Solving maximum flows by proportional flow augmentations on layered networks

I-Lin Wang[1*] and Chen-Hsien Lee[1]

[1]Department of Industrial and Information Management

National Cheng Kung University

No.1 University Rd. Tainan, Taiwan 701

[*]ilinwang@mail.ncku.edu.tw

Abstract

The *s-t* maximum flow problem seeks the maximum possible flow sent from a source node *s* to a sink node *t* in a given capacitated network. Conventional maximum flow algorithms either augment flows via simple paths or ship flows from nodes with excess flows via admissible arcs to their neighbors. In this paper, we propose a new polynomial-time maximum flow algorithm called proportional arc augmenting (PAA) algorithm with the intuition of augmenting flows based on the relative proportions of the residual capacities between each arcs emanating from the same node in layered networks which are composed by all the *s-t* shortest paths. PAA will augment fractional flows over all the admissible arcs in the admissible subnetworks proportionally without violating the flow balance and capacity constraints. Our computational results indicate PAA may outperform existing maximum flow algorithms in practice for several classes of problems.

Keywords: maximum flow, layered network, proportional augmenting, polynomial time algorithm

1. Introduction

The *s-t* maximum flow problem is a fundamental network optimization problem which finds the maximum value of flows in a capacitated network between the source node *s* and the sink node *t*. Conventional maximum flow algorithms (see [1] for details) either augment flows via simple paths or ship flows from nodes with excess flows via admissible arcs to their neighbors.

The first augmenting path algorithm was proposed by Ford and Fulkerson [7] which iteratively augments flows along a path in the residual network until no more augmenting paths exist. This algorithm runs in pseudo-polynomial time. Edmonds and Karp [6] modified Fold-Fulkerson algorithm that iteratively finds the shortest augmenting path to augment flows. Dinic [4] creates an $O(n)$ sequence of structured *s-t* shortest path networks, called *layered networks*, and solves the maximum flow problem for each layered network. Since these two algorithms guarantee to saturate one arc in the worst case, the running time of these algorithms can be bounded in polynomial time.

Karzanov [13] first introduced the concept of preflow-push and combined it with layered network to improve Dinic's algorithm. Goldberg and Tarjan [11] proposed a better preflow-push algorithm which combined the ideas of preflows and node distance labels. The distance label associated with a node represents a lower bound on the number of arcs from it to *t* in the residual network.

Recently, Fujishige [8] proposed a maximum adjacency (MA) ordering max-flow algorithm to augment more flows in an augmenting subnetwork than any single augmenting path in the residual network. Later, Matsuoka and Fujishige [14] proposed a modified MA ordering algorithm based on preflow-push. Their algorithm outperforms the preflow-push algorithm by Goldberg-Tarjan [11] on some special families of maximum flow problems.

Chang [2] proposed a least-squares dual-primal (LSDP) algorithm for solving the maximum flow problem without any degenerate pivots. In LSDP, many phase-I nonnegative least-squares (NNLS) subproblems are iteratively solved for improving the flows nondegenerately. Instead of solving the quadratic programming problem (NNLS) directly, LSDP treats NNLS subproblems as electronic circuits and solves for the best improving fractional flows via Kirchhoff's laws. The complexity for LSDP is still not analyzed yet.

In this paper, we propose a new polynomial-time maximum flow algorithm called proportional arc augmenting algorithm with intuitions of augmenting flows based on the relative proportions of the residual capacities between each arcs emanating from the same node in layered networks.

Similar to Dinic's algorithm, PAA augments flows on layered networks. Instead of augmenting integral flows via single *s-t* shortest paths as conventional maximum flow algorithms in each iteration, PAA augments fractional flows along all the arcs at the same time in an admissible subnetwork. In particular, conventional augmenting-path based algorithms only seek single augmenting path and ship integral flows according to the bottleneck arcs in that path, so that at least one arc will be saturated at each iteration. On the other hand, the amount of flows shipped by PAA for each arc is determined proportionally based on its residual capacity so that more flows may be augmented along those arcs with larger residual capacities. As a result, at least one node will be saturated since all of its outgoing arcs will be saturated at the same time due to the proportional augmentation of flows. After saturating the layered network, PAA constructs another layered network to augment flows. PAA repeats these procedures until no more layered network can be further constructed, which means there exists no *s-t* augmenting path.The rest of this paper is organized as follows. Section II gives fundamental definitions and backgrounds used in this paper. Section III describes details of our proposed algorithm PAA. In section IV we illustrate how PAA works by a small example. Section V shows computational results. Section VI concludes the paper and gives suggestions for future research.

2. Preliminaries

Let $G = (N, A, s, t, u)$ be a directed network where N is the node set with $|N| = n$; A is the arc set with $|A|= m$; s and t denote the source and sink nodes, respectively; and u is the arc capacity vector whose element is a nonnegative integer value u_{ij} for each arc (i, j)□A.

A flow is a real function $X : A \rightarrow R$ that satisfies the node flow balance constraint (1) and the arc capacity constraint (2).

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} = 0 \ \forall i \in N - \{s,t\} \qquad (1)$$

$$0 \le x_{ij} \le u_{ij} \ \ \forall (i,j) \in A \qquad (2)$$

Given a feasible flow vector x, a residual network $G(x) = (N(x), A(x), s, t, r(x))$ can be constructed as follows: for each arc (i, j) with flow x_{ij}, we replace (i, j) by two arcs, (i, j) and (j,i) with *residual capacities* $r_{ij}(x) = u_{ij} - x_{ij}$ and $r_{ji}(x) = x_{ij}$, respectively. An arc $(i, j) \in A(x)$ is *saturated* when $r_{ij}(x) = 0$. The residual network only contains arcs with positive residual capacities.

For each node $i \in N(x)$, we define its distance label $d(i)$ as the lower bound on the number of arcs connecting to it from the source node s in $G(x)$. Note that this definition is different from the conventional one used in literature for measuring the minimum number of arcs connecting to *t*. We can conduct a breath-first search (BFS) on *G*(*x*) starting from *s* to calculate the distance labels, denoted by *d*(*i*) for each node *i* in *N*(*x*), as shown in Figure 1(a). Then, we can define the admissible arcs set, denoted by *A'*(*x*), as the set of arcs (i, j) in *A*(*x*) satisfying *d*(*j*) = *d*(*i*) + 1 and r_{ij} > 0. A layered network, denoted by $G'(x) = (N', A', s, t, r')$, is an admissible subnetwork of $G(x)$ that contains all the admissible arcs A' of $A(x)$ with residual capacity r', and their associated node set N'. Figure 1(b) shows the corresponding layered network of Figure 1(a).

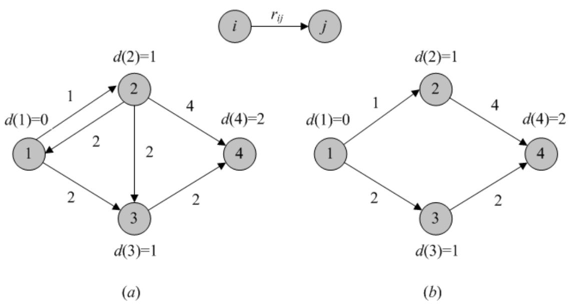


Fig 1. An example of residual network and its corresponding layered

Detailed procedures of our proposed PAA algorithm will be given in next section.

3. Proportional Arc Augmenting Algorithm

Our proposed PAA algorithm exploits similar algorithmic framework of Dinic's algorithm. In particular, in each outer iteration, a layered network $G'(x)$ is constructed and solved for its maximum flow, until no more layered network can be constructed. To solve the maximum flow for each layered network, Dinic's algorithm seeks an augmenting-flow vector, denoted by $\delta=[\delta_{ij}]$ for each admissible arc $(i, j) \in A'$, to augment flows along single augmenting path *p* at each inner iteration. In other words, δ_{ij} =1 for each $(i, j) \in p$ and δ_{ij} =0 for each $(i, j) \in A' - p$. On the other hand, the augmenting-flow vector by PAA guarantees δ_{ij} >0 for each $(i, j) \in A'$. In a sense, PAA may tend to ship more flows than conventional augmenting-path based algorithms at each inner iteration, since all the admissible arcs are used to ship flows rather than via only those admissible arcs on a single path.

Let *b*(*i*) be the sum of flows collected from all the incoming arcs of node *i*, which then will be distributed to all the outgoing arcs of node *i*; *S*(*i*) be the sum of residual capacities associated with all the admissible arcs emanating from node *i*; $\delta=[\delta_{ij}]$ be a unit augmenting vector with |*A'*| elements, where each element δ_{ij} corresponds to a unit flow improving direction associated with admissible arc (*i*, *j*); and θ be the maximum step length to augment flows along all arcs without violating the arc capacity constraints.

Procedure *Construct-layered-net*($\bar{G}$, *d, BO, IBO*)

```
Let $\bar{G} = (\bar{N}, \bar{A}, s, t, r)$, set $d(s) := 0$, $BO(s) := 1$
  and $IBO(1) := s$, set $d(i) := \infty$, $BO(i) := 0$ for
  each node $i \in \bar{N} - \{s\}$ and $IBO(k) := 0$ for
  $k = 2, \ldots, |N'|$;
Conduct BFS on $\bar{G}$ from $s$ to $t$, calculate $d(i)$
  for each node $i \in \bar{N} - \{s\}$, update $BO(i)$ for each
  node $i$ that is reachable from $s$ and can reach $t$,
  and update $IBO(\cdot)$;
$\tilde{N} := \{i : d(i) \neq \infty, i \in \bar{N}\}; \tilde{A} := \varnothing;$
For each arc $(i, j) \in \bar{A}$ do
    If $d(j) = d(i) + 1$ and $r_{ij} > 0$ then
        $\tilde{A} = \tilde{A} \cup \{(i, j)\}$; $\tilde{A} = \tilde{A} \cup \{(i, j)\}$;
Return $\tilde{G} = (\tilde{N}, \tilde{A}, s, t, r)$;
```

Fig 2. Pseudo-code for constructing a layered network

First we perform a BFS to construct a layered network $G'(x) = (N', A', s, t, r')$, where each node in *N'* can be reachable from *s* and also can reach *t* with the least number of admissible arcs. In order to efficiently calculate the augmenting-flow vector δ, we need to scan nodes according to a BFS order. To this end, we define two arrays of size $|N'|$, $BO(\cdot)$ and $IBO(\cdot)$, to record the BFS order for each node and the node index encountered by this order (i.e. $IBO(\cdot)$ represents the inverse of $BO(\cdot)$), starting from $s=IBO(1)$. In particular, suppose node i is the kth node in $G'(x)$ searched by BFS, then we will have $BO(i)=k$ and $IBO(k)=i$.

Note that $BO(\cdot)$ represents a topological order sequence since a layered network is acyclic. Figure 2 illustrates the procedure to construct a layered network. If the BFS can not reach t, which means there exists no admissible subnetwork to augment flow from s to t, and thus we terminate PAA.

```
Procedure Calculate-augmenting-vector (G'(x),IBO)
b(s) := 1 and order := 1;
While $i \neq t$ do
        $i = IBO(order)$;
        $S(i) := \sum_{(i,j)\in A'} r_{ij}$;
        $\delta_{ij} := b(i)\cdot r_{ij} / S(i)$ and $b(j) := b(j) + \delta_{ij}$;
        order := order + 1;
End while;
Return $\delta$;
```

Fig 3. Pseudo-code for calculating the augmenting vector

After a successful BFS, we calculate δ by first setting $b(s)=1$, selecting a node i by $IBO(k)$ for each $k=1,\ldots,|N'|$, distributing $b(i)$ along all the admissible arcs emanating from node i according to the relative proportion of their residual capacities r_{ij} to their head nodes j's (i.e. calculating. $\delta_{ij} := b(i)\cdot r_{ij} / S(i)$), updating $b(j) := b(j) + \delta_{ij}$, and then repeat these procedures until all nodes in $IBO(\cdot)$ have been processed. Note that our augmenting mechanism augments flows along all admissible arcs in a layered network, while conventional augmenting-path based algorithms only augment flow along a single path. Figure 3 shows the pseudo-code for calculating the proportional flow augmentation vector δ.

```
Procedure Augment-flow(G'(x), $\theta$, $\delta$, $N_{saturated}$)
For each arc $(i,j) \in A'$ do
    If arc $(i,j) \in A$ then
        $x_{ij} := x_{ij} + \delta_{ij}\times\theta$;
        If $x_{ij} = u_{ij}$ then add node $i$ to $N_{saturated}$;
    Else then
        $x_{ij} := x_{ij} - \delta_{ij}\times\theta$;
        If $x_{ij} = 0$ then add node $j$ to $N_{saturated}$;
```

Fig 4. Pseudo-code for augmenting flows

After calculating the augmenting vector δ, the maximum step length θ can be calculated by $\min_{(i,j)\in A'}\{r_{ij}/\delta_{ij}\}$ so that θ units of flows can be augmented from s to t via all the (admissible) arcs in the layered network, while the capacity constraints are still satisfied. Figure 4 lists the procedure to augment flows.

Suppose θ is determined by a bottleneck arc $(i^*, j^*) = \arg\min_{(i,j)\in A'}\{r_{ij}/\delta_{ij}\}$, it is easy to show that

$r_{i^*j^*}/\delta_{i^*j^*} \quad = r_{i^*j^*}/(b(i^*)\cdot r_{i^*j^*}/S(i^*)) \quad = S(i^*)/b(i^*)$

which equals to r_{i^*j}/δ_{i^*j} for any other arc $(i^*, j)\in A'$ also emanating from node i^*. In other words, not only will arc (i^*, j^*) be saturated, all the other arcs $(i^*, j)\in A'$ emanating from node i^* will be saturated as well. Moreover, one may directly calculate θ by $\min_{i\in N'}\{S(i)/b(i)\}$ in $O(|N'|)$ time, instead of $\min_{(i,j)\in p}\{r_{ij}\}$ for some augmenting path p in $O(|p|)$ time, to identify at least a bottleneck node i^* to be removed from the layered network. Although $|p| \le |N'|$, the amount of flow to be augmented by PAA tends to be (but not necessarily) greater than which by conventional augmenting-path based algorithms. The intuition is that PAA exploits the capacities of all the admissible arcs, while conventional augmenting-path based algorithms only use capacities along a single path.

There are cases where PAA ships more flows than conventional augmenting-path based algorithms. For example, for the networks in which all arcs have the same capacities, starting with the same initial flow, PAA can augment flows no less than Dinic's algorithm in next iteration. Furthermore, if all the arcs in the augmenting path identified by Dinic's algorithm are not bridges, then PAA guarantees to augment more flows than Dinic's algorithm in this iteration.

```
Procedure Update-layered-net
($\bar{G}$, $N_{saturated}$, BO, IBO)
  Let $\bar{G} = (\bar{N}, \bar{A}, s, t, r)$, recursively remove those
    nodes in $N_{saturated}$ from $\bar{N}$, their associated
    arcs, and other nodes without any outgoing
    arcs, until the remaining nodes in $\bar{N}$ can
    reach t and can be reachable from s.
  Conduct BFS on $\bar{G}$ from s to t, update
    BO(i) for each node i that is reachable from s
    and can reach t;
  Return $\tilde{G} = (\tilde{N}, \tilde{A}, s, t, r)$;
```

Fig 5. Pseudo-code for updating a layered network

After augmenting the flow vector x to $x + \theta\delta$, PAA updates the layered network by the procedure Update-layered-net(*G'*(*x*), $N_{saturated}$, *d*, *BO*, *IBO*) as shown in Figure 5. This procedure is similar to the procedure of constructing a layered network (see Figure 1 for details), but without calculating the distance labels. In particular, this procedure will remove those nodes in *N'* that can not reach *t* or can not be reachable from *s*, and then re-calculate the BFS traversal order *BO*(•) and its inverse *IBO*(•) for the remaining nodes in *N'*.

The pseudo-code of algorithm PAA is described in Figure 6. In particular, PAA starts by constructing an initial layered network *G'*(*x*). As long *G'*(*x*) is connected from *s* to *t* (i.e. $d(t) \neq \infty$), it will iteratively calculates an augmenting-flow vector δ for all admissible arcs in *G'*(*x*), calculates the maximum step length θ, augments flow *x*, and updates *G'*(*x*), until *G'*(*x*) becomes disconnected from *s* to *t* (i.e. $d(t) = \infty$ or $BO(t)=0$) and then it constructs a new layered network. These procedures repeat until no more layered network can be constructed to connect *s* and *t*, which means the maximum flow for the original network has been achieved.

```
Algorithm Proportional-arc-augmenting(G)
Initialization:
    Set $d(s) := 0$, $d(i) := \infty$ for each node $i \in N - \{s\}$;
    Set $x_{ij} := 0$ for each arc $(i, j) \in A$;
    G'(x)=Construct-layered-net(G(x), d, BO, IBO);
While $d(t) \neq \infty$ do
    δ=Calculate-augmenting-vector(G'(x),IBO);
    $\theta := \min_{i \in N'} \{S(i)/b(i)\}$;
    x=Augmenting-flow(G'(x), θ, δ, $N_{saturated}$);
    G'(x)=Update-layered-net(G'(x), $N_{saturated}$,BO, IBO);
    If $s \notin N'$ or $t \notin N'$ then
        G'(x)=Construct-layered-net(G(x), d, BO, IBO);
End while;
```

Fig 6. Proportional-arc-Augmenting algorithm

The complexity for procedures *Construct-layered-net*, *Calculate-augmenting-vector*, *Augmenting-flow*, and *Update-layered-net* all take $O(|A'|) = O(m)$ time, since all of them have to scan each admissible arc at most constant number of times. Calculating θ takes $O(|N'|) = O(n)$ time. The following lemma shows that the maximum value in a layered network can be calculated in *O*(*nm*) time.

Lemma 1. *Calculating the maximum flows for* *each layered network by PAA takes O(nm) time.*

Proof. When we calculate the maximum step length θ, at least one node will be saturated and removed. Since no nodes will be added to $G'(x)$ at any latter iteration, $G'(x)$ becomes disconnected from s to t in $O(n)$ iterations. Since all other procedures in each iteration take $O(m)$ time, thus the maximum flows for each layered network by PAA takes $O(nm)$ *time.*

Similar to Dinic's algorithm which iteratively solves a layered network until no more layered network exists, algorithm PAA also terminates when there exists no more layered network. Note that $d(t)$, the distance label for the sink node, will increase at least one after constructing a new layered network, and $d(t) \le n-1$ since no simple path connecting s to t contains more than n-1 arcs. Hence, we can conclude that the PAA runs in $O(n(nm))=O(n^2m)$ time, the same as Dinic's algorithm.

Theorem 2. The proportional arc augmenting algorithm runs in $O(n^2m)$ time.

4. An Illustrate Example

Here we give an example to illustrate how PAA solves a max- flow problem. In Figure 7(a), we would like to ship as much as flow as possible from node 1 to 6. Thus $N = \{1,2,3,4,5,6\}$, $A = \{(1,2), (1,3), (2,3), (2,4), (2,5), (3,4), (3,5), (4,6), (5,4), (5,6)\}$, and $u = (9,6,2,7,1,2,6,9,5,17)$.

First, we conduct a BFS to calculate the distance labels d = (0,1,1,2,2,3) and store the topological order BO = (1,2,3,4,5,6), as listed in Figure 7(b).

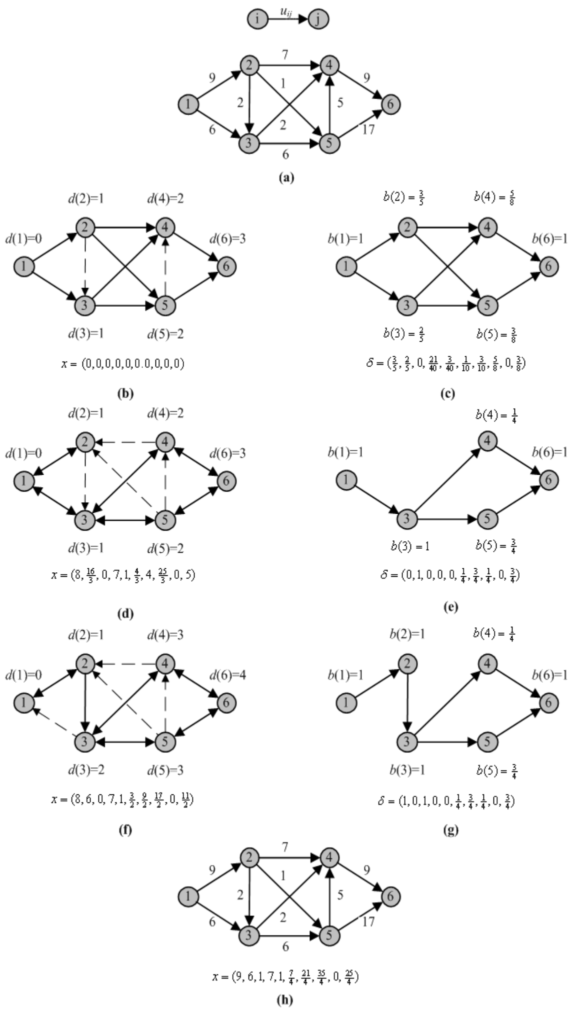


Fig 7. A max-flow problem solving by PAA

Based on distance labels, a layered network in Figure 7(c) is constructed. We first calculate $S(i)$, the sum of residual capacities from each node i in the order of BO, and get S = (15,8,8,9,17). Let $b(1)$ = 1, starting from node $i=IBO(k)$, for $k=1,\ldots,|N'|$, we calculate the proportion of outgoing admissible arcs δ_{12} = 9/15 and δ_{13}=6/15 and then add these two values to $b(2)$ and $b(3)$, respectively. Then we can calculate b = (1,3/5,2/5,5/8,3/8,1) and δ = (5/5, 2/5, 0, 21/40, 3/40, 1/10, 3/10, 5/8, 0, 3/8) as illustrate in Figure 7(c). The maximum step length can be calculated by θ = 40/3 to update flow x = (8,16/3,0,7,1,4/3,4,25/3,0,5) in Figure 7(d). Since all the outgoing admissible arcs of node 2 are saturated, node 2 can be removed from the layered network.

The algorithm continues to solve for δ = (0, 1, 0, 0, 0, 1/4, 3/4, 1/4, 0, 3/4) and θ = 2/3, then we calculate a better feasible flow x = (8,6,0,7,1,3/2,9/2,17/2,0,11/2) as illustrate in Figure 7(e) and (f), and then node 1 is removed which disconnected the current layered network.

Next, we recalculate the distance labels and construct a new layered network as illustrate in Figure 7(g). We then calculate δ = (1,0,1,0,0,1/4,3/4,1/4,0,3/4) and a better feasible flow x = (9,6,1,7,1,7/4,21/4,35/4,0,25/4) by the maximum step length $\theta = 1$. After removing node 1, the layered network becomes disconnected from s to t. We then recalculate the distance labels but still can not connect s to t. Thus we terminate PAA and the optimality is attained.

Note that PAA augment fractional flows, rather than the integral flows as conventional maximum flow algorithms. If maximum integral flows are required, we can conduct another procedure similar to flow decomposition algorithm to convert all the fractional arc flows into integral arc flows within $O(nm)$ time. PAA

5. Computational Results

In this section, we compared the computational efficiency of four maximum flow algorithms. All the testings were conducted on an Intel personal computer with Inter Core 2 Quad 2.40GHz CPU and 4GB RAM that runs Microsoft Windows Vista Enterprise. All the algorithms are written in *C* language and complied by *gcc* using the –O3 optimization option. Four maximum flow algorithms are tested: (1) PAA; (2) Dinic's algorithm [4] implemented by [3, 9], denoted by DF; (3) preflow MA Ordering algorithm [14], denoted by FMAP; and (4) highest-label-first implementation [9] of the preflow-push algorithm [11], denoted by HI_PR. We record the running time for all algorithms in seconds, without considering the time spent for processing the input and output.

We tested five problem families (GENRMF-LONG, GENRMF-WIDE, AK, WASHINGTON-10, and ACYCLIC-DENSE) created by four maximum flow problem generators: GENRMF, WASHINGTON, AK, and ACYCLIC. These generators can be downloaded from the DIMACS website [15] or Goldberg's personal website [10].

The GENRMF network families contain b grid frames of $a \times a$ nodes. Each node in a grid frame is connected to its neighbors by an arc and each node in a frame and its next frame is randomly connected by an arc. The number of nodes is a^2b and total number of arcs is $5a^2b - 4ab - a^2$. The capacities of arcs inside frames are uniformly at random chosen from the range $[c_1, c_2]$ and arcs between frames are $c_2 \times a \times a$, respectively. The source and sink nodes are located in the first and last frame, respectively. Depending on different settings of a and b, we have tested two GENRMF families: GENRMF-LONG and GENRMF-WIDE, where $a = 2^{k_1/4}$, $b = 2^{k_1/2}$, $c_1 = 1$, $c_2 = 100$ for GENRMF-LONG family and $a = 2^{2k_1/5}$, $b = 2^{k_1/5}$, $c_1 = 1$, $c_2 = 100$ for GENRMF-WIDE family for different k_1's.

The WASHINGTON network family we use is set by the WASHINGTON generator with function 10 (see [12] for its detailed settings), which contains $3k_2 - 1$ nodes and $4k_2 + 1$ arcs for different k_2's. The AK network family generated by the AK generator (see [3] for its detailed settings) is composed by two parallel subnetworks with $4k_3 + 6$ nodes and $6k_3 + 7$ arcs for different k_3's. The ACYCLIC-DENSE network family by ACYCLIC generator (see [3, 12] for its detailed settings) which contains 2^{k_4} nodes where each node connects to all other nodes with larger indices by equal-capacity arcs.

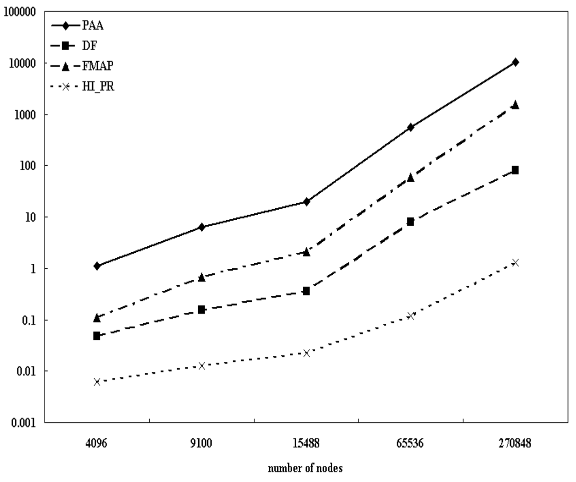
The results of computational testings are illustrated in Figure 6, 7, 8, 9, and 10 for network families of GENRMF-LONG, GENRMF-WIDE, AK, WASHINGTON-10, and ACYCLIC-DENSE, respectively. For each problem family, we generate five groups of 10 random cases, where each group represents cases of similar sizes. For each random case, we solve its maximum flow by each of the four algorithms (i.e. PAA, DF, FMAP, and HI_PR). Then we record the average running time for each group and each algorithm, and plot the results in the figures.

Figure 6 and Figure 7 indicate that HI_PR and DF are more efficient, PAA takes much more time for most cases except large cases of GENRMF-WIDE where FMAP performs the worst. Figure 8 shows that FMAP is the most efficient algorithm and PAA still takes much more time than others for AK networks. However, the performance of PAA in AK networks is already better than its performance in solving GENRMF networks.

Figure 9 shows PAA outperforms FMAP and performs almost the same as HI_PR for the WASHINGTON-10 networks. With the special structure of WASHINGTON-10 networks, PAA terminates within one iteration. In particular, it only constructs a single layered network in $O(m)$ time, augments flows once which takes $O(3m)$ time, and then terminates. Thus the theoretical complexity for PAA in solving WASHINGTON-10 networks can be reduced to $O(m)$ time. This explains why PAA almost has the best running time (similar to HI_PR) for solving WASHINGTON-10 networks. On the other hand, FMAP performs the worst in all cases of this network family.

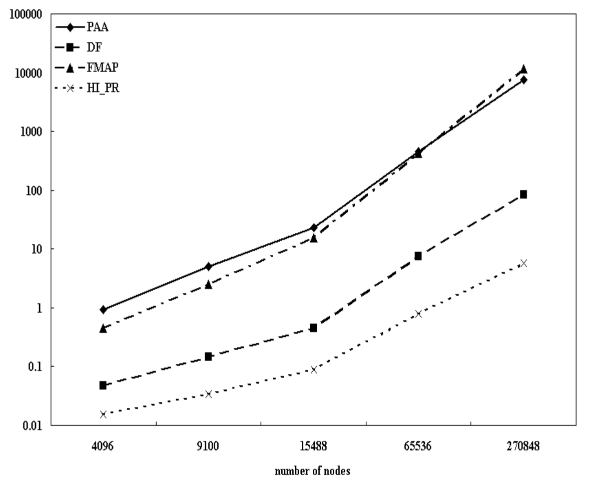
Figure 10 shows results for the ACYCLIC-DENSE where all arcs have the same capacity in each case. PAA performs a little slower than the others. Note that for this specific network family, PAA takes $O(m)$ time since it suffices to perform two iterations of flow augmentation to calculate the maximum flow.

In summary, PAA is not as competitive as conventional maximum flow algorithms, based on the preliminary computational experiments conducted in this paper. Here we suspect the cause of its inefficiency may be due to the calculation of improving vectors which scans all the admissible arcs to saturate a node, while conventional augmenting-path based algorithms can saturate an arc very efficiently. As a result, the constant associated with $O(nm)$ to saturate a layered network by PAA may be larger than which by conventional augmenting-path based algorithms. More analyses are required to draw more solid conclusions.



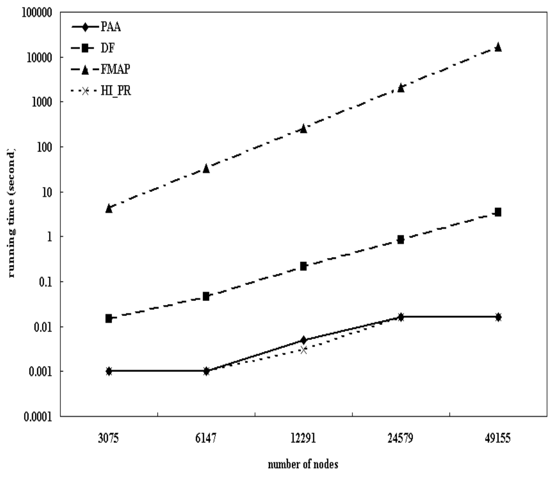
Nodes	Arcs	Running time (second)			
		PAA	DF	FMAP	HI_PR
4096	18368	1.1076	0.047	0.1094	0.006
9100	41760	6.3744	0.1528	0.6742	0.013
15488	71687	19.9646	0.3588	2.0968	0.022
65536	311040	560.7392	7.8876	58.4564	0.119
270848	1306607	10095.963	79.732	1566.478	1.27

Fig 6. GENRMF-LONG generator result



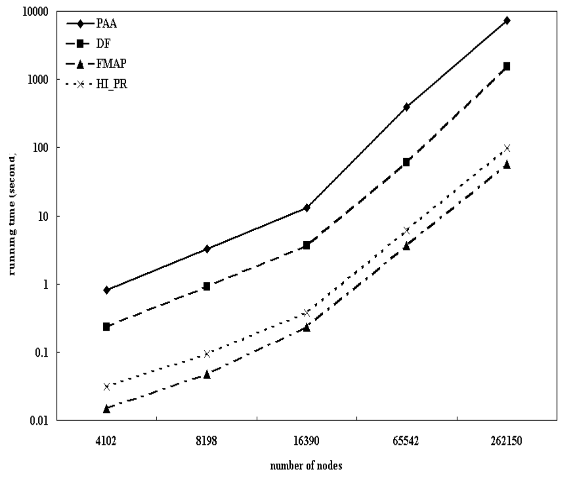
Nodes	Arcs	Running time (second)			
		PAA	DF	FMAP	HI_PR
4096	18368	0.9266	0.047	0.4554	0.015
9100	41760	5.0416	0.1462	2.4776	0.034
15488	71687	22.8446	0.4432	15.6092	0.087
65536	311040	451.4606	7.4504	421.764	0.78
270848	1306607	7621.252	85.303	11623.981	5.701

Fig 7. GENRMF-WIDE generator result



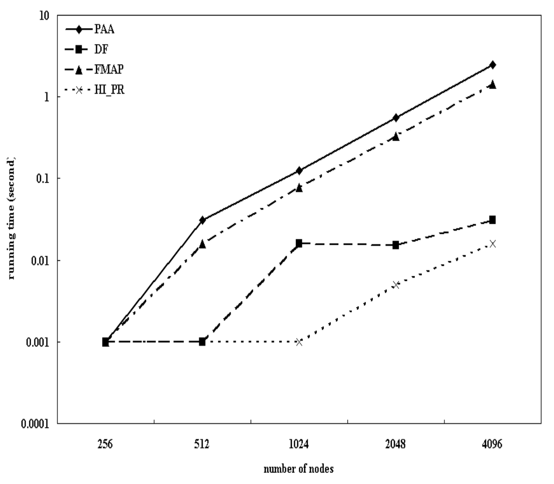
Nodes	Arcs	Running time (second)			
		PAA	DF	FMAP	HI_PR
3075	4097	0.001	0.015	4.258	0.001
6147	8193	0.001	0.047	33.26	0.001
12291	16385	0.005	0.218	262.799	0.003
24579	32769	0.016	0.858	2088.853	0.016
49155	65537	0.016	3.447	16676.881	0.016

Fig 9. WASHINGTON-10 generator result



Nodes	Arcs	Running time (second)			
		PAA	DF	FMAP	HI_PR
4102	6151	0.811	0.234	0.015	0.031
8198	12295	3.26	0.905	0.047	0.094
16390	24583	13.276	3.603	0.234	0.374
65542	98311	397.412	60.326	3.619	6.068
262150	393223	7246.309	1516.0	57.579	97.173

Fig 8. AK generator result



Nodes	Arcs	Running time (second)			
		PAA	DF	FMAP	HI_PR
256	36240	0.001	0.001	0.001	0.001
512	130816	0.031	0.001	0.016	0.001
1024	523776	0.125	0.016	0.078	0.001
2048	2096128	0.561	0.015	0.328	0.005
4096	8386560	2.481	0.031	1.436	0.016

Fig 10. ACYCLIC-DENSE generator with same capacity result

6. Conclusions

Unlike most maximum flow algorithms that saturate an arc at each iteration, our proposed PAA algorithm saturates a node at each iteration by augmenting flows in a proportional way based on the relative proportions of residual capacities for arcs emanating from each node in a layered network. Although PAA takes $O(n^2m)$ time, which is the same as Dinic's algorithm, its empirical efficiency is not as good, based on our preliminary computational experiments. On the other hand, we find that PAA may be suitable for networks of symmetric topology or arc capacities such as the WASHINGTON-10 or ACYCLIC-DENSE networks, where PAA can terminate within very few iterations.

Despite the discouraging computational performance of PAA in our testings, we still encourage researchers to further investigate ways to improve the computational efficiency of PAA. To this end, we suggest to investigate better design of data structures to prevent repeated arc scans, or techniques such as scaling phase to avoid small augmentations.

Acknowledgment

I-Lin Wang is partly supported by NSC under the grant NSC 97-2221-E-006-173.

References

1. R. K. Ahuja and J. B. Orlin, "Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems", *Naval Research Logistics*, vol. 38, pp. 413-430, 1991.
2. J. H. Chang, "Solving the network flow problem by a least-squares primal-dual method", Unpublished master's thesis, National Chen Kung University.
3. B. V. Cherkassky and A. V. Goldberg, "On Implementing the Push-Relabel Method for the Maximum Flow Problem", *Algorithmica*, vol. 19, pp. 390-410, 1997.
4. E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation", *Soviet Mathematics Doklady*, vol. 11, pp. 1277-1280, 1970.
5. Y. Dinitz, "Dinitz' Algorithm: The Original Version and Even's Version", *Lecture Notes in Computer Science*, vol. 3895, pp. 218-240, 2006.
6. J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the Association for Computing Machinery*, vol. 19, pp. 248-264, 1972.
7. L. R. Ford and D. R. Fulkerson, "*Flows in networks*", Princeton, N.J.: Princeton University Press, 1962.
8. S. Fujishige, "A maximum flow algorithm using MA ordering", *Operations Research Letters*, vol. 31, pp. 176-178, 2003.
9. A. V. Goldberg, "Network Optimization Library", Available: http://www.avglab.com/andrew/soft.html.
10. A. V. Goldberg, "Synthetic Maximum Flow Families", Available: http://www.avglab.com/andrew/CATS/maxflow_synthetic.html.
11. A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem", *Proceedings of the 18th Association for Computing Machinery Symposium on the Theory of Computing*, pp. 136-146, 1986. Full paper in *Journal of the Association for Computing Machinery*, vol. 35, pp. 921-940, 1988.
12. D. S. Johnson and C. C. McGeoch, "*Network Flows and Matching: First Dimacs Implementation Challenge*", American Mathematical Society, Providence, RI, 1993.

13. A. V. Karzanov, "Determining the maximal flow in a network by the method of preflows", *Soviet Mathematics Doklady*, vol. 15, pp. 434-437, 1974.
14. Y. Matsuoka and S. Fujishige, "Practical efficiency of maximum flow algorithms using MA ordering and preflows", *Journal of the Operations Research Society of Japan*, vol. 48, pp. 297-307, 2005.
15. The first DIMACS international algorithm implementation challenge: Network Flows and Matching, 1990, Available: ftp://dimacs.rutgers.edu/pub/netflow/.

在階層網路上依節線容量等比例推送流量以求解最大流量問題之研究

王逸琳[1*] 李俊賢[1]
[1]成功大學工業與資訊管理學系（台南市東區大學路一號）
*ilinwang@mail.ncku.edu.tw

摘要

最大流量問題為一基本之網路最佳化問題，旨在於一具有流量上限的網路中尋找從起點到訖點間可流通之最大可能流量，此問題已有超過五十年以上的研究歷史。傳統的最大流量演算法大致可分為兩大類：其一為不斷自起點找出一條可擴張流量至訖點之擴張路徑來擴張流量；而另一則為先塞滿起點連結出去之弧，允許節點暫存流量，再將具有暫存流量的節點沿其鄰近之一條可行弧逐次擴張流量於其鄰近點，直至所有之暫存流量擴張至訖點或返回起點而止。這些解法在每次擴張流量時，皆僅能沿網路圖之單一路徑或單一弧進行，本論文以能一次即利用網路中所有的可行弧來擴張流量為目標，使用最短路徑所建構之階層網路來擴張流量，提出一個新的具多項式時間複雜度之最大流量演算法。該演算法根據階層網路中各弧殘餘流量上限的比例來分配每次之擴張流量，稱為proportional arc augmenting (PAA) 演算法。PAA演算法依據可行子網路中各節點所連結出去之可行弧的殘餘流量上限比例，計算出該子網路所有弧之擴張流量向量δ，再根據此擴張流量向量計算出該次從起點到訖點之擴張流量，此種等比例擴張流量方式符合具較大流量上限之弧能擴張更多流量的直覺，並保證每次至少可塞滿一個節點。在一個具有n個節點及m條弧的網路上，PAA演算法可在$O(n^2m)$多項式時間內算出其最大流量。此外，本論文亦提出一些加速PAA演算法實作效率之方法。

關鍵字：*最大流量問題、階層網路、等比例推送、多項式時間演算法*