[Volume 5 issue 9 September 2017]
Page No.1797-1807 ISSN :232-7167

Design of algorithms to Generalized Minimum Flow Network Problem

A. Anto Kinsley , B. Uma Maheswari,

[1]Associate Professor, Department of Mathematics, St. Xavier's College (Autonomous), Palayamkottai, Tirunelveli - 627002, Tamil Nadu, India.
[2]Research Scholar, Department of Mathematics, St. Xavier's College (Autonomous), Palayamkottai, Tirunelveli - 627002, Tamil Nadu, India.

Corresponding Author:- A. Anto Kinsley

Associate Professor, Department of Mathematics, St. Xavier's College (Autonomous),Palayamkottai-627002,Tirunelveli, Tamil Nadu, India.

Abstract
In any traditional network, there is an implicit assumption that flow is conserved on every arc. In generalized networks, each arc has a positive multiplier $\gamma(u,v)$ called a gain factor, associated with it, representing the fraction of flow that remains when it is sent along that arc. The generalized maximum flow problem is identical to the traditional maximum flow problem, except that it can also model network with "leak" flow. In this paper, an algorithm is designed for the generalized minimum flow problem that consists of applying a maximum flow algorithm by modifying the network. This algorithm always decreases flow along paths from source vertex to the sink vertex with sufficiently large residual capacity and it runs in O (EV) time.

Keywords: Network flow, minimum flow problem, residual capacity, gain function.

1. Introduction

Although it has its own applications, the minimum flow problem was not dealt so often as the maximum flow and minimum cost flow problem. There are many problems that occur in economy that can be reduced to minimum flow problems.

For instance, we present the machine setup problem. A job shop needs to perform p tasks on a particular day. It is known the start time $\pi(i)$ and the end time $\pi'(i)$ for each task i, $i = 1,\dots p$. The workers must perform these tasks according to this schedule so that exactly one worker performs each task. A worker cannot work on two jobs at the same time. It is known the setup time $\pi_2(i, j)$ required for a worker to go from task i to task j. We wish to find the minimum number of workers to perform the tasks.

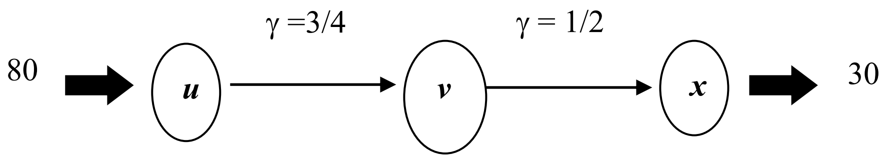
We can formulate this problem as a minimum flow problem in the network $G = (N, A, l, c, s, t)$, where $N = N_1 \cup N_2 \cup N_3 \cup N_4$, $N_1 = \{s\}$, $N_2 = \{i \setminus i = 1,\dots p\}$, $N_3 = \{i' \setminus i' = 1,\dots p\}$, $N_4 = \{t\}$, $A = A_1 \cup A_2 \cup A_3 \cup A_4$, $A_1 = \{(s, i) \setminus i \in N_2\}$, $A_2 = \{(i, i') \setminus i, i' = 1,\dots p\}$, $A_3 = \{(i', j) \setminus \pi'(i') + \pi_2(i', j) \le \pi(j)\}$, $A_4 = \{(i', t) \setminus i' \in N_3\}$, $l (s, i) = 0$, $c (s, i) = 1$, for any $(s, i) \in A_1$, $l (s, i) = 0$, $c (s, i) = 1$, for any $(s, i) \in A_1$, $l (i, i') = 1$, $c (i, i') = 1$, for any $(i, i') \in A_2$, $l (i', j) = 0$, $c (i', j) = 1$, for any $(i', j) \in A_3$, $l (i', t) = 0$, $c (i', t) = 1$, for any $(i', t) \in A_4$.

We solve the minimum flow problem in the network $G = (N, A, l, c, s, t)$ and the value of the minimum flow is the number of workers that can perform the tasks.

The computation of a minimum flow in a network has been investigated by Ciurea and ciupala in [2], [3], [4]. The brief outline of the paper is given below. Section 3 deals with an algorithm for generalized minimum flow in networks. Section 4 deals with an example for the application of algorithms.

The generalized maximum flow problem is a natural generalization of the traditional maximum flow problem. In traditional networks, there is an implicit assumption that flow is conserved on every arc. This assumption may be violated if water leak as it is pumped through a pipeline.

In a generalized network, a fixed percentage of the flow is lost when it is sent along an arc. Specially, each arc (u, v) has an associated gain factor $\gamma(u, v)$. When $f(u, v)$ units of flow enter into the arc (u, v) through node u then $\gamma(u, v) f(u, v)$ arrive at v. As the example in below illustrates, if 80 units of flow are sent into an arc (u, v) with gain factor 3/4, then 60 units reach node v: if these 60 units are then sent through an arc (v, x) with gain factor ½, then 30 units arrive at x.



2. Terminology and Preliminaries

In this section we discuss some basic notions and results used in the rest of the paper.

We consider a capacitated network $G = (N, A, l, c, s, t)$ with a nonnegative capacity $c(x, y)$ and with a nonnegative lower bounds function $l(x, y)$ associated with each arc $(x, y) \in A$. We distinguish two special vertices in the network G, a source vertex s and a sink vertex t. We further assume, without loss of generality that the network contains no parallel edges. A *flow* is a function $f: A \to R_+$ satisfying the conditions:

$$f(x,N) - f(N,x) = \begin{cases} \upsilon, & if\ x = s \\ 0, & if\ x \neq s,t \\ -\upsilon, & if\ x = t \end{cases} \qquad (1.a)$$

$$l(x,y) \leq f(x,y) \leq c(x,y), \quad \forall (x,y) \in A \qquad (1.b)$$

for some $\upsilon \geq 0$ Where $f(x,N) = \sum_{y/(x,y)\in A} f(x,y)$ and $f(N,x) = \sum_{y/(y,x)\in A} f(y,x)$ .We refer to υ as the *value of the flow f*.

The maximum (minimum) flow problem is to determine a flow f for which υ is maximized (minimized).

For solving the minimum flow problem we will use the following definitions.

A *pre-flow* f is a function $f: A \to R_+$ that satisfies (1.b) and

$f(N, x) - f(x, N) \geq 0$, $\forall x \in N - \{s, t\}$ for maximum flow problem and (2.a)

$f(x, N) - f(N, x) \leq 0$, $\forall x \in N - \{s, t\}$ for minimum flow problem. (2.b)

For any pre-flow f we define the *excess*, respectively *deficit* of a vertex x as

$\bar{e}(x) = f(N, x) - f(x, N)$, $\forall x \in N$ and (3.a)

$\hat{e}(x) = f(x, N) - f(N, x)$, $\forall x \in N$. (3.b)

for maximum and minimum flow problem respectively .

We refer to a vertex x with $\hat{e}(x)=0$ as a *balanced vertex*. A pre-flow f satisfying the condition $\hat{e}(x)=0$ for each vertex $x \in N - \{s, t\}$ is a *flow*. Thus, a flow is a particular case of pre-flow.

For the minimum flow problem, the *residual capacity* $\hat{r}(x,y)$ of any arc $(x, y) \in A$, with respect to a given pre-flow f, is given by $\hat{r}(x,y) = c(y,x) - f(y,x) + f(x,y) - l(x,y)$. By convention, if $(x, y) \in A$ and $(y, x) \notin A$ then we add the arc (y, x) to the set of arcs A and we set $l(y, x) = 0$ and $c(y, x) = 0$.

The network $\hat{G}_f = \left(\hat{N}, A\right)$ consisting only of the arcs with positive residual capacity is referred to as the *residual network* (with respect to the pre-flow *f*).

A *cut* is a partition of the vertex set *N* into two subsets *S* and $T = N - S$. We represent this cut using notation [*S*, *T*]. We refer to a cut [*S*, *T*] as an *s-t cut* if $s \in S$ and $t \in T$. We refer to an arc (*x*, *y*) with $x \in S$ and $y \in T$ as a *forward arc* of the cut and an arc (*x*, *y*) with $x \in T$ and $y \in S$ as a *backward arc* of the cut.
Let (*S*, *T*) denote the set of forward arcs in the cut and (*T*, *S*) denote the set of backward arcs.

For the minimum flow problem, we define the *capacity* $\hat{C}[S,T]$ of a *s-t* cut [*S*, *T*] as $\hat{C}(S,T) = l(S,T) - C(T,S)$. We refer to an *s-t* cut whose capacity $\bar{C}(S,T)$ is the maximum among all s-t cuts as a *maximum cut*.
For example, suppose that *N* is a network with edge capacities C_{ab} and $P = \{P_{ab}\}$ is an assignment of flow to the edges of *N*. Thus we assume that, at each internal node, the flow in is at least as large as the flow out. We call *P* a feasible pre-flow if, in addition $0 \le P_e \le C_e$. Our general strategy is to start with an initial pre-flow then successively modify the flows on various edges so that at all times the pre-flow is feasible and moves steadily toward satisfying flow conservation.
Suppose we have a network *N* with a pre-flow $P = \{p_{ab}\}$. If there is an internal vertex with positive excess $ex(a) > 0$ (deficit $\hat{e}(a) < 0$) then we would like to push (pull) this flow away from *a* and toward the sink *t*. If we can somehow push all the excess (deficit) flows to the sink then we will have a maximum (minimum) flow. If we want to pull deficit flow from *b* to *a* then we can do it backward along *b a* if $p_{ba} < c_{ba}$ or forward along *ab* if $p_{ab} > 0$ or both. A pull on *ba* is performed by moving up to $\hat{e}(a)$ units of flow from *b* to *a* with up to $c_{ba} - p_{ba}$ units moving backward on *ba* and up to p_{ab} units moving forward on *ab*. Once a pull along *ba* has been performed, the deficit $\hat{e}(a)$ at '*a*' will increase while $\hat{e}(b)$ will decrease.

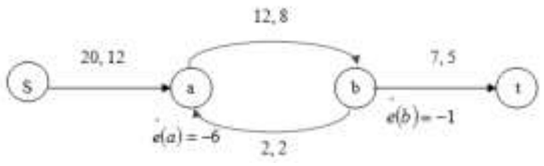


Figure1. A simple pre-flow

As an example, consider the simple network in Figure 1. There are 6 deficit units at *a.* We can pull all 6 units along *ab* with 4 units moving backward along *ab* and 2 units moving forward along '*ba'*. The resultant flow is shown in Figure 2. The pre-flow now satisfies flow conservation at '*a*' but not at '*b*'.

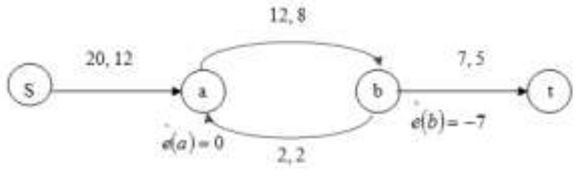


Figure2. The pre-flow after pulling on *ab*

There might be several different ways to perform a push along *ab*. If, in the example, we were only pushing away an excess of 3 units there would not be enough new flow both to saturate *ab* and to empty *ba*; there would be a choice to make. Moreover, the push on *ab* might not exhaust the excess at '*a*' so that further pushes are needed at '*a*'.

2.1 The Min-Flow Max-Cut Theorem

Let $S \subset V(X)$ and suppose that S contains the source and S does not contain the sink t. Let T denote all other vertices of X, that is, $T = V(X) - S$.

The cut (S, T) is the set of all edges whose initial vertex is in S and whose terminal vertex is in T. Then we state the following Lemma [1]. It is useful to prove the following theorem.

Lemma 2.1.1. Let f be a flow value Q in a network X, and let (S, T) be a cut in X. Then
$Q = f(S, T) - f(T, S) \leq \text{cap}\ (S, T)$.

Theorem 2.1.2. The minimum possible value of any flow in a network is equal to the maximum capacity of any cut in that network.

Proof: Let U and V be the two sets of vertices of the network X and let f be a flow function for X.
Let $f(U, V)$ be the sum of the values of the flow function along all edges whose initial vertex lies in U and whose terminal vertex lies in V. Let cap (U, V) be the sum of the capacities of all of those edges. Then the net flow out of U is $f\left(U,\bar{U}\right) - f\left(\bar{U},U\right)$

Let f be a flow in X of minimum value and call procedure label and scan (x, f, *why halt*). Let S be the set of vertices of X that have been labeled when the algorithm terminates with '*why halt*' = "flow is minimum". Clearly $s \in S$, $t \notin S$. We claim that every edge of the cut is saturated.

Let $(x, y) \in$ cut (S, T). Then $x \in S$, $y \notin S$. Suppose that (x, y) is not saturated. Then y can be labeled, when we scan X, Then $y \in S$ which is a contradiction. Similarly, if (y, x) is an edge when $y \in T$ and $x \in S$, then the flow $f(y, x) = 0$, else again y would have been labeled when we were scanning X, which is a contradiction.

Hence every edge from S to T is carrying as much flow as its capacity permits, and every edge from T to S is carrying no flow at all.

Hence $f(S, T) = \text{cap}\ (S, T)$ and $f(T, S) = 0$.
$\therefore$, At Min $Q = f(S, T) = \text{cap}\ (S, T)$ by the lemma 2.1.1.
Hence at Min $Q = \text{cap}\ (S, T)$
$\therefore$ The Minimum flow is equal to the maximum capacity of the cut (S, T).

Theorem 2.1.2. A flow f^* is a minimum flow if and only if the residual network $G_f^{\ *}$ contains no directed path from the source vertex to the sink vertex.

Proof: This theorem can be proved as the well known theorem, Augmenting Path Theorem

The minimum flow problem in a network can be solved in two phases:
(1) Establishing a feasible flow
(2) Establishing the minimum flow, from a given feasible flow.

Phase (1): Establishing a feasible flow

The problem of determining a feasible flow consists of finding a function $f : A \rightarrow R_+$ satisfying conditions (1a) and (1b). First, we transform this problem into a circulation problem by adding an arc (t, s) of infinite capacity and zero lower bound. This arc carries the flow sent from vertex s to vertex t back to vertex s. Clearly, the minimum flow problem admits a feasible flow if and only if the circulation problem admits a feasible circulation. Because these two problems are equivalent, we focus on finding a feasible circulation if it exists. The feasible circulation problem is to identify a flow f satisfying these following constraints:

$f(x, N) - f(N, x) = 0$; for every $x \in N$ (4a)

$l(x, y) \leq f(x, y) \leq c(x, y)$ for every edge $(x, y) \in A$ (4b)

By replacing $f(x, y) = f'(x, y) + l(x, y)$ in (4a) and (4b), we obtain the following transformed problem:

$f'(x, N) - f'(N, x) = b(x)$ for every $x \in N$ (5a)

$0 \leq f(x, y) \leq c(x, y) - l(x, y)$ for every edge $(x, y) \in A$ (5b)

With the supplies/demands $b(x)$ at the vertices defined by $b(x) = l(N, x) - l(x, N)$

$$\sum_{x \in N} b(x) = 0.$$

We can solve this feasible flow problem by solving a maximum flow problem defined in a transformed network. We introduce two new vertices: a source vertex s' and a sink vertex t'. For each vertex x with $b(x) > 0$ we add source edge (s', x) with capacity $b(x)$ and for each vertex x with $b(x) < 0$ we add sink arc (x, t') with capacity $-b(x)$. Then we solve a maximum flow problem in this transformed network. If the maximum flow saturates all the source and the sink arcs, then the initial problem has a feasible solution (which is the restriction of the maximum flow that saturates all the source and sink edges to the initial set of edges A); otherwise it is infeasible.

Phase(2): Establishing a minimum flow

There are two approaches for solving maximum flow problem: (1) using augmenting path algorithms and (2) using pre-flow push algorithms. From the first algorithms we can easily obtain algorithms for minimum flow by replacing augmenting paths with decreasing paths. The second type of algorithms for maximum flow does not permit such an easy transformation.

Let us say that an edge e of x is *burned* from u to v if $f(e)$ is above the capacity.

2.2 Generalized Minimum Flow problem

Definition 2.2.1

The generalized minimum flow problem is a *generalized network* $G = (V, E, t, c, \gamma, e)$ where V is an n-set of vertices, E is an m-set of directed edges, $t \in v$ is a distinguished vertex called the sink, $c : E \rightarrow R \geq 0$ is a capacity function, $\gamma : E \rightarrow R \geq 0$ is a *gain function*, and $e : V \rightarrow R \geq 0$ is an *initial deficit function*. A *residual edge* is an edge with negative capacity. A *flow generating cycle* is a cycle whose gain is more than one.

Definition 2.2.2

A *generalized pseudo flow* is a function $f : E \rightarrow R$ that satisfies the *capacity constraints* $f(v, w) \leq C(v, w)$ for all $(v, w) \in E$ and the *anti-symmetry constraints* $f(v, w) = -\gamma(w, v) f(w, v)$ for all $(v, w) \in E$. The *residual deficit* of f at vertex v is $e_f(v) = e(v) - \sum_{(v,w) \in E} f(v, w)$ i.e., the initial excess minus the net flow out of v. If $e_f(v)$ is negative we say that f has residual deficit at node v. A *generalized flow* is a generalized pseudo flow that has no residual excess, but it is allowed to have residual deficits. A *proper generalized flow* is a flow which does not generate any additional residual deficit, except possibly at the sink.

We note that a flow can be converted into a proper flow, by removing flow on useless paths and cycles. Let $OPT(G)$ denote the minimum possible value of any flow in network G. A flow f is *optimal* in network G if $|f| = OPT(G)$ and ξ - *optimal* if $|g| \geq (1 - \xi)\, OPT(G)$. The (*approximate*) *generalized minimum flow problem* is to find a (ξ-) optimal flow.

Optimality conditions

An *augmenting path* is a residual path from a node with residual deficit to the sink. A *generalized augmenting path* (GAP) is a residual flow-generating cycle, together with a (possibly trivial) residual path from a node on this cycle to the sink. By sending flow along augmenting paths or GAPs we increase the net flow into the sink.

Theorem 2.2.3 A flow f is optimal in network G if and only if there are no augmenting paths or GAPs in G_f.
Proof: Refer [3].

In the next section we present a generic algorithm for minimum flow and present an example of these algorithms.

3.Algorithm for minimum flow in networks

The generic algorithm for the minimum flow problem is the following.

Generic Algorithm;

```
Begin
    Let $f$ be a feasible flow in network $G$;
    Determine the residual network $G_f$;
    While $G_f$ contains a directed path from the source vertex $s$ to the sink vertex $t$ do
    Begin
    Identify a path $P$ from the source vertex $s$ to the sink vertex $t$;
        $f(x, y) = l(x, y) + \max\{\hat{r}(x, y) - c(y, x) + l(y, x), 0\}$
    Update the residual network $G_f$;
    End
End.
```

Theorem 3.1 (Integrality Theorem). If all arc capacities and lower bounds are integers and there exists a feasible flow, then the minimum flow problem has an integer minimum flow.

Proof: If all arc capacities and lower bounds are integers, the feasible flow, that is established as it is shown above by solving a maximum flow problem, is an integer flow.

We will prove the theorem by an induction argument applied to the number of decreases. Since the initial flow is an integer flow and all arc capacities and lower bounds are integers, the initial residual capacities are all integers. The flow decreased in any iteration equals the minimum residual capacity of the arcs, which form some directed path, which by the induction hypothesis is integer. Consequently, the residual capacities in the next iteration will again be integer.

Since the residual capacities $r(x, y)$, the lower bounds $l(x, y)$ and the arc capacities $c(x, y)$ are all integer, when we convert the residual capacities into flows by the method described previously, the arc flows $f(x, y)$ will be integer valued as well.

Since the residual capacities are integers, after each decrease the value of the flow becomes smaller with at least one unit. Since the value of the minimum flow cannot be smaller than the capacity of any cut, the algorithm will terminate in a finite number of iterations. □

Let V be the number of vertices and E be the number of edges of the flow network. Each iteration of the while loop takes $O(E)$ time. Then the above algorithms work with $O(EV)$ time complexity.

3.Example

We represent in below Figure the initial bipartite network $G' = (N'_1 \cup N'_2, A', l', c')$ with $N'_1 = \{2, 3, 4\}$ and $N'_2 = \{5, 6\}$.

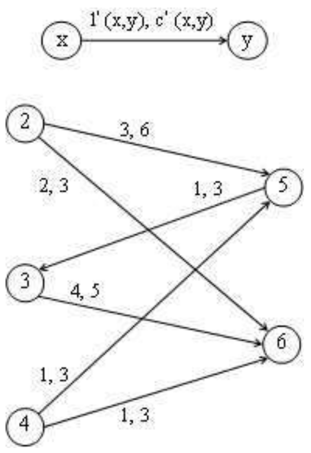


Figure 3. The initial bipartite network

The extended network is $G = (N_1 \cup N_2, A, l, c, s, t)$ with the source vertex s as 1, the sink vertex t as 7, $N_1 = \{2, 3, 4, 7\}$ and $N_2 = \{1, 5, 6\}$. This network is represented in Figure 4.The minimum flow in the extended network is in Figure 5.

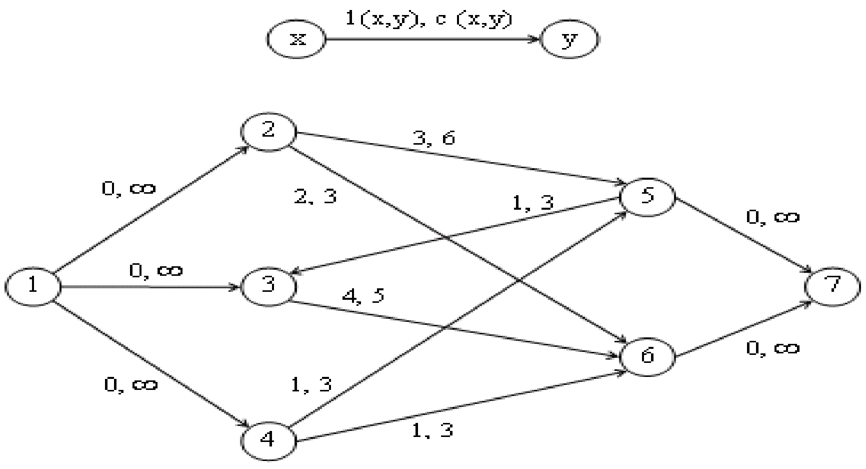


Figure 4. The extended network

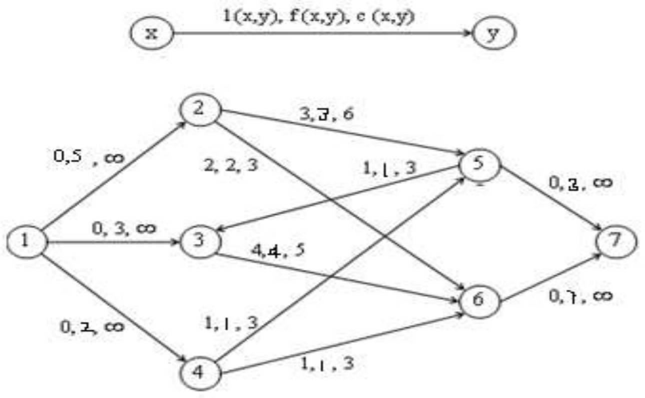


Figure 5. The minimum flow network

The value of minimum flow is $\hat{\upsilon} = 10$ and the maximum cut is $\left[\hat{S},\hat{T}\right] = \left(\hat{S},\hat{T}\right) \cup \left(\hat{T},\hat{S}\right)$ = {(2, 5), (2, 6), (3, 6), (4, 5), (4, 6)} ∪ {(5, 3)} with $l\left(\hat{S},\hat{T}\right) - C\left(\hat{T},\hat{S}\right) = 3 + 2 + 4 + 1 + 1 - 1 = 10$.

The minimum flow in the initial bipartite network from N_1' to N_2' is represented in Figure 6.

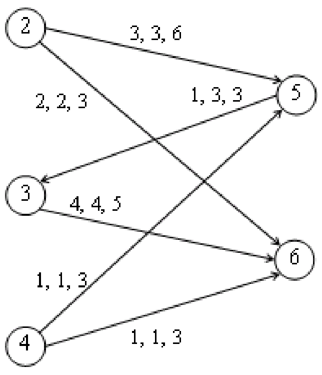


Figure 6. The final network flow

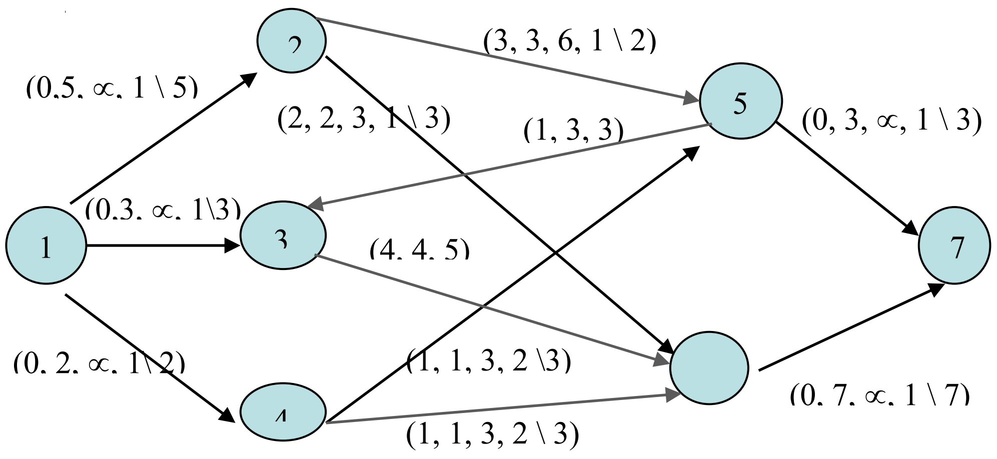
We shall present an algorithm sends flow along all gain augmenting paths simultaneously, using a maximum flow computation the algorithm is given below.

Algorithm 4.1

```
Procedure min flow (X: network; f: flow; gain function $\gamma : E \to R$: min - out-flow: real)
{Finds minimum out flow in a given network}
Min -out-flow value: = 0
Set $\gamma = 1 - \xi, where 0 < \xi < 1$
While there exists an augmenting path do
Min – out - flow value: = min – out - flow value + $\gamma$, where $\gamma = \sum\gamma(x) + \sum\gamma(y)$,
End {while}.
```

To find minimum out flow

Consider the network with gain function



In this network we have GAP:
F_1: 1 – 2 – 5 – 7 & 1 – 2 - 6 – 7 and F_2: 1 – 4 – 6 – 7& 1 - 4 – 5 - 7.
Let x and y be the gain function of the corresponding paths F_1 and F_2
In F_1, The gain function of the path 1 – 2 – 5 - 7 can be expressed as

$$x \to \frac{4x}{5} = 0.8x,\ \frac{4x}{5} \to \frac{4x}{5} * \frac{1}{2} * \frac{2}{3} = 0.8x * 0.3333$$

In the path 1 – 2 – 6 – 7, the expression becomes

$$x \to \frac{4x}{5} = 0.8x,\ \frac{4x}{5} \to \frac{4x}{5} * \frac{2}{3} * \frac{6}{7} = 0.8x * 0.5714$$

In F_2, The gain function of the 1 – 4 – 6 – 7 can be expressed as

$$y \to \frac{y}{2} = 0.5y,\ \frac{y}{2} \to \frac{y}{2} * \frac{1}{3} * \frac{6}{7} = 0.5y * 0.2857$$

Maximum out flow = sum of the gain function of x + sum of the gain function of y.

<u>Iteration 1</u>

When x = 1, Augment flow along (1, 2, 5, 7)

The gain function $\frac{4x}{5} = 0.8x$

0.8x * 0.3333 = 0.2664

Augment flow along (1, 2, 6, 7)

0.8x * 0.5714 = 0.4517

When y = 1, Augment flow along (1, 4, 6, 7)

The gain function $\frac{y}{2} = 0.5y$

0.5y * 0.2857 = 0.1429

Augment flow along (1, 4, 5, 7), 0.5y * 0.2222 = 0.1111

Minimum out flow = 0.2664 + 0.4571 + 0.1429 + 0.1111 = 0.9775

<u>Iteration 2</u>

When x = 2, Augment flow along (1, 2, 5, 7)

The gain function $\frac{4x}{5} = 0.8 * 2 = 1.6$

1.6 * 0.3333 = 0.528

Augment flow along (1, 2, 6, 7)

1.6 * 0.5714 = 0.9142

When y = 2, Augment flow along (1, 4, 6, 7)

The gain function $\frac{y}{2} = 0.5y = 0.5 * 2 = 1$

1 * 0.2857 = 0.2857

Augment flow along (1, 4, 5, 7)

1 * 0.2222 = 0.2222

Minimum out flow = 0.528 + 0.9142+ 0.2857 + 0.2222 = 1.9501

Proceeding like this, we will increase the value of x and y, up to x = 6 and add with y = 3. Since the flow value cannot exceed the capacity.

Result: Minimum in flow = 10
Minimum out flow = 5

5. Applications

In traditional networks, there is an implicit assumption that flow is conserved on every arc. Many practical applications violate this conservation assumption. The gain factors can represent physical transformations of one commodity into a lesser or greater amount of the same commodity. Some examples include: spoilage, theft, evaporation, taxes, seepage, deterioration, interest, or breeding. The gain factors can also model the transformation of one commodity into a different commodity. Some examples include: converting raw materials into finished goods, currency conversion, and machine scheduling. We explain the latter two examples next.

Currency Conversion

We use the currency conversion problem as an example of the types of problems that can be modeled using generalized flows. Later, we will use this problem to gain intuition. In the currency conversion problem, the goal is to take advantage of discrepancies in currency conversion rates. Given certain amount of one currency, say 1000 U.S dollars, the goal is to convert it into the maximum amount of another currency, say French Francs, through a sequence of currency conversions. We assume that limited amounts of currency can be treated without affecting the exchange rates.

Scheduling unrelated parallel machines

As a second example, we consider the problem of scheduling N jobs to run on M unrelated machines. The goal is to schedule all of the jobs by a pre specified time T. Each job must be assigned to exactly one machine. Each machine can process any of the jobs, but at most one job at a time. Machine i requires a pre-specified amount of time P_{ij} to process job j.

6. Conclusion

In this paper we described the generic algorithm for generalized minimum flow. This algorithm is a special implementation of the algorithm developed by Ciurea and Ciupala in [4]. Our new algorithm runs in O (EV) time, which is subsequently better than the running time of generic pre-flow algorithm.
The following table summarizes the algorithms for determining minimum flow and their complexities.

Algorithm for minimum flow	Running time
Generic pre-flow algorithm	O (E^2V)
FIFO pre-flow algorithm	O (E^2)
Deficit scaling algorithm	O ($EV + E^2 \log C$)
Highest-label pre-flow algorithm	O ($E^2V^{1/2}$)

7. References

1. R. Ahuja, T. Magnanti and J. Orlin, Network Flows: Theory, Algorithms and Applications Prentice Hall, Inc., Englewood Cliffs, NY, 1993.
2. E. Ciurea and L. Ciupala, Algorithms for minimum flows, Computer Science Journal of Moldova 9, No.3 (27), 2001, pp. 275-290.
3. E. Ciurea and L. Ciupala, Sequential and parallel algorithms for minimum flows, Journal of Applied Mathematics and Computing, Vol. 15, No. 1-2, 2004, pp. 53–75.

4. E. Ciurea, O. Georgescu and D. Marinescu, Improved Algorithms for minimum flows in bipartite networks, International Journal of Computers, issue 4, volume 15, 2008, 351-360.
5. H. S. Wolf, Algorithms and Complexity, Prentice Hall International, Inc., U. S. A. 1986.
6. A. Anto Kinsley and B. Uma Maheswari, Design of algorithms to Generalized Maximum flow network Problem, International Journal of Advanced Research in Computer Science and Technology(IJARCST 2016) Vol.4,Issue 1 (Jan. – Mar 2016) pp:31- 34.
7. A. Anto Kinsley and B. Uma Maheswari, A Pre-Flow Push algorithm to Generalsed Maximum flow Problem International Journal of Engineering Science and Research Technology (IJESRT) Feb. 2016 pp:547-553.