of the $2m$ augmentations carry less than $\frac{1}{2m}(f^* - f)$ units flow then capacity of the maximum capacity augmenting path is reduced by a factor of 2. Since the maximum capacity is U and the minimum capacity is 1after at most $O(m \log_2 U)$ iterations maximum flow is obtained.

Shortest augmenting path [7]: Flow is augmented along the shortest path in terms of number of arcs. If an arc $(i,j) \in A$ is saturated then $d(i) = d(j) + 1$. In order to saturate this arc again, we need to have $u_f(i,j) > 0$ implying (j,i) lies on the shortest path. Let d' be the distance labeling in the later case then$d'(j) = d'(i) + 1$. But distance labels are non-decreasing. Therefore, $d'(j) = d'(i) + 1 \geq d(i) + 1 = d(j) + 2$ which shows that for the saturation of an arc $(i,j) \in A$ the distance label $d(i)$ should be increased at least 2 units. Any vertex $i \in V$ is at a distance of at most $n - 1$ units from the sink. Hence, an arc $(i,j) \in A$is saturated at most$O(n)$ times. The network has $m$arcs, so there are at most $O(mn)$ iterations.

Layered network [6]: The vertices are partitioned into layers $V_0, V_1, \dots, V_l$such that vertices with the same distance labels lie on the same layer. In each layer, shortest paths have the same length. Since saturated arcs are deleted the reverse arcs are not included. An augmenting path requires $O(n)$ time and there are at most m arcs to saturate. Hence, there are $O(mn)$ iterations.

Preflow push relabel algorithm [10]: Augmenting path algorithms might use the same arc repeatedly but preflow push relabel algorithm avoids this situation as all arcs outgoing from the source are saturated.Similar to augmenting path algorithms, preflow push algorithm also works on residual network and violates conservation constraints at the intermediate vertices so some vertices have incoming flow more than the outgoing flow. The algorithm repeatedly performs push and relabel operations. Firstly, the excess is pushed towards one of its adjacent vertex closer to the sink, and if t is not reachable the excess is pushed back towards the vertices that are closer to the source. Each push operation saturates at least one arc. If push operation is not applicable then the vertex is relabeled and its distance label increases progressively. Algorithms terminate when there are no active vertices changing preflow in to a flow and then a maximum flow.

Preflow push relabel algorithm

1. Determine exact distance label $d(i) \forall\ i \in V$
2. Set $f(s,i) = u(s,i)\ \forall\ (s,i) \in A(i)$
3. Set $d(s) = n$
4. While there exists active vertices
   a. Select an active vertex i
   b. If there exists an admissible arc (i,j)push, otherwise relabel i

Push along arc (i,j)

Applicability:i is active, $u_f(i,j) > 0$ and $d(i) = d(j) + 1$
Action: Push $\delta = \min\{e_f(i), u_f(i,j)\}$ units flow from i to j.

Relabel vertex i
Applicability:$i$is active, and $\forall j \in V, u_f(i,j) > 0 \Rightarrow d(i) \leq d(j)$

Action: relabel$d(i) = \min\{d(j) + 1 : u_f(i,j) > 0\}$

The push operation pushes $\delta = \min(e_f(i), u_f(i,j))$ units flow from i to j if $d(i) = d(j) + 1$. As a result$f(i,j)$ and $e_f(j)$ are increased by δ units whereas $f(j,i)$ and $e_f(i)$ are decreased by δ units. The

push operation is saturated if $\delta = u_f(i,j)$. If vertex i is still active and push operation is not applicable, then vertex i is relabeled by setting $d(i) = \min\{d(j) + 1 : (i,j) \in A_f\}$. For each vertex i, the distance label $d(i) < 2n - 1$. The source and sink are never relabeled. Hence, the total number of relabel operations is $(2n-1)(n-2) < 2n^2$.

The complexity of the algorithm is determined by the number of non-saturating pushes. Consider a potential function $\Phi = \sum d(i) : i \text{is active}$. If push operation along the arc (i,j) is non-saturating, then i is not an active. Total increase in Φ due to the relabeling operations is $(2n-1)(n-2)$.The non-saturating push operation decreases Φ at least 1 unit and saturating push operation increasesΦ at most $2n-1$ units. There are at most $2n-1$ saturating pushes per arc and m arcs to saturate. Hence, there are $(2n-1)m$ saturating pushes. During initialization and at the end of the algorithm$\Phi$is 0. Hence, total decrease and increase in Φ is at most $(2n-1)(2n-1)m + (2n-1)(n-2) < 4mn^2$. Hence, the algorithm has $O(mn^2)$ time complexity.

The bottleneck operation in the generic preflow push operation is the number of non-saturating pushes that is reduced by specifying the rule for the selection of active vertices. A nice feature of this algorithm is its further improvement. Proper selection of active vertices can lead to better time complexity. One approach is the selection of active vertices in first in first out order where vertices are examined in phase. To determine the number of phases we consider a potential function $\Phi = \max\{d(i) : i \text{ is active}\}$. During a phase, if the algorithm performs at least one relabel operation, then Φ increases at most$2n^2$. If the algorithm does not perform relabel operations, then Φ decreases by at least *1* unit. Each phase does not contain more than n vertices. Hence, the algorithm has $O(n^3)$ time complexity.Cheriyan and Maheshwari [3] obtained $O(n^2\sqrt{m})$ time bound with the highest label active vertex selection rule. They used the same arc repeatedly until it gets saturated. The proof is based on the concept of flow atom and flow path. Cheriyan and Mehlhorn[4] also obtained the time $O(n^2\sqrt{m})$using the potential function

$$\Phi = \sum_{i:i\text{isactive}} \frac{d'(i)}{k}, d'(i) = |\{j : d(j) \le d(i)\}|, \quad k = \sqrt{m}$$

Excess scaling algorithm of Ahuja and Orlin [2] pushes flow from an active vertex with sufficiently large excess to the vertices with sufficiently small excess with smallest distance label. A vertex i with $e(i) \ge \frac{\Delta}{2} \ge \frac{e_{\max}}{2}$ is considered with sufficiently large excess where Δ is the least integer which is a power of *2*. It pushes $\min\{e(i),\ u_f(i,j), \Delta - e(j)\}$units flow which is at least $\frac{\Delta}{2}$. Initially,we have $\Delta = 2^{\lceil \log_2 U \rceil}$ and during Δ scaling phase it is $\frac{\Delta}{2} \le e_{\max} \le \Delta$. When $e_{\max} \le \frac{\Delta}{2}$, a new phase begins. After $\lceil \log_2 U \rceil + 1$ phase,$e_{\max}$decreases to *0* and algorithm terminates withthe time complexity $O(mn + n^2 \log_2 U)$.

4. Dynamic Flows:

Flows that are associated with travel time along the arcs are dynamic. Let τ_e be the travel time along the arc. Then, the function $\tau_e : A \to Z^+$ determines the time required to travel flow along the arc $e = (i,j)$. Travel time along the arc means if a unit flow is at vertex i at time θ, then it reaches to the vertex j at time $\theta + \tau_e$. With the predetermined time T,within which total flow is to be sent,the dynamic network is $N = (V, A, u, s, t, \tau, T)$ where total time T is discretized as {*0,1,2, … ,T*}.

Time expanded network of Ford and Fulkerson [9] plays a vital role in the solution of the maximum dynamic flow problem. It contains one copy for each vertex in the dynamic network for each time step and defined as $N_T = (V_T, A_M \cup A_H)$. Maximum dynamic flow in N is equivalent to the maximum static flow in time expanded networkN_T. The set of vertices $V_T$and the set arcs$A_T$are defined as $V_T = \{i(\theta): i \in V,\ \theta = 0,1,2,\dots,T\}$ and $A_T = A_H \cup A_M$ whereA_H is the set of holdover arcs determined by$A_H = \{(i(\theta), j(\theta + 1)): i \in V,\ \theta = 0,1,\dots,T-1\}$ and A_M is the set of movement arcs determined by$A_M = \{(i(\theta), j(\theta + \tau_e)) : e \in A,\ \theta = 0,1,\dots T - \tau_e\}$.

If a travel time is considered as a cost, then solution of maximum discrete dynamic flow problem is same as the temporal solution of minimum cost static flow problem. To solve the minimum cost flow problemwe have to determine the possible paths in the network with the minimum cost through which maximum amount of flow can be sent.

Let $P = \{p_1, p_2, \dots, p_k\}$ be the path decomposition of feasible static flow f such that $f(p_k)$ units flow can be send along the path p_k. The travel time along the path p_k is $\tau(p_k) = \sum \tau_e : e \in p_k$. We can send $f(p_k)$ units flow along the path p_k at time units $0,1,2,\dots,T - \tau(p_k)$ and flow send after $T - \tau(p_k)$ time does not reach the destination within time T. Such $f(p_k)$ flows are temporally repeated flow for the time $T - \tau(p_k) + 1$ with flow value.

$$\sum_{p_k \in P} (T + \tau(p_k) - 1) f(p_k)$$

Minimum cost circulation algorithm [12]

1. In a given dynamic network$N = (V, A, u, s, t, \tau, T)$with additional arc$a = (t, s)$, set $f(i,j) = 0$ for all $(i,j) \in A$.
2. Apply Algorithm in [12] to find MCF f by considering transit time on arcs as cost.
3. Decompose f in to path flows using the Algorithm of Ford and Fulkerson [9]
4. Obtain a maximum dynamic flow.

The maximum temporally repeated flow can be obtained by finding the minimum cost circulation in the static network. For this, we add an arc $a = (t, s)$ in the network with capacity $u(a) = u(t, s) = \infty$ and travel time$\tau(a) = \tau(t, s) = -(T + 1)$ where cost on the arcs is replaced by time. The flow along the arc $a = (t, s)$ is ignored and resulting $s - t$ flow is decomposed in to paths.

$$Val(f, T) = \sum_{p_k \in P} \big(T + 1 - \tau(p_k)\big)\big(f(p_k)\big) = (T + 1)\text{val}(f) - \sum_{e \in A} \tau(e) f(e)$$

5. Conclusion:

Maximum flow algorithms are applicable to solve different real life problems. In this paper, we studied two different classes of the maximum flow algorithms and their efficiency criteria. We discussed about themaximum static flow algorithms and their developments in Section 3. Similarly, in Section 4,we discussed about the maximum dynamic flow, formation and use of time expanded graph.

References

1. Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *"Network Flows:Theory, Algorithm and Applications",* Prentice Hall, Englewood Cliffs. New Jercey (1993)
2. Ahuja, R. K., and Orlin, J. B., *"A fast and simple algorithm for the maximum flow problem",*Operation Research.37, 748-759, (1989)
3. Cheriyan, J. and Maheshwari, S. N., *"Analysis of pre flow push algorithms for maximum network",*SIAM Journal of Computing. 18, 1057-1086, (1989)
4. Cheriyan, J. and Mehlhorn, N., *"An analysis of highest-level selection rule in the preflow push max-flow algorithm", Information Processing Letters.* 69, 239-242, (1999)
5. Dhamala, T. N., Pyakurel, U. and Dempe, S., *"A critical survey on network optimization algorithms for evacuation planning problems",*International Journal of Operation Research. 15(3), 101-133,(2018)
6. Dinic E. A., *"Algorithm for solution of a problem of maximum flow in networks with power estimation"*, Soviet Mathematical Doklady.11**,** 1277-1280, (1970)
7. Edmonds, J. and Karp, R. M., *"Theoretical improvements in algorithmic efficiency for network flow problems",* Journal of Association of Computing Machinary.19, 248-264, (1972)
8. Ford, L. R. and Fulkerson, D. R.,*"Maximal flows through a network"*, Canadian Journal of Mathematics.8**,** 399-404, (1956)
9. Ford, L. R. and Fulkerson, D. R.,*"Constructing maximal dynamic flows, from static flows",*Operations Researh.6, 419-433, (1958)
10. Goldberg, A.V. and Tarjan R. E.,*"A new approach to the maximum flow problem"*,Journal of Association Computing Machinary. 35, 921-940, (1988)
11. Karzanov, A. V., *"Determining the Maximal flow in a network by the method of preflows",*Soviet Mathematics Dokladi. 15**,** 434-437, (1974)
12. Klein, M., *"A primal method for minimal cost flows with applications to the assignment and transportation problems",* Management Science. 14, 205-220, (1967)
13. Pyakurel, U. and Adhikari, M.C.,*"Priority based flow improvement with intermediate storage",* Submitted to International Journal of Operations Research. 2020, (cf. Optimization Online http://www.optimization-online.org/DB_HTML/2020/07/7891.html)
14. Pyakurel, U. and Dempe, S., *"Network flow with intermediate storage: models and algorithms",* SN Operation Research Forum. DOI:10.107/s43069-020-00033-0, (2020)
15. Pyakurel, U., Wagle, S. and Adhikari, M.C., *"Efficient lane reversals for prioritized maximum flow",* International Journal of Innovative Science, Engineering & Technology. 7 (2020)