A Comparative Study of Algorithms for Maximum Flow Problems

Ranjit Jha
Lecturer
Department of Mathematics
Kathfod International College, Balkumari, Lalitpur, Kathmandu
Email:ranjitjha68@gmail.com

ABSTRACT

The max flow problem means to send as much flow as possible from source to the sink satisfying the following two conditions: the capacity of each edge is greater than or equal to its flow and inflow and outflow are same throughout the network for every node except source and sink. Those problems which are related to maximum flow are called maximum flow problems. The maximum flow problem has its wide applications in real life situation like airline scheduling, communication networks; electrical power etc. Various algorithms are there to solve this problem in the literature. In this paper, we present comparative study of the existing algorithms to maximum flow problem.

Keywords: Maximum Flow Problem, Augmenting Path, Dinic Algorithm, Karzanov Algorithm.

Introduction

The maximum flow problem (MFP) is one of the basic problems in the network optimization. There are four types of network model: Shortest-path model, Minimum spanning tree model, Maximal-flow model and Minimum-cost capacitated network model. Among these four models, we have restudied on the Maximal-flow model. The objective of this model is to find the maximum number of flow that can be sent from source to sink through the different edges of the network. The network consists nodes and arcs. Arcs connect the nodes.

At first, the maximal flow problem was studied by Ford and Fulkerson [1962] and solved this problem by using the augmenting path algorithm by them. In 1962, Ford and Fulkerson gave well known Ford-Fulkerson algorithm [1962] to find the maximum flow in a flow network. Ford-Fulkerson uses depth-first search to find the augmenting paths through a residual graph. Edmond and Karp [1972] gave two labeling algorithms: first augments flow along shortest augmenting paths which runs in $O(nm^2)$ time and

second augments flow along paths with maximum residual capacity which runs in $O(m^2 \log U)$ time. Dinic [1970] introduced the concept of shortest path network, called layered network, and showed by constructing blocking flows in layered network, and the maximum flow can be obtained in $O(n^2 m)$ time. Karzanov [1972] introduced the concept of pre-flows and showed that an implementation that maintains pre-flows and pushes flow from nodes with excess obtains a maximum flow in $O(n^3)$ time. Cherkasssky [1977] presented further improvement of Karzanov's algorithm runs in $O(n^2 m^{\frac{1}{2}})$ time. Gabow incorporated scaling technique into Dinic's algorithm [1970] and developed an O(nmlogU) time algorithm. Goldberg and Tarjan suggested First In First Out (FIFO) and highest label pre-flow algorithms ran in $O(n^3)$ and $O(nm \log (n^2/m))$ time using simple data structures and dynamic tree data structures respectively. Derigs and Meier [1989] implemented several versions of Goldberg and Tarjan's algorithm and then they found that Goldberg and Tarjan's algorithm is substantially faster than Dinic's and Karjan's algorithms. Similarly, Anderson and Setubal [1992] find different versions like First In First Out, Highest Label, Stack to be the best for different classes of networks and queue implementations to be about four times faster than Dinic's algorithm [1970]. After Ford and Fulkerson [1962], many researchers have improved several algorithms for solving maximum flow problems. Maximum flow problem is used in many applied fields like Computer Science, Transportation, Scheduling, Telecommunication, Management, Logistics and other branches of Operations Research.

In this paper, we have restudied some of the existing maximum flow algorithms and compared their performance. This paper has been organized as follow: Section 2 includes formulation of maximum flow problem; Section 3 includes solution techniques; Section 4 includes the comparison of the approaches and Section 5 includes the concluding remarks.

Mathematical Formulation

Let G be the directed graph containing N and A as the sets of nodes and arcs that is edges respectively with positive integer capacity c_{ij} $\forall$ (i, j) $\in$ A. Also, let |N| = n and |A| = m. Suppose that the graph does not contain multiple arcs and $\forall$ (i, j) $\in$ A, $\exists$ $\forall$ (j,i)$\in$A, possibly having zero capacity. We define A(i) = {(i, j) $\in$ A : j $\in$ N} and U = max.{c_{ij}, (i, j) $\in$ A}. Mathematically, the maximum flow problem is stated as follow:

Maximize the flow value v,

Subject to

$$\sum_{i \in V} (x_{ij} - x_{ji}) = 0 \ \forall \ j \in V - \{s,t\} \quad \cdots\cdots(1)$$

$$\sum_{i \in V-(s)} x_{si} = v \quad \cdots\cdots(2)$$

$$\sum_{i \in V-(t)} x_{it} = v \quad \cdots\cdots(3)$$

Where V stands for the set of nodes; A stands for the set of edges; suffices I and j stand for the intermediate nodes; v stands for the total flow value of the problem; x_{ij} stands for the flow from a node i to the another node j and c_{ij} stands for the capacity on the arc (i, j). The unique nodes s and t are called the source and sink or destination respectively. Equation 1 represents the flow conservation; Equations 2 and 3 represent the net flow out of source and into the sink respectively. The main objective of this problem is to maximize the flow value in the network.

Solution Techniques

We have described here solution techniques which are used to solve the maximum flow problems as:

Breadth First Search (BFS) Algorithm

BFS [2014] is a graph traversing (traversing means visiting each and every vertex and edge exactly once in a well-defined order) algorithm. In this method, we start traversing from any vertex and then traverse the graph layer wise to exploring the neighbouring connected nodes. Then we move towards the next level neighbouring nodes. If the given graph is un-weighted, then we can use this algorithm which is very simple and fast. The time complexity of BFS is O(m + n), where n and m are number of nodes and number of edges respectively.

Depth First Search (DFS) Algorithm

It is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary nodes as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. DFS [2014] is a recursive algorithm which uses the backtracking (backtracking means when we are moving forward and there are no one nodes along the current path then we move to backwards on the same path to find node to traverse) idea. All nodes are visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected. The time complexity of DFS is O(m + n), where n and m are number of nodes and number of edges respectively.

Largest Augmenting Path Algorithm

A path from source to sink is said to be augmenting whose edges either non-full forward or non-empty backward edges. In Largest Augmenting Path Algorithm [1972], first we select a path from source to sink, we find minimum capacity or bottleneck capacity along the selected path and then send that minimum capacity to sink. Its time complexity is O(F.E), where F is total flow value and E is number of edges respectively.

Dinic Algorithm

Dinic's algorithm [1970] is a strongly polynomial maximum flow algorithm with running time $O(n^2m)$. It proceeds by constructing shortest path network, called layered

networks and by blocking flows in these networks. It is extremely fast in practice and works even better on the bipartite graphs giving a time complexity $O((mn)^{\frac{1}{2}})$ due to the algorithm's reduction to Hope-croft Karp. The algorithm was originally invented by Yefim Dinitz in 1969 and published in 1970. The algorithm was later modified slightly and popularized by Shimon Even [1976]. The introduction of the concepts of the level graph and blocking flow enable Dinic's algorithm to achieve its performance. It can be shown that the number of edges in each blocking flow increases by at least 1 each time and thus there are at most (n - 1) blocking flows in the algorithm, where n is the number of vertices in the network.

Layer Updating Method

Layer updating is the method proposed by To-Yat Cheung [1980] with the main ides is to maintain, by updating without destruction, a two way layered sub-networks throughout the whole process. On the basis of current flow pattern (construct two sun-networks) is called layered sub-network and referent sun-network of the original network.

Karzanov Algorithm

It is a pre-flow push algorithm, but pushes flow from the source to the sink using layered networks instead of distance labels. Karzanov [1972] describes a pre-flow based algorithm to construct a blocking flow in a layered network in $O(n^2)$ time. This repeatedly performs two operations: push and balance. The push operation pushes the flow from one layer an active node to the next layer very near to the sink in the layered network and the balance operation returns the flow that can't be sent to the next layer to the nodes in previous layer it came from. This algorithm repeatedly performs forward and reverse passes on active nodes. In a forward pass, the algorithm examines active nodes in the decreasing order of the layers they belong to and preforms push operations. In a backward pass, the algorithm examines active nodes in the increasing order of the layer they belong to and performs balance operations. The algorithm terminates where there are no active nodes. This algorithm constructs a blocking flow in a layered network in $O(n^2)$ time and hence the running time of the algorithm is $O(n^3)$.

Dinic-Karzanov Algorithm

Dinic-Karzanov Algorithm [1976] is the combination of Dinic [1970]and Karzanov [1972] methods but this method is same as the Karzanov method. In this method, the layered sub-network is replaced by Dinic's referent sub-network throughout all the advance and balance operations.

Kinariwala-Rao Algorithm

Kinariwala-Rao [1977] uses the flow switching method which is complicated and lengthy. It is alternate approach to the maximum flow problem based upon the concept of redistribution of flows so as to maximize the flow from the source to the sink. In this

method, flow conservation is achieved by decreasing the flow along those eliminating paths which connect a vertex with more outgoing flow than incoming flow to another vertex with more incoming flow than outgoing flow. Flow switching means, flow refers to velocity or physical movement of gas, steam, or liquid within a pipe that triggers the flow switch. When there is no flow present, the velocity either drops or completely stops, in either case, the switch will revert to its original position. The flow switching is used in water treatment systems, additive or blending systems, air supply systems and duct type heating etc.

Ford-Fulkerson Algorithm

The Ford–Fulkerson algorithm, a greedy algorithm for maximum flow that is not in general strongly polynomial. Ford-Fulkerson gives many basic facts about the maximum flow problem. In particular, the maximum flow value is equal to the minimum cut capacity and this theorem is called the max-flow min-cut, and that a flow is called maximum if and only if no augmenting path is found, called the augmenting path theorem. We introduce the Ford-Fulkerson augmenting path algorithm [1972], and give its implementation using the labelling method. They use the algorithm to prove the integrality theorem, which states that a maximum flow problem with integral arc capacities has an integral optimal solution.

Goldberg-Tarjan Algoriyhm (or, Push-Relabeling Algorithm)

The push relabel algorithm given by A.V. Goldberg and R. E. Tarjan [2014] works by manipulating the pre-flow in a graph. First step is saturating all the edges exiting the source. Next step is moving the excess into nodes that are estimated to be very near to the target. If at some point the excess of a node cannot reach the target, the excess is moved back into the source. At last, the pre-flow is therefore an actual flow and is the maximum flow.

Shortest Augmenting Path Algorithm

The shortest augmenting path algorithm (SAP) is given by Edmonds and Karp [1977] and Dinic [1970]. It is shown that the augmenting path length in SAP is non-decreasing monotone, and at most m augmenting paths of length k,$k \in (1,n-1)$, are found by the algorithm. Thus the number of iterations of the algorithm is at most $(n-1)m = O(n^3)$. This bound is tight by Zades, N. [1973] and concludes with a discussion of the implementation of the SPA by using BFS for finding shortest augmenting paths. The time complexity of the resulting method is $O(nm^2)$. However, SAP can be discovered in an average of O(n) time.

Generic Pre-flow Push Algorithm

The pre-flow-push algorithms [1995] maintain a pre-flow and proceed by examining the active nodes, i.e., nodes with positive excess. The main idea of the algorithm is to select an active node and to attempt to send its excess near to the sink. As sending flow

on admissible arcs pushes the flow near to the sink, the algorithm always pushes flow on admissible arcs. If the active node being examined has no admissible arc, then we increase its distance label to create at least one admissible arc. If active nodes are not found then the algorithm terminates.

Highest Label Pre-flow Push Algorithm

This algorithm always pushes flow from an active node with the highest distance label. Let $d^* = \max\{d(i):i \text{ is active}\}$. The algorithm first examines nodes with distance label d^* and pushes flow to nodes with distance label d^*-1, and these nodes, in turn, push flow to nodes with distance labels equal to d^*-2, and so on, until either the algorithm relabels a node or it has exhausted all the active nodes. When it has relabelled a node, the algorithm repeats the same process. Goldberg and Tarjan [2014] obtained a bound of $O(n^3)$ on the number of non-saturating pushes performed by the algorithm. After that, Cheriyan and Maheshwari [1989] showed that this algorithm actually performs $O(n^2\sqrt{m})$ non-saturating pushes and this bound is tight.

Lowest Label Pre-flow Push Algorithm

The lowest-label pre-flow push algorithm always pushes the flow from an active node with the smallest distance label. Its implementation is same as the highest-label pre-flow push algorithm. This algorithm performs $O(n^2 m)$ non-saturating pushes and runs in $O(n^2 m)$ time.

First In First Out (FIFO) Pre-flow Push Algorithm

This algorithm examines active nodes in the FIFO order and maintains the set of active nodes in a queue called QUEUE. It selects a node i from the front of QUEUE for examination. The algorithm examines node i until it becomes inactive or it is relabelled. Later, node i is added to the rear of QUEUE. The algorithm terminates when QUEUE becomes empty. Goldberg and Tarjan [2014] showed that the FIFO implementation performs $O(n^3)$ non-saturating pushes and can be implemented in $O(n^3)$ time.

Capacity Scaling Algorithm (CSA)

Capacity Scaling Algorithm was originally suggested by Gabow [1985]. Ahuja and Orlin [1988] subsequently developed a variant of this approach which is better empirically. The main idea behind the CSA is to augment flow along a path with sufficiently large residual capacity such that the number of augmentations is sufficiently small. The CSA uses a parameter Δ and with respect to a given flow x, defines the Δ - residual network as a sub-graph of the residual network where the residual capacity of every arc is at least Δ. We denote the Δ- residual network by $G(x, \Delta)$. The capacity scaling algorithm has the following three representative operations: Relabels, Augmentations and constructing Δ- residual networks. The running time of the capacity scaling algorithm is $O(nm \log U)$.

Excess Scaling Algorithms (ESA)

Excess-scaling algorithms [1988] are special implementations of the generic pre-flow push algorithms and incorporate scaling technique which dramatically improves the number of non-saturating pushes in the worst case. The essential idea in the (original) excess scaling algorithm is to assure that each non-saturating push carries sufficiently large flow so that the number of non-saturating pushes is sufficiently small. The algorithm defines the term sufficiently large and sufficiently small iteratively.

Let e_{max}= max{e(i):i active} and Δ be an upper bound on e_{max}. We refer to a node i with $e(i) \geq \frac{\Delta}{2} \geq \frac{e_{max}}{2}$ as a node with large excess, and a node with small excess otherwise initially, $\Delta=2^{[\log U]}$, i.e. the largest power of 2 ≤ U.

Original Scaling Algorithm

The original excess scaling algorithm performs a number of scaling phases with different values of the scale factor Δ. In the Δ - scaling phase, the algorithm selects a node i with large excess, and among such nodes selects a node with the smallest distance label, and performs push / relabel (i) with the slight modification that during a push on arc (i, j), the algorithm pushes min{e(i), r_{ij}, Δ-e(j)}, units of flow. (It can be shown that the above rules ensure that each non-saturating push carries at least $\frac{\Delta}{2}$ units of flow and no excess exceeds Δ). When there is no node with large excess, then the algorithm reduces Δ by a factor 2, and repeats the above process until Δ=1, when the algorithm terminates. To implement this algorithm, we maintain the singly linked stacks SLIST (k) for each k = 1, 2, … , 2n − 1, where SLIST (k) stores the set of large excess nodes with distance label equal to k. We determine a large excess node with the smallest distance label by maintaining a variable level and using a scheme similar to that for the highest-label pre-flow push algorithm. Ahuja and Orlin [2] have shown that the excess scaling algorithm performs $O(n^2 \log U)$ non-saturating pushes and can be implemented in $O(nm+ n^2 \log U)$ time.

Performance Analysis

The main aim of this paper is to compare the above approaches. Dinic [1970] and Edmonds and Karp [1972] independently showed that an augmenting path obtained by the BFS method is the shortest. Karzanov's [1972] max-flow algorithm is based on a concept of pre-flow, which is a function on the arcs that may violate, in a certain way, the flow conservation condition in nonterminal nodes. The algorithm, called the pre-flow method, takes advantages of handling pre-flows on intermediate iterations, due to which the running time of reduces to $O(n^3)$; here n and m are the numbers of nodes and number of arcs. Subsequently pre-flows and the idea of push operations have been widely used in other max-flow algorithms, in particular, in Cherkassky's algorithm [1977] is slightly faster and in Goldberg's pus-relabel algorithm [1985] of the same

complexity $O(n^3)$. Similar to Dinic algorithm, the pre-flow algorithm consists of O(n) stages, each solving a blocking flow in a layered network. The pre-flow algorithm solves the auxiliary problem in $O(n^2)$ time, thus yielding the time bound $O(n^3)$ for the whole algorithm. In fact, the algorithm of finding a blocking flow can be slightly modified so as to work with an arbitrary acyclic, not necessarily layered network.

Concluding Remarks

We consider here some of the algorithms related to maximum flow problems and compare their performance. There are number of efficient algorithms based on the Ford-Fulkerson method. Among them, DFS, BFS, and Largest augmentation methods are simple to solve the problems and each consist of cycles in which labels are created and paths are augmented, if found. The Dinic method is simple to use and is same as the BFS method except that a referent is created within each cycle and that a sub-procedure path is used for path searching within a referent (i.e. layer). The layer updating method is so lengthy, mainly because of the complicated process involved in updating the tree structures of the labels. Following sub procedures are used: advance, candidate, delete, update and flow change. The Karzanov method consists mainly of alternative calls to two sub procedures advance and balance. The Dinic-Karzanov method is a modification of the Karzanov method. An additional sub-procedure is used to create a referent, within which the advance and balance operations are confined. The Kinariwala-Rao method performs alternatively two processes: flow augmentation by saturating a cut and flow conservation by decreasing flow along eliminating paths. A sub-procedure called ELIM is used to identify and eliminate all the eliminating paths, each of which connects a vertex with excessive outflow to a vector with excessive inflow.

References

Avinash, Purva and Apurva (2014). A appraisal paper on Breadth-first search, Depth-first search and Red black tree. In: *International Journal of Scientific and Research Publications,* Volume 4, Issue 3, 2-4.

AHO, A. V., HOPCROFT, J. E., and ULLMAN, J. D. (1975). *The Design and Analysis of Computer Algorithms.* Addison- Wesley, Reading, Mass.

Ahuja, R. K., Murali Kodialam, Mishra, A.K., and Orlin, J. B. (1997). Computational Investigations of Maximum Flow Algorithm. *European Journal of Operational Research,* 97(3), 509-542.

Ahuja, R.K. and Orlin, J. B. (1988). A Fast and Simple Algorithm for the Maximum Flow problem. Bayer, G. (1968). Algorithm 324 Max-flow. *Comm. ACM 1*1, 2, 117 - 118.

Cherkassky, B. V. (1977). Algorithm for construction of maximal flow in networks with complexity of O(n^2 √p) operations, In: *Mathematical Methods in Economical Research,* issue 7, Nauka, Moscow, 117 - 126.

Dinic, E. A. (1970). Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. *Soviet Math.* Dokl. 11, 1277 - 1280.

Edmonds, J., and KARP, R. M. (1972). Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* 19, 2, 248 - 264.

Even, S. (1976). *The max-flow algorithm of Dinic and Karzanov, an exposition.* Unpub. Rep, Dept. Comptr. Sci., Techion, Haifa, Israel.

Fong, C. O., and RAO, M. R. (1972). Accelerated labelling algorithms for the maximal flow problem with applications to transportation and assignment problems. *Working Paper* No. 7222, Graduate School of Management, U. of Rochester, N. Y.

Ford, L. R., and Fulkerson, D. R. (1962). Flows In Networks *Princeton University Press*, *Princeton, N. J.,* 17 - 19.

Goldberg, A. V. (1985). A new max-flow algorithm Technical Report MIT. TM-291, Cambridge.

Goldberg, A. V. and Tarjan, R.E. (2014). Efficient Maximum Flow Algorithms, *Communication of the ACM*, 57(8), 82-89.

Harris, T. E. and Ross, F. S. (1955). *Fundamentals of a Method for Evaluating Rail Net Capacities.* RAND CORP SANTA MONICA CA.

Johnson, E. L. (1966). Networks and basic solutions. *Oper.* Res. 14, 619 - 623.

Karzanov, A. V. (1972) Determing the Maximal Flow in a Network by the Method of Pre-flows. *Soviet Math.* Dokl. 15, 434 - 437.

Kinriwala, B., and RAO, A. G. (1977). Flow Switching Approach to the Maximum Flow Problem *I. J. ACM* 24, 4, 630 - 645.

Lin, P.M., and Leon, B.J. (1974). Improving the Efficiency of Labelling Algorithms for Maximum flow in Networks. *Proc IEEE Int. Symp. on Circuits and Systems*, 162 - 166. Orlin, J.B. (2012) Max Flows in O(nm) Time, or Better, Revised.

Srinivasan, V., and Thompson, G.L. (1972). Accelerated Algorithms for Labelling and Relabelling Trees, with Applications to Distributions problems. *J. ACM* 19, 4, 712 - 726.

To-Yat Cheung (1980). Computational Comparison of Eight Methods for the Maximum Network Flow Problem. ACM *Transactions on Mathematical Software (TOMS),* 6(1), 1-16

Zades, N.A (1973). Bad Network problem for the Simplex Method and Other Minimum Cost Flow Algorithms. *Mathematical Programming,* 5:255 - 266.