International Journal of Computer Science and Communication Vol. 2, No. 2, July-December 2011, pp. 933-938

Parametric Maximum Flow Problem: Techniques and Algorithms

Deepika Sareen, Dr Deepak Garg

Thapar University, Patiala

sareendeepika@yahoo.co.in, dgarg@thapar.edu

Abstract
We encounter many different types of networks in our everyday lives, including electrical, telephone, cable, highway, rail, manufacturing and computer networks. Networks consist of special points called nodes and links connecting pairs of nodes called arcs. The maximum flow problem is one of the most fundamental problems in network flow theory and has been investigated extensively. From last few decades researchers have made a steady stream of innovation that have resulted in new solution methods and improvements to known methods. Continuous improvements to algorithms have been made by researchers for solving several classes of problems. In this paper recent techniques and algorithms related to parametric maximum flow problem are given.
Keywords: Max Flow, Efficient Algorithms, Parametric Flow

1.0 Introduction

This problem was first formulated by Fulkerson and Dantzig (1955) and Dantzig and Fulkerson (1956), and solved by Ford and Fulkerson (1956) using their well known augmenting path algorithm. Edmonds and Karp (1972) showed that the Ford and Fulkerson algorithm runs in time O (nm^2) if flows are augmented along shortest paths from source to sink. Independently, Dinic (1970) introduced the concept of shortest path networks, called layered networks, and obtained an O (n^2m) algorithm. This bound was improved to O (n^3) by Karzanov (1974), who introduced the concept of preflows in a layered network. Efficient algorithms for computing maximum flows are important not only because they are applied directly to the analysis of traffic or communication networks, but are often employed in the sub-problems of other network problems. Fundamental algorithms for network flow, including methods for maximum flow problem, were designed and efficient algorithms exist to solve different instances of this problem [1]. A natural generalization of the maximum flow problem is obtained by making the capacities of certain arcs functions of a single parameter λ. This problem is known as the parametric maximum flow problem. The current best time bounds for the ordinary maximum flow problem on a network with n vertices, m arcs, and integral arc capacities bounded by U are $O(nm\log_{m/(n \log n)} n)$ [2] and $O(\min\{n^{2/3},m^{1/2}\}m\log(n^2/m) \log U)$ [3]. The former algorithm is based on the push-relabel method [4]. In this problem, some or all arcs emanating from the source node have capacities that are increasing linear functions of a parameter λ, while other arcs have fixed capacities. We wish to determine the maximum flow in the network for all values of λ □ (0, ∞). We refer to this problem as the source parametric maximum flow (SPMF) problem. Gallo, Grigoriadis, and Tarjan [5] give applications of this problem. In fact, they consider the problem in which arcs directed into the sink are also parametrized.The SPMF problem has been solved by Itai and Rodeh [6] in O (kn^2m) time using the proportional augmentation algorithm, where k is the number of arcs with parametric capacities. Proportional augmentation is a technique for augmenting flows on a number of paths from different origins to the same destination in a proportionate manner until none of the arcs in these paths gets blocked. Itai and Rodeh's algorithm proceeds by creating at most kn layered networks and constructing blocking flows in these networks by proportional augmentations. Gallo, Grigoriadis, and Tarjan [5] have solved this problem in O (nm log (n^2/m)) time.

2.0 Maximum Flow Problem:

The maximum flow problem is to find the max. flow that can be sent through the arc of the network from some specified node source(S) to specified node sink(T) and a capacity function that maps edges to positive real numbers, u: E $\longmapsto R^+$. Maximum flow involves a directed network with arc carry flow; the relevant parameters are arc capacity and flow [7] [8] [9].

Definitions and Notation of Maximum Flow Problem:

Suppose G (V, E) is a finite directed graph in which every edge (u, v) □E has a non-negative, real-valued capacity c (u, v). If (u, v) □ E, we assume that c (u, v) = 0. We distinguish two vertices: a source s and a sink t. A flow network is a real function f:V×V R with the following three properties for all nodes u and v:

Capacity constraints:	f(u,v)≤c(u,v). The flow along an edge cannot exceed its capacity.
Skew symmetry	f(u,v)=- f(u,v). The net flow from u to v must be the opposite of the net flow from v to u

$\sum_{w \in V} f(u,w) = 0$, unless

Flow conservati — $u = s$ or $u = t$. The net flow to a node is zero, except for the source, which produces flow, and the sink, which "consumes" flow.

Residual Network:

The residual capacity of an edge is $c_f(u, v) = c(u, v) - f(u, v)$. This defines a residual network denoted $G_f(V, E_f)$, giving the amount of available capacity. In short residual network is nothing but the remaining network i.e. it shows that how much more flow can be send on the network.

Augmenting path methods:

An Augmenting path P is a simple path from s to t on a residual network that is an alternating sequence of the vertices and edges of the form s, e_1, v_1, e_2, v_2,------e_k,t in which no vertex is repeated and no forward edge is saturated and no backward edge is free. It can be shown that the flow through a network of rational capacities is optimal if and only if it contains no augmenting path, and since each augmentation adds to the flow, we will eventually find the maximum. An augmenting path is a path (u_1, u_2 ,..., u_k) in the residual network, where $u_1 = s$, $u_k = t$, and $c_f(u_i, u_{i+1}) > 0$. A network is at maximum flow if and only if there is no augmenting path in the residual network.

The Ford-Fulkerson algorithm (named for L. R. Ford, Jr. and D. R. Fulkerson) computes the maximum flow in a flow network by using augmenting path algorithm [10]. It was published in 1956. The name "Ford-Fulkerson" is often also used for the Edmonds-Karp algorithm, which is a specialization of Ford-Fulkerson. The first polynomial time algorithm for the max flow problem produced by Edmonds and Karp in 1969. By using this algorithm we find the maximal flow of the given directed network using the augmenting path either one path at a time. Edmonds and Karp showed that by augmenting flows along shortest paths, the augmenting path algorithm runs in time O (nm^2) on networks with n nodes and m arcs [11].

In 1970 Dinic discover another algorithm in which all shortest length augmenting at once using a layered network approach using this approach improve the running time complexity the running time of this algorithm is O (n^2m) [12] but this method failed for preflow concept.

In 1974 Karzanov is introduced a preflow concept i.e. preflow is like a flow, except that total amount flowing into a vertex is allowed to exceed the total amount flowing out. The method maintains a preflow in the original network and pushes local flow excess toward the sink along what are estimated to be shortest paths. This algorithm runs as fast as any other known method on dense graphs, achieving an O (n^3) time bound on an n-vertex graph [7].

Generalization of the maximum flow problem is obtained by making the capacities of certain arcs functions of a single parameter λ. This problem is known as the parametric maximum flow problem.

Definition and Notation of Parametric Maximum Flow: Given a capacitated network G = (N, A, l, u, s, t), let n = |N| and m = |A|. The upper bound function and the lower bound function are two nonnegative functions, u (i, j) and l (i, j) associated with each arc (i, j) □ A. The network has two special nodes: a source node represented by s and a sink node represented by t. A flow is a function f: A→R^+ satisfying the next conditions:

$$\sum_{j\,|\,(i,j)\in A} f(i,j) - \sum_{j\,|\,(j,i)\in A} f(j,i) = \begin{cases} v, i = s \\ 0, i \neq s, t \\ -v, i = t \end{cases}$$

for some v ≥ 0, where v is referred to as the value of the flow f. Any flow on a directed network satisfying the flow bound constraints:

$\ell(i, j) \le f(i, j) \le u(i, j) \forall (i, j) \in A$

for every arc (i, j) □A is referred to as a feasible flow. A cut is a partition of the node set N into two subsets S and T = N − S, represented using the notation [S, T] .Alternatively, a cut can be defined as the set of arcs whose endpoints belong to different subsets S and T. A cut is nontrivial if both S and T are nonempty. An arc (i, j) with i□S and j□T is referred to as a forward arc of the cut while an arc (i, j) with i□T and j□S as a backward arc of the cut. Let (S, T) denote the set of forward arcs in the cut and let (T, S) denote the set of backward arcs. A cut [S, T] is s-t cut if s□S and t□T [13].

Basically parametric algorithm is nothing but the extension of the maximum flow algorithm in which the some arcs capacities are not fixed but the functions of a single parameter.

For the parametric flow problem, the lower bound and upper bound functions as well as the flow functions on any of the arcs (i, j) are piecewise linear functions of a parameter λ defined on an interval [0, □]. On the set of all piecewise linear functions and ordering cannot be defined for the whole interval [0, □] since two piecewise linear functions are not necessarily comparable. Therefore a partitioning J_k of the interval of the parameter [0, □] into disjoints subintervals, J_1 □ . . . □ J_k = [0, □] with $J_{k1} \cap J_{k2}$ = Ø, for all $k_1 \neq k_2$ must be defined so that on each of the subintervals J_k and ordering to be defined as:

$f_1(\lambda) \le f_2(\lambda)$ for J_k $\Leftrightarrow f_1(\lambda) \le f_2(\lambda), \forall \lambda \in J_k$.

As parametric algorithm is extension of the maximum flow so the algorithm parametric maximum flow has their basic idea from maximum flow algorithm. Gallo et al. [5] showed that certain versions of the push-relabel algorithm for ordinary maximum flow can be extended to the parametric problem while only increasing the worst-case time bound by a constant factor. Recently Zhang et al. [8, 14] proposed a novel, simple balancing algorithm for the

parametric problem on bipartite networks. They claimed good performance for their algorithm on networks arising from a real-world application.

The pseudoflow algorithm for the maximum flow problem is one that employs only pseudoflows and does not generate flows explicitly. The algorithm solves directly a problem equivalent to the minimum cut problem - the maximum blocking cut problem. Once the maximum blocking cut solution is available, the additional complexity required to find the respective maximum flow is

O (mlog n). A variant of the algorithm is a new parametric maximum flow algorithm generating all breakpoints in the same complexity required to solve the constant capacities maximum flow problem. The pseudoflow algorithm has also a simplex variant, pseudoflow-simplex, that can be implemented to solve the maximum flow problem. One feature of the pseudoflow algorithm is that it can initialize with any pseudoflow. This feature allows it to reach an optimal solution quickly when the initial pseudoflow is "close" to an optimal solution. The complexities of the pseudoflow algorithm, the pseudoflow-simplex and the parametric variants of pseudoflow and pseudoflow-simplex algorithms are all O (mnlog n) on a graph with n nodes and m arcs. Therefore, the pseudoflow-simplex algorithm is the fastest simplex algorithm known for the parametric maximum flow problem [15].

Generic preflow-push algorithm on bipartite networks works by examining active vertices and push in excess from these vertices to vertices estimated to be closer to t. If t is not reachable, however, an attempt is made to push the excess back to s. Eventually, there will be no excess on any vertex other than t. At this point the preflow is a flow, and moreover it is a maximum flow [4]. The algorithms use distance labels to measure the closeness of a vertex to the sink or the source.

The bipartite preflow-push algorithm. The basic idea behind the bipartite preflow-push algorithm is to perform bipushes from vertices in V_1. A bipush is a push over two consecutive eligible edges; it moves excess from a vertex in V_1 to another vertex in V_1. This approach has all the advantages of the usual approach, and the additional advantage that it leads to improved running times. This approach ensures that no vertex in V_2 ever has any excess. Since all the excess resides at vertices in V_1, it suffices to account for the nonsaturating bipushes emanating from vertices in V_1. Since $V_1 \leq V_2$, the number of nonsaturating bipushes is reduced. The bipartite preflow-push algorithm is a simple generalization of the generic preflow-push algorithm [18].

The max flow and min cut problems are an important part of the foundations of combinatorial optimization, see, e.g., Ahuja, Magnanti and Orlin [1].There are many applications where the capacities of the arcs vary as a function of a scalar parameter λ, and where we would like to compute max flows and/or min cuts for all values of λ. In many of the applications the min cuts are more important than the max flows. When capacities depend on a parameter λ we consider it as Min cut problem. [13].

3. Algorithms Maximum Flow

3.1 GGT (Gallo-Grigoriadis-Tarjan) Algorithm

3.1.1 Push-Relabel Algorithm

The GGT algorithm is based on the push-relabel algorithm [4] for the maximum flow problem. The push-relabel algorithm worked in more localized manner than the Ford Fulkerson method. As examine the entire residual network for augmented path, Push-relable algorithm works on single vertex at a time, sees only at the vertex neighbors in the residual networks. Unlike Ford Fulkerson method Push-relabel algorithm do not maintain flow conversion property. Push-relabel algorithm uses two basic operations, push and relabel, and maintains a flow and integral distance labels on vertices. The important properties of the algorithm are that the distance labels are monotonically increasing, the value of each distance label changes by O (n), and the work of the algorithm is charged to the distance label increases

3.1.2 GGT Algorithm

The GGT model of a max flow network has the usual max flow network on a directed graph with source s, and sink t. The capacity $c_{ij}(\lambda)$ on arc i → j is a function of a scalar parameter λ, where we assume that

$$c_{ij}(\lambda) \begin{cases} non-increasing\ in\ \lambda \text{ when } i = s \\ non-decreasing \text{ in } \lambda \text{ when } j = t \\ constant\ \lambda \text{ otherwise} \end{cases}$$

Suppose that we want max flows and min cuts for an increasing sequence of values of λ, say $\lambda_1 < \lambda_2 < \cdots < \lambda_k$, with k = O (|N|). The λ_i are given on-line, i.e., we might not find out the value of λ_{i+1} until after we compute the max flow for λi.

GGT starts by computing a max flow for λ_1 using the push-relabel algorithm of Goldberg and Tarjan [4]. This means that our description of how push-relabel works for GGT will be reversed from the description in [5]. Thus distance labels will be a lower bound on distance to s instead of distance from t, etc. Residual network can be thought as residual arc i → j whenever we have arc i → j in graph with $f_{ij} < c_{ij}(\lambda)$(we can push forwards on arc i → j) and if we have $f_{ij} > 0$ then we can push backwards on j → i. The algorithm keeps a distance label l_i on each node i that measures the approximate length of the shortest augmenting path (shortest directed residual path) to s, and it maintains the properties that $l_s = 0$, $l_t = |N|$, and $l_i \leq l_j + 1$ ∀residual arcs.

It also maintains the property that the excess $e_i \equiv \sum_j f_{ij} - \sum_k f_{ki}$ is non-negative for the current flow f.

Then GGT computes λ_2 and updates the capacities. It then updates the flows f by setting $f_{jt} = c_{jt}(\lambda_2)$ for arcs j → t with $l_j < |N|$ (this update is given as $x_{jt} = \max(f_{jt}, c_{jt}(\lambda_2))$ in [5], but it is easy to see that this expression always equals $c_{jt}(\lambda_2)$,and $f_{si} = \min(f_{si}, c_{jt}(\lambda_2))$. GGT then show that this

new f satisfies $l_i \leq l_j + 1$ ($\forall$residual arcs) and $e_i \equiv \sum_j f_{ij} - \sum_k f_{ki}$ w.r.t. the distance labels at optimality for λ_1. Thus a max flow and min cut for λ_2 can be computed starting from this new flow and the old distance labels. GGT continues in this way for all λ up to λ_k. Since the distance labels never decrease, when the number of parameter values is O(|N|) all this processing takes only the same time as a single max flow, or O(|N||A| log(|N|^2/|A|)) time for the dynamic tree implementation of the push-relabel algorithm.

3.2 Parametric Maximum Flow Algorithm

In this section we describe two algorithms for the parametric flow problem, a simple algorithm based on graph contraction and the GGT algorithm, which also uses amortization to improve the worst-case complexity.

Let c (i, j) denote the capacity of edge (i, j) as a function of λ and suppose that we wish to solve the maximum flow problem for parameter values $\lambda_1 \leq \lambda_{2.....} \leq \lambda_l$. Clearly, for *l* different values of λ, a solution can be found using *l* invocations of a maximum flow algorithm. This approach takes no advantage of the similarity of the successive problems to be solved, however. Gallo, Grigoriadis, and Tarjan [5] gave an algorithm for finding the maximum flow for 0(n) increasing values of λ in the same asymptotic time that it takes to run the Grigoriadis -Tarjan maximum flow algorithm once. If the capacities are linear functions of λ, it is easy to show that the value of the maximum flow, when viewed as a function of λ, is a piecewise linear function with no more than n - 2 breakpoints.

Let start by giving one iteration of the algorithm, i.e., determining the maximum flow for parameter value λ_i, if the maximum flow for parameter value λ_{i-1} is given. The algorithm appears in below:-

Step 1 (Update preflow)
Let k=k+1
□ (s, i) □E with d (i) < 2n$_1$. Let f(s, i) = max {uλ_i (s, i), f(s, i)}.
□ (i, t) □E. let f (i, t) = min {uλ_i (i, t), f (i, t)).
□v□ V_2while e (v) > 0, do push/relabel (v).
Step 2 (Find maximum flow) Run the bipartite FIFO algorithm on the network with capacities uλi, beginning with flow f and distance labels d.

First, we update the capacities. The capacity of an edge leaving the source may have increased. If so, we saturate the edge, by setting its flow equal to its new capacity. The capacity of an edge leaving the sink may have decreased. If it has decreased below the flow on the edge, we decrease the flow so that it is equal to the capacity. Since t □V_1 by assumption, this may create excess on vertices in V_2. Therefore, we immediately push any such excess to vertices in V_l, thus re-establishing the invariant that no excess is on vertices in V_2. The second step consists of running the bipartite FIFO algorithm in the network beginning with the current f and d. This gives us a maximum flow for the parameter value λi.

3.3 Star Balancing Algorithm

The star balancing algorithm is an algorithm used for solving instances of the parametric maximum flow problem that meet the following constraints:

1. The network is bipartite, that is, V \{s, t} can be partitioned into sets V_1 and V_2 such that all arcs from s are to members of V_1, all arcs to t are from members of V_2, and all other arcs are from members of V_1 to members of V_2
2. All arcs from s have capacity λ
3. All arcs to t have constant capacity
4. All other arcs have infinite capacity.

For each arc (s, u), we define λ (f, u) to be the unique value of λ such that c ((s, u), λ) = f(s, u), and refer to it as u's λ - value. Additionally, it will be useful to have notation for describing the changes made to the flow during the process of the algorithm's execution. Define a z-straddling α-move to be the process of starting from an initial flow f and pushing $\alpha > 0$ units of flow along a simple cycle (s, u_1, v, u_2, s) for which λ (f, u_1) + $\alpha \leq z \leq \lambda$ (f, u_2) − α. Any z-straddling move for any z is defined to be a balancing move.

The star balancing algorithm begins by replacing the arcs from the source with arcs of infinite capacity, and then finding an arbitrary maximum flow in the resulting network, which can be done in linear time. Next, the algorithm repeatedly balances members v of V_2 by changing the current flow f to a new flow f′ via modifying the flows on arcs among {(s, u) □ (u, v) | (u, v) □ A} so that there are no remaining balancing moves involving v.

Conclusion : There is a lot of scope of doing research in different areas of max flow problems. GGT framework is that several different max flow algorithms fit into the GGT framework, but others such as the (in most cases faster) Goldberg-Rao algorithm does not. It might be possible to find a different or more comprehensive framework for parametric min cut that includes other max flow algorithms, or the problems which are not currently being look in content of GGT framework can be relooked into it. Due to increase in the volume and complexity of application data, few of the corresponding algorithms of max flow need to be remade for parallel, distributed architecture. For all large data sets Simultaneous parametric maximum flow algorithm (SPM) takes less time to get complete curve than the time needed by the Preflow-Push algorithm to find the solution at single lambda value. Analysis of complexity of algorithm is not yet defined. Some of the previously developed algorithms can be improved using recently available data structures for large datasets.

8. References

[1] Ahuja, R., Magnanti, T., Orlin, J. Network Flows. Theory, Algorithms and Applications. Prentice Hall, Inc., Englewood Cliffs, New Jersey 1993.

[2]V. King, S. Rao, and R. Tarjan. A Faster Deterministic Maximum Flow Algorithm. J. Algorithms, 17:447–474, 1994.

[3] A. V. Goldberg and S. Rao. Beyond the Flow Decomposition Barrier. J. ACM, 45(5):783–797, 1998.

[4] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum-Flow Problem. J. ACM, 35(4):921–940, 1988.

[5] Gallo, G., Grigoriadis M.D., and Tarjan R.E. A Fast Parametric Flow Algorithm. SIAM Journal of Computing, 18, 30-55, 1989.

[6] Itai, A. and Rodeh, M. Scheduling Transmission in a Network, Journal of Algorithms, 6, 409-429 ,1985.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Book on Introduction to Algorithms Clifford Stein, Third Edition, 2010.

[8]R. Tarjan, J.Ward, B. Zhang, Y. Zhou, and J.Mao. Balancing Applied to Maximum Network Flow Problems. In Proc. ESA, LNCS 4168, pages 612–623, 2006.

[9] L. R. Ford, Jr., and D. R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, N.J., 1962.

[10] Ford, L.R., Jr. and Fulkerson, D.R., "Maximal Flow Through a Network," Canadian Journal of Mathematics, 8, 399-404 , 1956.

[11] Edmonds, J., and Karp, R. M. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. J. ACM 19, 2, 248-264, Apr. 1972.

[12] Dinic, E. A. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. Sov. Math. Dokl. II (1970), 1277-1280.

[13] Mircea Parpalea. Min-Max Algorithm For The Parametric Flow Problem Bulletin of the Transilvania University of Bra¸sov , Vol 3(52) , 2010.

[14] B. Zhang, J. Ward, and Q. Feng. Simultaneous Parametric Maximum Flow Algorithm with Vertex Balancing. Technical Report HPL-2005-121, HP Labs, 2005.

[15] Dorit S. Hochbaum. The Pseudoflow algorithm: A New Algorithm for the Maximum Flow Problem, Operations research, 2008.