Solving Maximum Flows on Distribution Networks: Network Compaction and Algorithm

I-Lin Wang
Institute of Information Management
National Cheng Kung University, Taiwan
ilinwang@mail.ncku.edu.tw

Ju-Chun Lin
Institute of Information Management
National Cheng Kung University, Taiwan
r7692110@mail.ncku.edu.tw

Abstract

The maximum flow (max-flow) problem is a fundamental network optimization problem which computes for the largest possible amount of flow sent through the network from a source node to a sink node. This problem appears in many applications and has been investigated extensively over the recent four decades. Traditional max-flow problem may require some modification in its constraints to deal with real-world applications. Fang and Qi proposed a new max-flow model, named as manufacturing network flow model, to describe a network with special distillation nodes or combination nodes. Based on this new model, Ting proposes an approach to solve the max-flow problem in a distribution network which contains both ordinary and distillation nodes. Her approach identifies an augmenting subgraph connecting both source and sink nodes which can be further decomposed into several components where flows in each component can be expressed by a single variable and solved by a system of homogeneous linear equations. Their method requires manual detection for components and thus is not trivial to implement. In this paper, we first propose a polynomial-time network compaction algorithm which simplifies the distribution network, and then present detailed procedures for solving the max-flows on a distribution network.

1. Introduction

In the competitive business environment nowadays, cooperative and competitive relationship among customers, retailers, distributors, manufacturers, and vendors in a supply chain becomes much more complicated to achieve a better supply chain management. For example, suppose one unit of product A (10g/unit) is made of two units of material B (1g/unit) and one unit of material C (8g/unit). Material B and C may be purchased from different vendors by different channels. There may be several manufacturers who make product A with different production rates and obtain materials B and C via different channels. Using Figure 1 as an example, there are two manufactures (M_1 and M_2) and three vendors (V_1, V_2 and V_3) denoted by white circles. The process of decomposing products or semi-products into materials can be represented by gray circles and half-circles. The materials or semi-products are then purchasable from several vendors. For each outgoing arc of a gray half-circle, we associate a bracket with a number indicating the weight proportions of materials required for composing one unit of product. An arc connecting circles or half-circles may represent a process of production, decomposing, shipping, or outsourcing. We also associate a parenthesis for each arc indicating the object name and capacity for the process. If a process has unlimited capacity, we use a "∞" to represent its capacity. For example, the capacity of arc (M_1, M_2) means M_2 can provide at most 300g of product A to M_1. To produce x g of product A, M_2 has to purchase $0.2x$ g of material B and $0.8x$ g of material C. Also, M_2 can purchase at most 40g and 10g of B from V_1 and V_2, respectively. This transformed network thus only differs from the conventional one by the appearance of gray half-circles and the flow distribution constraints associated with their leaving arcs. Such a network model is called a *distribution network*, a special case of a manufacturing network flow model investigated by Fang and Qi [4]. In their model, a gray half-circle is called a *distillation* node or a *combination* node depending on its orientation.

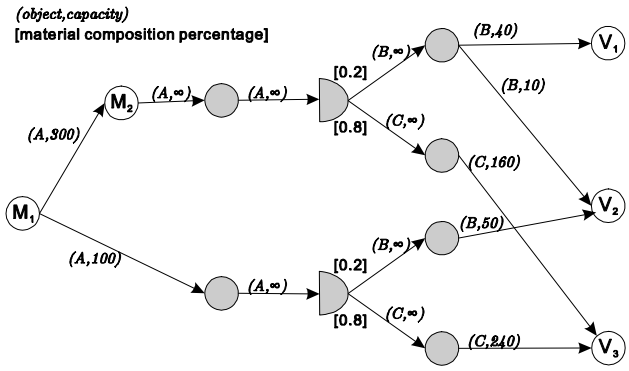


Figure 1. A supply chain example

In general, manufacturers either produce final products made of materials purchased from several vendors, or put part of their production out to contract with other manufacturers. Thus a final product may be made through several channels in different stages of a supply chain. Identifying the bottleneck for a supply chain becomes an important issue to improve the entire efficiency. In other words, calculating the maximum throughput (i.e. product flow) for either the entire chain or part of the chain helps us to evaluate the chain's efficiency. This problem in fact is a maximum flow problem on a distribution network in which the flow passing through some specific nodes has to be distributed or combined by predefined proportions. This maximum flow example as

shown in Figure 1 just illustrates possible relationships in a 2-stage supply chain. In practice, similar relationships and interactions among different stages in the supply chain do exist and make the problem much more complicated and difficult to solve.

To solve the maximum flow problem for a distribution network, Ting [12] proposes an approach which identifies an augmenting subgraph connecting both source and sink nodes. The augmenting subgraph can be further decomposed into several components where flows in each component can be expressed by a single variable and solved by a system of linear equations. Their method requires manual detection for components and thus is difficult to implement by modern computers. In this paper, we observe special properties for a distribution network and propose a method to compact the network which simplifies the problem. Then we suggest an implementation for Ting's approach and analyze its complexity.

2. Preliminaries

2.1 Notation and formulation

The traditional maximum flow problem was first formulated by Fulkerson and Dantzig. Let *G = (N,A)* be a directed graph with node set *N* and arc set *A*. We consider a capacitated network *G* where each arc (*i,j*) in *A* is associated with a nonnegative capacity u_{ij} . Let $n = |N|$, $m = |A|$, and $U = \max_{(i,j)\in A}\{u_{ij}\}$. There are three types of nodes: S-node, T-node and O-node (see Figure 2). An S-node is a source node connected only by outgoing arcs. A T-node denotes a sink node connected only by incoming arcs. An O-node represents a transshipment node connected with both incoming and outgoing arcs. For each node *i*, we associate an integer *b*(*i*) representing its supply/demand. In particular, *b*(*i*) > 0 if *i* is a supply node; *b*(*i*) < 0 if *i* is a demand node; and *b*(*i*) = 0 for each transshipment node *i*. Usually an S-node is a supply node, a T-node is a demand node, and any transshipment is an O-node.

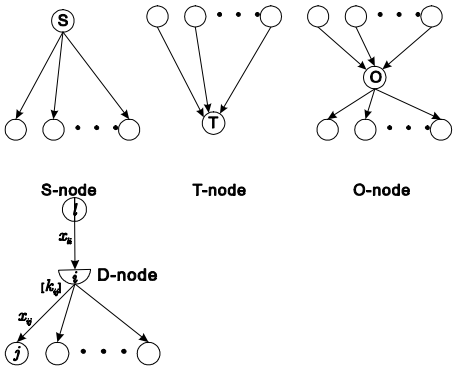


Figure 2. S-node, T-node, O-node and D-node

A network flow model has to obey the constraints for flow balance and bounds. In particular, for each node *i*, the amount of flow leaves *i* minus the amount of flow enters *i* should equal to *b*(*i*). Also, the flow passing through an arc (*i,j*) can not exceed its capacity u_{ij} , but must be more than its lower bound l_{ij} . In this thesis we assume $l_{ij} = 0$ for each arc (*i,j*). The traditional maximum flow problem seeks the maximum amount of flow, denoted by *V*, that is shipped from the S-node to the T-nodes while the flow assignment satisfies the arc capacities and flow balance constraints for all arcs and nodes. Let *S*, *T*, and *O* denote the set of S-nodes, T-nodes, and O-nodes, respectively.

Most maximum flow algorithms are applied on an induced graph *G*(*x*) named as the *residual network* for a given flow vector *x*. In particular, for each arc (*i,j*) with flow x_{ij} , we replace (*i,j*) by two arcs, (*i,j*)and (*j,i*) with *residual capacities* $r_{ij} = u_{ij} - x_{ij}$ and $r_{ji} = x_{ij}$, respectively. An arc $(i, j) \in A$ is *saturated* when $r_{ij} = 0$. The residual network only contains arcs with positive residual capacities. An *augmenting path* from *s* to *t* is a path connecting *s* and *t* in the residual network. Therefore, if there exists an augmenting path in *G*(*x*) from an S-node to a T-node, it means more flow can be shipped along this augmenting path and thus the current flow vector *x* is not optimal.

2.2 Conventional max-flow problem and solution methods

The traditional maximum flow problem can be formulated as follows:

$$\max \sum_{i\in S} b_i$$

$$s.t. \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i \begin{cases} \geq 0, & \forall\ i \in S \\ = 0, & \forall\ i \in O \\ \leq 0, & \forall\ i \in T \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A$$

Assuming b_i and u_{ij} to be integral for each node *i* and arc (*i,j*), the optimal solution for this linear programming problem has to be integral due to the total unimodular property of the constraint matrix [9].

Table 1. Summary of conventional max-flow algorithms

Algorithm	Complexity
Fulkerson and Dantzig [7]	$O(n^2 mU)$
Ford and Fulkerson [5]	$O(mU)$
Edmonds and Karp [3]	$O(n^2 m)$
Dinic [2]	$O(n^2 m)$
Sleator and Tarjan [11]	$O(m \log n)$
Karzanov [10]	$O(n^3)$
Goldberg and Tarjan [8]	$O(m \log \frac{n^2}{m})$
Fujishige [6]	$O(nm \log n \log nU)$

Fulkerson and Dantzig [7] formulate the conventional maximum flow problem as a special minimum cost flow problem and then solve it by simplex algorithm. Ford and

Fulkerson [5] propose the first augmenting path algorithm which searches for a path connecting the source and sink and then augment flow as much as possible. Edmonds and Karp [3] suggest a polynomial-time augmenting path algorithm. Similar algorithm has also been proposed by Dinic [2] and further improved by Sleator and Tarjan [11]. Karzanov [10] introduce the first preflow-push algorithm in a layered network. Goldberg and Tarjan [8] give a better preflow-push algorithm. Recently, Fujishige [6] propose a MA ordering algorithm, which can be regarded as an acceleration to the Edmonds-Karp algorithm. Table 1 summarizes the complexity of these max-flow algorithms. More max-flow algorithms can be found in [1].

2.3 Distribution max-flow problem and solution methods

The distribution network model is first introduced by Fang and Qi [4]. Besides the three types of nodes (i.e. S-nodes, T-nodes, and O-nodes), the distribution network model further introduces a type of distillation node, denoted by D-node, as the half-circle previously appeared in Figure 1 and Figure 2.

A D-node *i* has only one incoming arc (*l*,*i*) and at least two outgoing arcs called *distribution arcs*. Each distribution arc (*i*,*j*) is associated with a positive rational number, denoted by k_{ij}, to specify the percentage of the flow in (*l*,*i*) to be distributed into the distribution arc (*i*,*j*). We also assume $\sum k_{ij} = 1 \quad \forall i \in D$ and $k_{ij} > 0$ for each distribution arc (*i*,*j*). Let x_{ij} represents the flow in arc (*i*,*j*), then the flow along each distribution arc (*i*,*j*) for a D-node *i* with one incoming arc (*l*,*i*) can be calculated by $x_{ij} = k_{ij} x_{li}$. We call the relationship for flows on arcs connecting a D-node as "flow distillation", and k_{ij} as a *distillation factor*. For each distribution arc (*i*,*j*), we define a *normalized capacity* $\bar{u}_{ij} = u_{ij} / k_{ij}$ to represent the least amount of flow required in arc (*l*,*i*) to saturate arc (*i*,*j*).

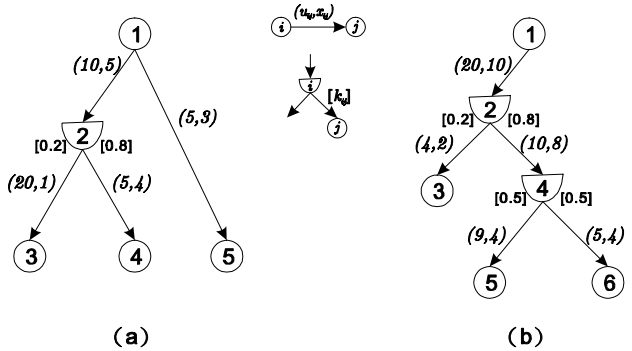


Figure 3. Two examples of distribution networks

A D-*family* is a group of nodes other than D-nodes adjacent to the same D-node. We call the node in a D-family that sends flow to the D-node as its *mother node*, and the nodes that receive flow from the D-node in a D-family as its *member nodes*. Within a D-family, the M-arc connects the mother node to the D-node, while an S-arc connects the D-node to a member node. By definition of a D-node, there is only one D-node, one mother node, and at least two member nodes in a D-family. For example, nodes 1, 2, 3 and 4 in Figure 3(a) form a D-family in which node 1 is the mother node, nodes 3 and 4 are member nodes, arc (1,2) is the M-arc, and arcs (2,3) and (2,4) are S-arcs. For cases where a D-node is adjacent to anther D-node, by definition they can not form a D-family. We call the set containing the maximum number of adjacent D-nodes as a D-*group*. For example nodes 2 and 4 in Figure 3(b) form a D-group.

The distribution maximum flow problem solves for the maximum amount of flow that the T-nodes can receive from the S-node as long as the flow obeys the constraints of conservation, bounds, and distillation in a distribution network. We denote the set of D-nodes by *D*. A distributed maximum flow problem can be formulated as follows:

$$\max \sum_{i \in S} b_i$$

$$s.t. \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = b_i \begin{cases} \geq 0, & \forall\, i \in S \\ = 0, & \forall\, i \in O \text{ or } D \\ \leq 0, & \forall\, i \in T \end{cases}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A$$
$$x_{ij} = k_{ij} x_{li} \quad \forall i \in D$$

Unlike the traditional maximum flow problem, assuming b_i and u_{ij} to be integral for each node *i* and arc (*i*,*j*) and k_{ij} to be rational for each distribution arc (*i*,*j*), the optimal solution for this linear programming problem is not guaranteed to be integral, since the property of total unimoduality has been affected by the flow distillation constraints. We may view this problem as a traditional maximum flow problem with side constraints (i.e. flow distillation). This problem is still a linear programming problem and thus can be solved by any LP algorithms and software.

Although Fang and Qi [4] firstly introduce the maximum flow problem in distributed networks, they do not provide its solution method. This maximum flow problem is more complicated and difficult to solve than the traditional one due to the flow distillation constraints. For example, suppose there exists an augmenting path connecting an S-node and T-node, but this path passes through a D-node. In such a case, we have to consider the D-node as another source node, and for each of its outgoing arcs we have to check whether there exists an augmenting path passing through that outgoing arc to a T-node. If we fail to find such an augmenting path from the D-node, we can not use the original augmenting path connecting a S-node and T-node, since sending flow via that path will force the D-node to send flows along all of its outgoing arcs and some of them will be "blocked out" somewhere in the network.

Ting [12] proposes an approach to solve this problem by adopting the concept of the Depth-First Search (DFS) which tries out every augmenting subgraph that goes to the sink or source and satisfies flow balance, bound and distillation constraints. After finding such an augmenting

subgraph, the algorithm then identifies components in the augmenting subgraph. Flow inside each component can be represented by a single variable. The flow balance constraints for all nodes that joint different components form a system of linear equations. Components joint with each other by nodes. Thus, the flow on an augmenting graph can be calculated by solving such a system of linear equations. The maximum flow can be calculated by iteratively identifying an augmenting subgraph, decomposing components, solving for variables associated with each component, and then calculating flows inside each component.

The correctness of Ting's approach relies on the correctness of the modified DFS algorithm she uses. Her algorithm is quite generic and also requires manual detection for some procedures. In Section 3 we will propose a network compaction procedure that reduces the problem size and then give an efficient implementation of her algorithm in Section 4.

3. Compacting distribution network

Any given distribution network may contain topology that is reducible. A preprocessing procedure can be applied to compact the original network to an equivalent one of smaller size. Here we explain why solving the maximum flow on these two networks gives the same answer and how the transformation can be conducted by a preprocessing procedure.

We observe that a distribution network may be further simplified by either merging several nodes or arcs, or unifying the effect of the capacities for some arcs. In particular, we give four rules to detect whether a distribution network is compactable, and then explain the compacting procedures.

(C1) Compacting D-groups

Nodes in the same D-group can be compacted into a D-node. A D-group contains the maximum set of adjacent D-nodes. Since flows over all arcs adjacent to a D-node are proportional to each other by some constants, we can easily calculate flow over an adjacent D-node which then can be used to calculate flows on all of its arcs. Therefore the arcs adjacent to a D-group have flows proportional to each other. The process of calculating flows and resetting capacities on all arcs adjacent to a D-group have the same result as merging all adjacent D-nodes. Thus an entire D-group can be replaced by a single representative D-node. Suppose there are q_r O-nodes ($i_{q_1}, i_{q_2}, \mathrm{K}, i_{q_r}$) adjacent to a D-group with a representative D-node i_o. Suppose an O-node i_{q_w} is connected to i_o by a path $p_{i_o i_{q_w}}$ (i.e. $i_o \to \mathrm{K} \to i_{q_w}$) composed of $\left|p_{i_o i_{q_w}}\right|$ S-arcs. The compacting process is as follows:

Among all the adjacent D-nodes, we retain the one (i_o) closest to the S-node and use i_o to represent the final merged D-node. For example, node 2 in Figure 4(a) is retained to be the representative D-node for that D-group.

a) Delete all original S-arcs associated with this D-group and add new S-arcs from $\{(i_o, i_{q_1}), (i_o, i_{q_2}), \mathrm{K}, (i_o, i_{q_r})\}$. In Figure 4(b), we create new arc (2,5) and arc (2,6).
b) The distillation factors on new S-arcs can be calculated by multiplying the distillation factors of S-arcs passing through intermediate D-nodes. In particular, $k_{i_o q_w} = \prod_{(i,j) \in p_{i_o q_w}} k_{ij}$. For example, the distillation factor for the new arc (2,5) in Figure 4(b) is $0.8 * 0.5 = 0.4$.
c) The capacities for the new S-arcs can be calculated by $\min_{y \text{ lies on } p_{i_o q_w},\ (y,y') \text{ and } (w',q_w) \text{ lies on } p_{i_o q_w}} \{u_{w'q_w}, \prod_{(i,j) \text{ lies on } p_{yq_w}} k_{ij} u_{yy'}\}$ For example, the capacity of arc (2,5) in Figure 4(b) equals to $\min\{u_{45}, k_{24} k_{45} u_{24}\}$.

It takes $O(m)$ time to apply this compacting rule by a search algorithm. Each time when we compact a D-group, at least one D-node and one arc connecting D-nodes will be reduced.

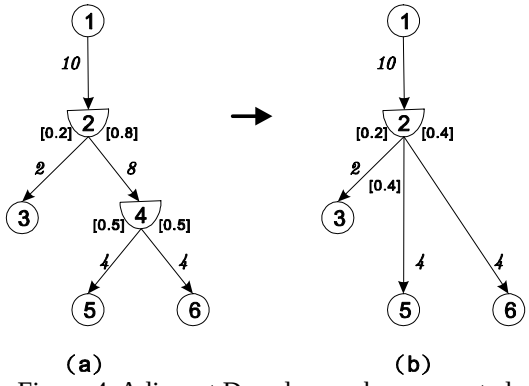
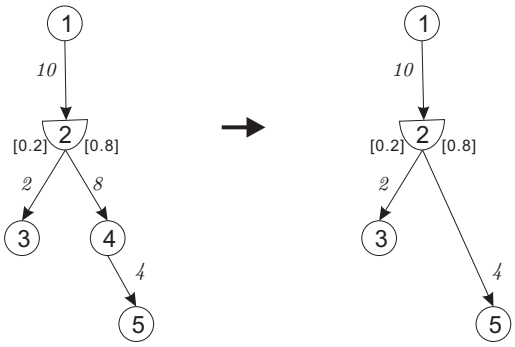


Figure 4. Adjacent D-nodes can be compacted

(C2) Compacting single transshipment O-nodes

Any O-node with only one incoming arc and one outgoing arc can be compacted. When an O-node with single incoming and output arcs, this O-node is simply used for transshipment and has no affection to the flows in those arcs. Thus we may remove this O-node and merge these two arcs into one single arc with new capacity equal to the minimum capacity of the original two arcs. Moreover, if the O-node is adjacent to a D -node, the capacities for all arcs adjacent with that D-node have to be adjusted accordingly by the first rule stated above. Take the distribution network in Figure 5(a) for example, after removing node 4 and merging arc (2,4) and (4,5) by an arc (2,5) with capacity equal to $\min\{8, 4\} = 4$ as shown in Figure 5(b).

This compacting process takes $O(n)$ time since we only need to scan each node once. Each time when we compact an O-node, one O-node and one arc connecting this O-node will be reduced.

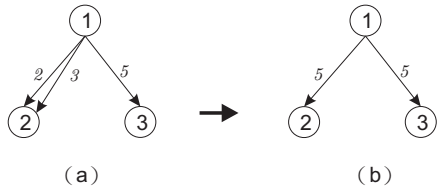


(a) (b)

Figure 5. Any O-node with only one incoming arc and one outgoing arc can be compacted.

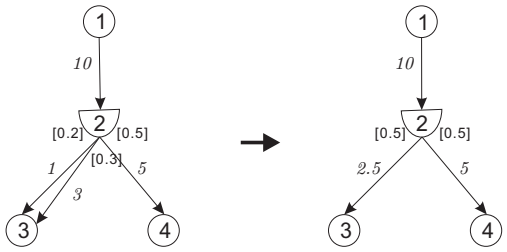
(C3) Compacting parallel arcs

Parallel arcs connecting to the same end nodes can be compacted. Although we assume no parallel arcs in the network, they may be formed in the intermediate compacting process. Since we may view those arcs with the same tail and head nodes as one huge arc, we merge these parallel arcs into one new arc and sum up their capacities to be its capacity. There are two cases for parallel arcs. One case is that the parallel arcs connect to the same end O-nodes and we sum up their capacities to be the capacity of new arc. See Figure 6 for example. The other case is that the parallel arcs connect to the same D-node and O-node. In this case, we merge parallel S-arcs into a single S-arc and calculate its distillation factor by summing up the distillation factors of these parallel S-arcs. Then, the new capacity can be calculated by finding out the minimum normalized capacity in these parallel S-arcs and multiplying it with the new distillation factor. See Figure 7 for examples. The distillation factor of new arc (2,3) is $0.2+0.3=0.5$. The capacity of the new arc (2,3) will be $\min\{\frac{1}{0.2},\frac{3}{0.3}\}*0.5=2.5$. This compacting process takes $O(m)$ time by scanning all outgoing arcs for each node. Each time when we compact parallel arcs, at least one arc will be reduced.



(a) (b)

Figure 6. Parallel arcs connecting to the same end nodes can be compacted

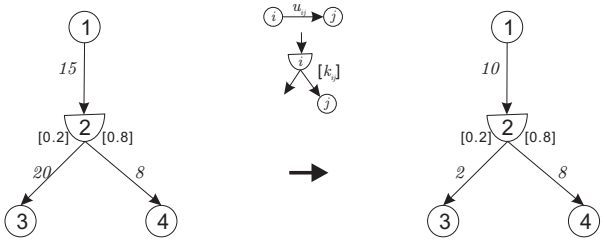


(a) (b)

Figure 7. Parallel arcs connecting to the same D node and O-node can be compacted

(C4) Compacting mismatched capacities

Any D-node connected with arcs of mismatched capacities can be compacted. According to the flow distillation constraints, the flow on each outgoing arc from a D-node is proportional to the flow on the incoming arc. Therefore we may easily identify the first saturated arc for a D-node by calculating the normalized capacity for each outgoing arc relative to the incoming arc. Take Figure 8(a) as an example, x_{12} has to be at least $\frac{20}{0.2}=100$ to saturate arc (2,3), and $\frac{8}{0.8}=10$ to saturate arc (2,4). Thus the bottleneck happens on arc (2,4) since it is the first arc to be saturated when we increase x_{12}. By this observation we can identify the minimum normalized capacity $\bar{u}_{\min}$ among all arcs adjacent to a D-node *i*, reset the capacity of its incoming arc (*l*,*i*) to be $\bar{u}_{\min}$, and then reset the capacity for each distribution arc (*i*,*j*) by $k_{ij}\bar{u}_{\min}$. For the example in Figure 8(a), $\bar{u}_{\min}=\min\{15,\frac{20}{0.2},\frac{8}{0.8}\}=10$. Thus arc (2,4) is the bottleneck and we can reset $u_{12}=\bar{u}_{\min}=10$ and $u_{23}=0.2\bar{u}_{\min}=2$ as shown in Figure 7(b). This compacting process simplify the problem since now once an arc adjacent to a D-node is saturated, all the other arcs will also be saturated. Thus a maximum flow algorithm may detect a bottleneck arc much earlier on this compacted network than the original one. This compacting process takes $O(m)$ time, but only needs to be conducted once in the end.



(a) (b)

Figure 7. Any D-node with mismatch capacities of arcs can be compacted

These four compacting rules may induce each other. For example, Figure 6(a) shows when we compact a single transshipment O-node, we may induce parallel arcs. Figure 9 illustrates inducing relations between these four compacting rules. Note that there are two directed cycles:

(C1)-(C3)-(C2)-(C1) and (C2)-(C3)-(C2). Thus we may iteratively conduct (C1), (C3) and (C2) or (C2) and (C3) until they finish their inducing relations and then conduct (C4) in the end of the process as illustrated in the following algorithm COMPACT.

Algorithm COMPACT(G)
While (G is not in compact form)
Case 1: detect a D-group, then apply (C1)
Case 2: detect a single transshipment O-node, then apply (C2)
Case 3: detect parallel arcs, then apply (C3)
End while
Case 4: detect capacity-mismatched arcs, then apply (C4)

Each time when any of compacting rules (C1), (C2) and (C3) is applied, the network will be reduced by at least one arc or one node. Thus the entire compacting process takes $O(m^2)$ time and produces a simplified network which has smaller size and is easier to detect the bottleneck for solve a maximum flow problem.

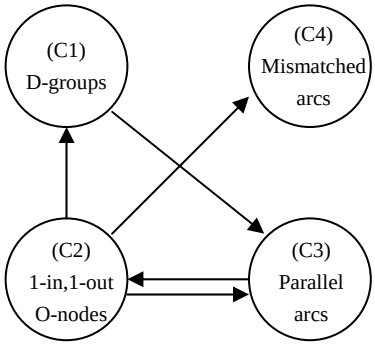


Figure 9. Inducing relations of different compacting rules

In summary, we observe two properties for a compacted distribution network. First, every intermediate node (i.e. an O-node or a D-node) is adjacent to at least three arcs, including one incoming arc and one outgoing arc. Second, D-node will not be adjacent to any other D-node. By these observations we may easily check whether a distribution network is compactable. For a distribution network not in its compact form, we may compact using the algorithm COMPACT.

4. Modified augmenting path algorithm

Ting [12] proposes an approach to solve the maximum flow problem in distributed network. However, her approach only contains a couple of rules, and is not described as an algorithm. Additionally, her approach requires manual detection for components, which is not trivial for implementation. Here we organize their approach and propose an implementation for their approach.

Modified augmenting path algorithm

```
flow:=0
flag:=true
While (flag==true) do
Begin
    While (flow<=0 ) do          // if the AG is invalid, find another one
    Begin
        Find another AG
        If (AG exists) then      // if there is another AG, find the flow
            Identify components in the AG
            Solve the linear equations
            Determine the flow could be sent
        Else then                // if there is no AG, exit the loop
            break
            flag := false
    End
    If (flag==true) then         // if there is a valid AG, send the flow
        Send flow
        Update the residual network
        flow:=0
    Else then break              // if there is no AG, exit the loop
End
```

Figure 10. Pseudo code for implementing Ting's approach

First we introduce additional notation required for our algorithm. Let *AG* represent an augmenting subgraph, *flag* indicate whether any *AG* exists (*flag*=1) in the current residual network, and *flow* be the total amount of flow that could be sent in an *AG*. Figure 10 illustrates steps for our implementation which includes five major procedures: (1) to construct an augmenting subgraph, (2) to identify components in an augmenting subgraph, (3) to solve a system of linear equations, (4) to determine the flow to be sent in an augmenting subgraph, and (5) to update the residual network

4.1 Constructing an augmenting subgraph

Ting [12] adopts the concept of the Depth-First Search (DFS) to try out every augmenting subgraph, which goes to the sink or source and satisfies flow distillation constraints. As we know, an augmenting path is a directed path from the source to the sink. Nevertheless, when the augmenting path in a distribution network reaches the sink, the algorithm has to track back any D-node in the augmenting subgraph and go along the outgoing arc of the D-nodes to satisfy flow distillation constraints, which may turn the augmenting path into the augmenting subgraph. In the intermediate stage of tracking back any D-node, the subgraph grows by adding arcs scanned by DFS and pauses when the sink, source or a node that is on current augmenting subgraph is re-visited, then it checks whether all the distribution arcs out of a visited D-node have been passed or not. If there exists a distribution arc not scanned out of a visited D-node, the algorithm keeps searching via that distribution arc. The procedure continues until no such distribution arc exists.

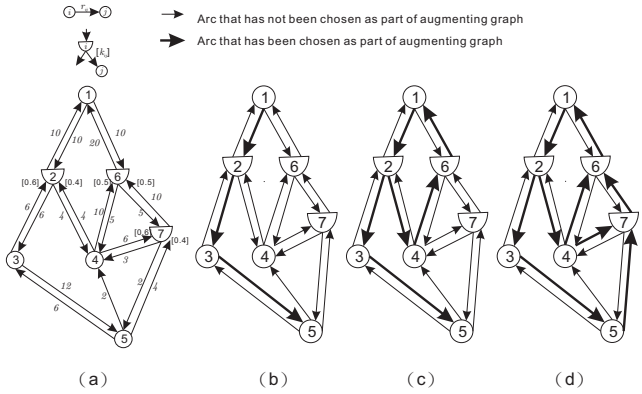


Figure 11. An example of constructing an augmenting subgraph

Take Figure 11 for example. Given a residual network as shown in Figure 11(a), the bold arc in Figure 11(b) denotes the augmenting subgraph that reaches a T-node but has not tracked back other distribution arcs from D-nodes. After tracking other distribution arcs from a D-node, we get the augmenting subgraph using bold arcs in Figure 11(c). We repeat the tracking back operations for each D-node in the augmenting subgraph. Finally, the complete augmenting subgraph is determined as shown in Figure 11(d)

Although the idea of using DFS looks straightforward, this procedure is in fact not so trivial. In fact, the searching order for an unscanned distribution arc does affect the validity of the algorithm. In particular, if there is more than one unscanned distribution arc out of a visited D-node, which unscanned distribution arc to be scanned first will affect the correctness of the algorithm. (See [12] for details) To guarantee the correctness of the algorithm, when we backtrack from a D-node connected by α unscanned arcs, we must finish probing all the $\alpha!$ searching orders. Thus although a modified DFS only takes $O(m)$ time, the exponentially grown searching orders make this algorithm take exponential time.

4.2 Identifying components in an augmenting subgraph

After an augmenting subgraph is constructed, the algorithm then identifies components in the augmenting subgraph. A component is a maximum subgraph in which arc flows can be expressed by linear functions of a single variable. In other words, once the amount of flow on one of the arcs inside a component is determined, all the other arc flows in the same component can be determined right away. Components are separated with each other by O -nodes which have at least three adjacent arcs in an augmenting subgraph. If an O-node has only two adjacent arcs, which must be one incoming arc and one outgoing arc (otherwise an O-node becomes an S-node or a T-node, which violates its definition), the flows on these two arcs must be the same and could be represented by a single variable which means these two arcs belong to the same component. Likewise, arcs connecting to the same D-node will be in the same components since the arc flows are proportional to each other. Therefore, arcs with flows represented by functions of the same variable belong to the same component

Continuing on the above example in Figure 11, suppose the flow in arc (1,2) is a, then the flows in arc (2,3) and arc (2,4) become $0.6a$ and $0.4a$ respectively. As for node 3 , because the flow in arc (2,3) equals to the flow in arc (3,5), we will know the flow in arc (3,5) is $0.6a$. As for node 4, we can not determine the flows on arc (4,6) and arc (4,7), so we may first assign two more variables to represent them but later we will find the flows on (4,6) and (4,7) are proportional to each other since they are in the same component. The result is shown in Figure 12(a). The flow balance constraints for all nodes that joint different components form a system of linear equations. The equation associated with node 4 is $0.4a = 0.5b + 0.3b$.

In implementation, we first identify the O-nodes which are incident to more than two arcs in an augmenting subgraph and define these nodes as *boundary nodes*. Starting from the source, a Breadth-First-Search (BFS) is applied to identify all of its reachable nodes. We may modify the BFS to backtrack whenever a boundary node is encountered, so that we can determine the flow value for each arc in the same component as a function of a single variable. We repeat the same procedures by starting from a different boundary node as long as it has an adjacent arc that has not been visited by the modified BFS. This procedure takes $O(n+m)$ time.

4.3 Solving a system of linear equations

By the flow balance constraints associated with each O-node jointing different components, we can obtain a system of homogeneous linear equations. Solving this system of linear equation can help us to unify the variables of different components. In particular, the objective is to replace all variables with a single variable instead of solving for the arc flow value directly. However, it is possible that this system of linear equation may have rank higher than the total number of components, which means the system of linear equations may only have the trivial solution (i.e. zero is the only solution). In such a case, the augmenting subgraph is ineffective since there is zero flow to be augmented. Otherwise, we can replace all variables with a single variable.

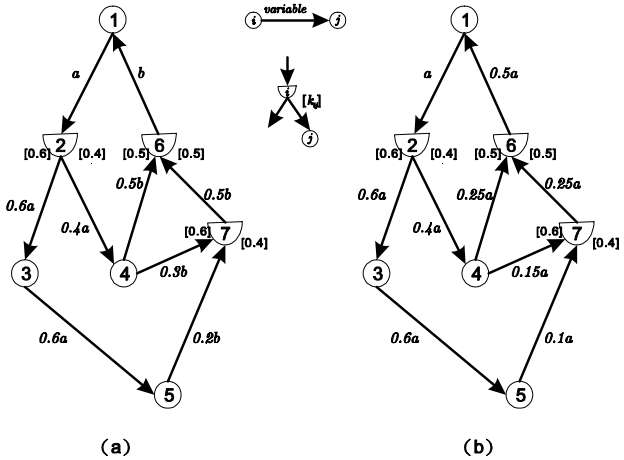


Figure 12. Identify components and replace all variables with a single variable

Using the example in Figure 12, there are two components jointed at node 4. The flow balance constraint at node 4 shows $0.4a = 0.5b + 0.3b$ which can be reduced to $b = 0.5a$, and then we can replace all variable b with a as shown in Figure 12(b). The total flow we could send in the augmenting subgraph is the total incoming flow of the source minus the outgoing flow of it. In the example of Figure 12(b), it will be $a - 0.5a = 0.5a$.

In implementation, we may detect the existence of nontrivial solution by counting the number of components and jointed O-nodes. To solve the system of homogeneous linear equations, we may use directive method such as Gaussian elimination to express each variable as a linear function of one universal variable. The time of this process takes $O(q^3)$ where q represents the total number of components.

4.4 Calculating the flow to be sent on an augmenting subgraph

After unifying the arc flows by linear functions of a single variable, we normalize the capacity again for each arc. In particular, suppose an augmenting subgraph $G' = (N', A')$ and we use the flow on arc (i', j'), x, to be the single variable to express flows over all other arcs. Suppose $x_{uv} = k'_{uv} x$ where k'_{uv} is a rational constant. Then the normalized capacity of arc (u, v) relative to arc (i', j') becomes u_{uv} / k'_{uv} which means the amount of flow (i', j') should contain to saturate (u, v). Exploiting the property of normalized capacity, we can identify the bottleneck arcs easily by calculating $\min\{u_{ij} / k'_{ij}\}$ for each $(i, j) \in A'$. Refer to Figure 11, the flow a can be calculated by $\min\{\frac{10}{1}, \frac{6}{0.6}, \frac{12}{0.6}, \frac{4}{0.4}, \frac{20}{0.5}, \frac{10}{0.25}, \frac{10}{0.25}, \frac{6}{0.15}, \frac{4}{0.1}\} = 10$, the maximum flow over the entire augmenting subgraph is $0.5a = 5$ and arc (1,2) is the first arc to be saturated when we increase the flow over the augmenting subgraph.

In implementation, this procedure can be done in $O(m)$ time.

4.5 Updating the residual network

After calculating the maximum flow along an augmenting subgraph, we update the residual network and repeat the above steps until there is no more augmenting subgraph. The maximum flow can be calculated by iteratively constructing an augmenting subgraph, decomposing components, solving for variables associated with each component, calculating flows inside each component, and then updating the residual network.

5. Conclusion and future research

This paper investigates the max-flow problem on a specialized network called distribution network in which a specialized D-node has to distribute its incoming flows by a vector of predefined distillation factors to its outgoing arcs. To simplify the problem, we propose a polynomial-time network compaction algorithm which reduces the size of the network and unifies the upper bounds so that an equivalent and simpler problem with smaller size can be solved. We propose a modified augmenting path algorithm with detailed procedures based on Ting's method [12], and analyze the complexity of our algorithm. The modified augmenting path algorithm may be not efficient since it involves total enumeration on searching orders for arcs connecting a D-node in the augmenting subgraph. This is very different from the conventional augmenting path algorithm. The inefficiency is caused by the distillation constraints associated with D-nodes and is inevitable.

On another thought, since the distillation constraints are locally associated with D-nodes, it may be worthy trying to modify the preflow-push algorithm [8] and handle the distillation constraints locally for each D-node. In particular, a preflow-push algorithm pushes flows via individual arcs instead of via augmenting paths, so it allows flow to be accumulated at some nodes (called active nodes) and violates flow balance constraints at intermediate stages. It pushes flow from an active node to its eligible neighbors which is one-arc closer to the sink than itself, and relabels an active node when there exist no eligible neighbors. If all the paths from active nodes to the sink nodes are saturated, these excess flows have to be shipped back to the source nodes. When one modifies the preflow-push algorithm to solve max-flow problem in a distribution network, one may encounter the difficulties such as how to label a D-node and how to send flows via a D-node when it connects with both active and inactive nodes. How to efficiently resolve these difficulties should be an interesting and challenging topic for future research.

Acknowledgements

This research is supported by NSC 93-2213-E-006-096

References

[1] Ahuja, R. K., Magnanti, T. and Orlin, J. *Network flows: theory, algorithms and applications.* Englewood Cliffs, New Jersey, U.S.A.: Prentice Hall, 1993.

[2] Dinic, E. A. "Algorithm for solution of a problem of

maximum flow in networks with power estimation," *Soviet Math. Dokl.*, 1970, 11, 1277-1280.

[3] Edmonds, J. and Karp, R. M. "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the Association for Computation Machine*, 1972, 19, 248-264.

[4] Fang, S. C. and Qi, L. "Manufacturing network flows: A generalized network flow model for manufacturing process modeling," *Optimization Methods and Software*, 2003, 18, 143-165.

[5] Ford, L. R. and Fulkerson, D. R. "Maximal flow through a network," *Canadian Journal of Mathematics*, 1956, 8, 399-404.

[6] Fujishige, S. "A maximum flow algorithm using ma orderings," *Operations Research Letters*, 2003, 31, 176 -178.

[7] Fulkerson, D. R. and Dantzig, G. B. "Computation of maximum flow in networks," *Naval Research Logistics Quarterly*, 1955, 2, 277-283.

[8] Goldberg, A. V. and Tarjan, R. E. "A new approach to the maximum flow problem," *Journal of the Association for Computation Machine*, 1988, 35, 921-940.

[9] Hoffman, A. and Kruskal, J. 1956. Integral boundary points of convex polyhedra. In H. Kuhn and A. Tucker (Eds.), *Linear inequalities and related systems* (pp. 233-246). Princeton, NJ: Princeton University Press.

[10] Karzanov, A. V. "Determining the maximal flow in a network by the method of preflows," *Soviet Math. Dokl.*, 1974, 15, 434-437.

[11] Sleator, D. D. and Tarjan, R. E. "A data structure for dynamic trees," *Journal of Computer and System Sciences*, 1983, 24, 362-391.

[12] Ting, M. J. 2005. *Maximum flow problem in the distribution network model.* Master's thesis, National Cheng Kung University.