JOURNAL OF INDUSTRIAL AND
MANAGEMENT OPTIMIZATION
Volume **2**, Number **3**, August **2006**

Website: http://AIMsciences.org

pp. **237–254**

MAXIMUM FLOW PROBLEM IN THE DISTRIBUTION NETWORK

R.L. Sheu

Department of Mathematics
National Cheng-Kung University, Tainan, Taiwan

M.J. Ting

Product Development Department
Shinkong Life Insurance Co., LTD., Taipei, Taiwan

I.L. Wang

Department of Industrial and Information Management
National Cheng-Kung University, Tainan, Taiwan

(Communicated by Shu-Cherng Fang)

Abstract. In this paper, we are concerned with the maximum flow problem in the distribution network, a new kind of network recently introduced by Fang and Qi. It differs from the traditional network by the presence of the D-node through which the commodities are to be distributed proportionally. Adding D-nodes complicates the network structure. Particularly, flows in the distribution network are frequently increased through multiple cycles. To this end, we develop a type of depth-first-search algorithm which counts and finds all unsaturated subgraphs. The unsaturated subgraphs, however, could be invalid either topologically or numerically. The validity are then judged by computing the flow increment with a method we call the *multi-labeling method*. Finally, we also provide a phase-one procedure for finding an initial flow.

1. **Introduction.** Consider the network $\mathcal{G} = (N, A, d, u)$ with $N = N_{PS} \cup N_O \cup N_S \cup N_T \cup N_D$ being the node set and A the arc set. The functions $d : A \to \mathcal{Q}_+$ and $u : A \to \mathcal{Q}_+$, which take non-negative rational values, represent respectively the minimum demand and the capacity on each arc. There are five types of nodes in the network $\mathcal{G}$: the S-node, the T-node, the O-node, the *Pseudo*-node, and the D-node. The S-nodes are source nodes. They can be thought to all come from a *Pseudo*-S-node. The T-nodes are termination (sink) nodes which can be combined into a single *Pseudo*-T-node. The O-nodes are the intermediate nodes other than the source and the sink nodes in the traditional network. The D-node is a very special feature. Each D-node has only one inward arc through which the goods must be proportioned to all its outward arcs at fixed ratios. It was first introduced by Fang and Qi in [5] to describe some manufacturing and/or distribution processes. Let the collection of each respective type be denoted by N_S, N_T, N_O, N_{PS}, and N_D.

2000 *Mathematics Subject Classification.* 90B10, 49M35.

Key words and phrases. Distribution network, maximum flow problem, labeling method, cycles.

This research work was partially supported by the National Center of Theoretic Sciences/ South of Taiwan.

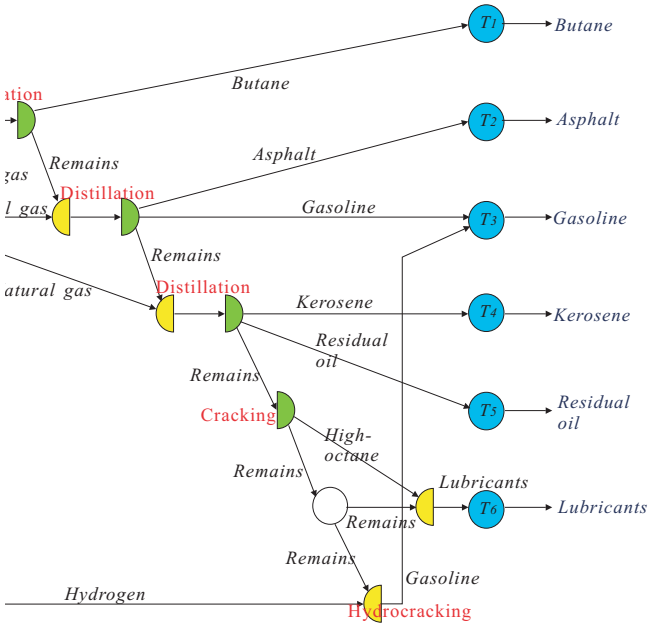

FIGURE 1. Refining crude oil

In a manufacturing case, D-nodes could be a distilling-like operation such as refining crude oil. See Figure 1 for an example. The D-shaped nodes here represent the distillation or cracking process, while the C-shaped nodes stand for the combination or synthesis processes. In this example, there are three S-nodes: S_1 is the oil field outputs; S_2 is the LPG or natural gas used to fuel the highly energy-intensive refining processes; S_3 provides hydrogen for hydrocracking. Among thousands of resulting products, we name only some representative ones which include butane, asphalt, gasoline, kerosene, lubricants, etc. Notice that petroleum products are usually distributed proportionally from the refinery. The proportionality depends on the geographical location, customer demand, and seasonal needs. Therefore, the whole manufacturing process can be modeled as a multi-commodity network problem with D/C-nodes and there is a recent result on the minimum cost flow problem by Mo et al. [10].

On the other hand, D-nodes could be used to describe the proportional allocation of a particular commodity. This might happen when a non-proportional distribution is not fair and a regulated proportional distribution is more desirable. Examples include water resources in a drought area, power supply in a highly energy-intensive industrial park, or the investment breakdown for a bunch of projects. Fang and Qi's original work in [5] deals with the minimum cost flow problem of this kind. For this reason, it is called the distribution network.

In this paper, we are concerned with the maximum flow problem of the *distribution network*. A feasible flow x in $\mathcal{G}$ is a rational-valued function $x : A \to \mathcal{Q}_+$ satisfying

$$\sum_{l \in E(i)} x_{li} = \sum_{j \in L(i)} x_{ij}, \qquad i \in N_O. \tag{1.1}$$

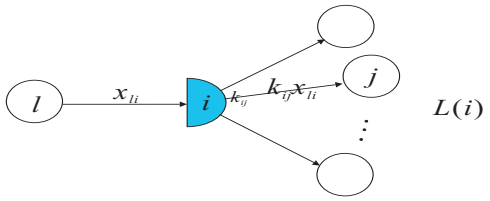

FIGURE 2. An example of a D-node

$$x_s = \sum_{j \in L(s)} x_{sj}, \qquad s \in N_S. \tag{1.2}$$

$$\sum_{l \in E(t)} x_{lt} = x_t, \qquad t \in N_T. \tag{1.3}$$

$$x_{ij} = k_{ij} x_{li}, \ \sum_{j \in L(i)} k_{ij} = 1, k_{ij} > 0, \qquad i \in N_D, j \in L(i). \tag{1.4}$$

$$0 \leqslant x_{ij} \leqslant u_{ij}, \qquad (i,j) \in A, \tag{1.5}$$

$$0 \leqslant x_s \leqslant u_s, \qquad s \in N_S, \tag{1.6}$$

$$0 \leqslant d_t \leqslant x_t, \qquad \forall t \in N_T, \tag{1.7}$$

where $E(i) = \{l \in N : (l,i) \in A\}, L(i) = \{j \in N : (i,j) \in A\}$. At an S-node s, we use an *inward pseudo-arc* $(\ ,s) \in (N_{PS}, N_S)$ to express the sources supplied at s. The flow x_s on $(\ ,s)$ has the upper limit u_s. Similarly, we use an *outward pseudo-arc* $(t,\ ) \in (N_T, N_{PS})$ to express the demand at a T-node t. The flow x_t on $(t,\ )$ must satisfy the minimum demand d_t. On any D-node i, we suppose $E(i)$ has one single node l, and the outgoing arcs from i satisfy the *proportional constraint* (1.4). See Figure 2 for an example of a D-node. Notice that equations (1.1)-(1.3) are conservation constraints, while (1.5)-(1.7) are upper/lower-bound restrictions on arcs.

Given a flow x, the value of x is defined to be $\nu(x) = \sum_{t \in N_T} x_t$. Therefore, the maximum flow problem in the distribution network takes the following LP form:

$$\begin{array}{lll} (P) & \max & \nu(x) \\ & \text{s.t.} & x \in F \end{array}$$

where F is the set of all flows satisfying constraints (1.1)-(1.7). We assume that problem (P) is always feasible throughout the entire discussion. See Figure 3 for an example of the distribution network. It has one S-node, five T-nodes, as well as a couple of O-nodes and D-nodes in between. For each arc (i,j), there associates a triplet (u_{ij}, d_{ij}, x_{ij}) indicating the upper bound, the lower bound, and the current flow value, respectively. The ratios for the proportional constraints can be read from the network. For instance, through the node D_1 one has to distribute 80% of the inflow to O_5 and 20% of which to O_6.

In literature, there are some special flow problems related to (P). The generalized flow problem (e.g., see [1]) has gain factors $(k_e > 1, e \in A)$ or loss factors $(0 < k_e < 1)$ for arcs in A. If one unit flow from node i to node j is sent through the arc (i,j), k_{ij} units will arrive at node j. In particular, if the arc (i,j) is lossy, namely, $0 < k_{ij} < 1$, one may introduce a dummy sink t and set $k_{it} = 1 - k_{ij}$ to make the node i a "D-like"-node of two outgoing arcs. This formulation is not the same

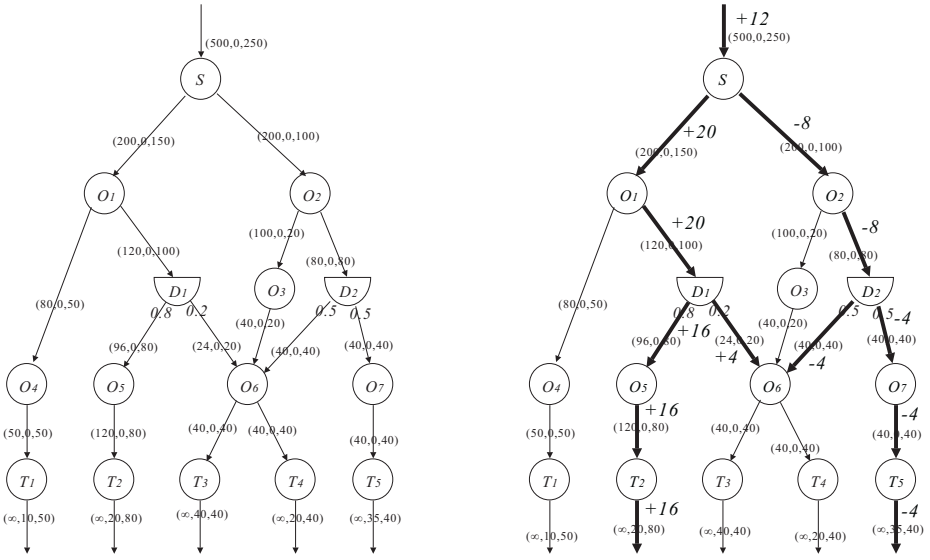

FIGURE 3. A distribution network

as our D-node since the dummy sink t, having no further outlets, violates the flow conservation required by our model.

On the other hand, Cohen and Megiddo [4] discussed a class of parametric flow problems in which the *fixed ratios flow problem* is similar to (P). This problem imposes an equivalence relation Q on A such that for every pair $(e_i, e_j) \in Q$, $x_{e_i} = k_{ij} x_{e_j}$. In the distribution network, let arcs which are incident with a common D-node be in the same equivalence class and also every single arc not adjacent to any D-node be itself an equivalence class. Then, (P) becomes a fixed ratios flow problem. Nevertheless, the result in [4] does not help much since (1) the algorithm is strongly polynomial only when the number of equivalence classes is a constant. This is surely too restrictive; (2) the strongly polynomial algorithm was proved through a sequence of reduction from other problems. A practical and implementable version of it was not studied.

It turns out that a network algorithm of (P) may not be easy to get. The D-nodes introduce a great degree of dependency among a cluster of arcs so that search algorithms on the network frequently generate cycles. Figure 3 demonstrates the complication where the flow can (indeed *must*) be increased through a cycle. It requires to reduce the amount to T_5 by 4 units so as to obtain the net gain of 12.

Our goal is to generate every possible "unsaturated subgraph" G_M of this kind as in Figure 3 so that the flow can be increased from s to t iteratively until there are no more such subgraphs. We implement a *depth-first-search-based* algorithm to search on the associated residual network. The strategies of searching for G_M will be discussed in Section 2. In Section 3, we discuss cycles, some of which are valid but some are not. In Section 4, a system of linearly homogeneous equations is used to either solve the flow increment on G_M or conclude that it is *invalid* indeed. In Section 5, we illustrate the importance of searching orders with which every possible G_M will not be missed. Section 6 describes our algorithm step by step. In Section

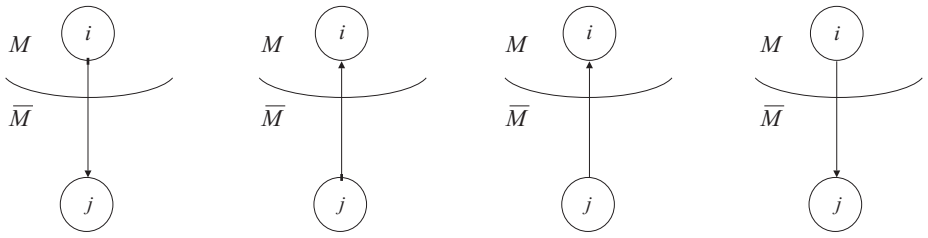

FIGURE 4. Four types of decisions at i

7, a *phase-one* procedure based on the "reverse-run" of the algorithm is given to find an initial flow. Finally, in Section 8, we conclude the paper.

2. **Basic strategies of searching.** Given a feasible flow x in $\mathcal{G}$, construct the residual network $\mathcal{G}_x = (N, A_x, u_x)$ as follows. Let

$$A_x = A_x^+ \cup A_x^-,$$
$$A_x^+ = \{a |\, a \in A, x(a) < u(a)\},$$
$$A_x^- = \{\overline{a} |\, a \in A, d(a) < x(a)\} \ (\overline{a} : \text{ a reorientation of } a),$$
$$u_x(a) = \begin{cases} u(a) - x(a), & a \in A_x^+, \\ x(a) - d(a), & \overline{a} \in A_x^-. \end{cases}$$

where $u_x(a)$ is the residual capacity of the arc a with respect to the flow x.

With the flow x, the algorithm begins with some $s \in N_s$ which is incident with a pseudo arc $(\ , s) \in A_x^+$. That is, s still has some available resources. Define $M = \{s\}$; $A_M = \{(\ , s)\}$ so that $G_M = (M, A_M) = (s, (\ , s))$ is the initial subgraph of the residual network $\mathcal{G}_x$. The G_M is maintained as a stack. At each stage in searching, the algorithm examines arcs connecting with the topmost node in the stack and includes into G_M only one arc that keeps two major network properties, the flow conservation and the proportionality at D-nodes. For example, if the topmost node is i and the algorithm is about to include the arc (i, j), we push $(j, (i, j))$ into G_M. Similarly, we push $(j, (j, i))$ into G_M for including the reverse arc (j, i). For convenience, denote the topmost entry in G_M as $topMost(G_M) = (j, (i, j))$ (or $(j, (j, i))$); the top most node as $topMostNode(G_M) = j$; the top most arc as $topMostArc(G_M) = (i, j)$ (or (j, i)); and finally denote the *second to the topmost* node to be $preTopNode(G_M) = i$. Also for the convenience in describing the searching technique, we define arcs emanating from M to $\overline{M}$ to be *forward* (denoted shortly by F) whereas a *backward* arc (B) is one that comes from $\overline{M}$ to M. In combinations, there are at most four options at the topmost node i: (a) (F, A_x^+); (b) (B, A_x^-); (c) (B, A_x^+); and (d) (F, A_x^-). See Figure 4.

The selection is to make any two consecutive pairs in G_M meet the flow conservation. Suppose $topMost(G_M) = (i, (l, i))$ is of the forward type and $i \in N_S \bigcup N_O$. To keep the flow conservation on the residual network $\mathcal{G}_x$, one must also choose a forward arc (i, j), either (F, A_x^+) or (F, A_x^-). Similarly, if $topMost(G_M) = (i, (i, l))$, then at i it should be followed by any backward arc (j, i). In other words, the S- and O-node have to be connected by both forward arcs or both backward arcs. See Figure 5.

FIGURE 5. O-nodes must be connected by two arcs of the same direction

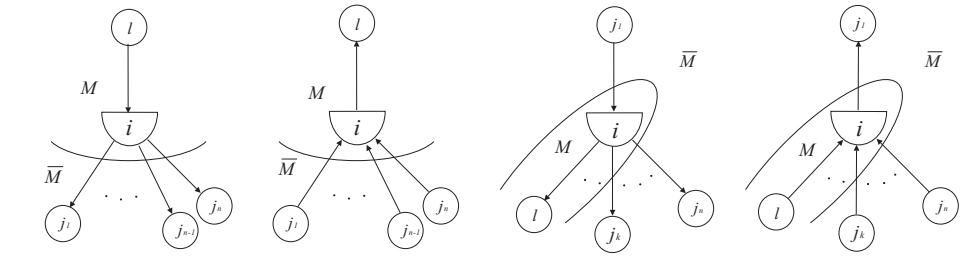

FIGURE 6. Four possible ways to enter a D-node

Suppose now i is a D-node. Listed below in Figure 6 are four possible ways to enter a D-node. Cases (3) and (4) do it from the top, while Cases (5) and (6) from the bottom. Due to the special network configuration of a D-node, if $topMost(G_M) = (i, (l, i))$ as in Case (3), all the adjacent arcs $(i, j_k), k = 1, 2, \ldots, n$ must be of type (F, A_x^+). In case (4), all arcs following i must be of type (B, A_x^-). In Case (5), (j_1, i) is of type (B, A_x^+) whereas the other $(i, j_k), k = 2, \ldots, n$ are of type (F, A_x^+). In Case (6), (i, j_1) is of type (F, A_x^-) but $(j_k, i), k = 2, \ldots, n$ are of type (B, A_x^-). This is very different from the generic augmenting path algorithm on the classical network for connecting only O-nodes [1].

In the following *advance phase*, we shall apply repeatedly the cases (1)-(6) to grow the unsaturated subgraph G_M. At $topMostNode(G_M)$, if there are no un-marked (un-visited) arcs of the right type (meaning Cases (1) - (6) above) in $\mathcal{G}_x$, $topMost(G_M)$ is said to be *saturated*. Otherwise, we include into G_M any one of the desired arcs and leave others for the next G_M to exploit. This process continues until it hits a pseudo node or forms a cycle (Step 2). In either case, the algorithm pauses temporarily and switches to a "D-cut". The D-cut is an arc $(l, k) \in \mathcal{G}_x$ such that either l or k is a D-node already included in G_M but (l, k) itself is not processed yet. If there is no such D-cut, the advance phase comes to a complete stop and the G_M is called "complete".

The advance phase

Step 1.: Use $topMost(G_M)$ to determine which cases, (1) - (6), are applicable. Push any *un-marked* arc of the right type and its associated node into G_M. If there is none, un-mark (release) the arcs adjacent to $topMostNode(G_M)$ for later accesses via different paths. Go to the *retreat phase* (below).

Step 2.: If $topMostNode(G_M) \in N_{PS}$ or $topMostNode(G_M) \in M$, look for the D-cuts of the right type by Cases (3)-(6). Push any such arc and its associated

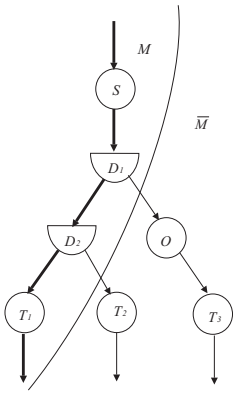

FIGURE 7.

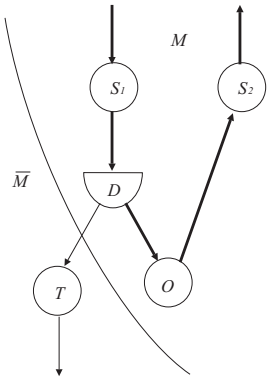

FIGURE 8.

node into G_M. Go to Step 1. If there is none, a complete G_M is obtained and the advance phase is stopped.

Step 3.: Go to Step 1.

In Figure 7, the advance phase first constructs the path, in this order: $\rightarrow S \rightarrow D_1 \rightarrow D_2 \rightarrow T_1 \rightarrow$ until $(\ ,(T_1,\ ))$ is pushed into G_M. Then, there are two D-cuts, (D_1, O) and (D_2, T_2), in which we used to choose the nearest one, (D_2, T_2), to continue. In Figure 8, we start from S_1 but have to stop temporarily when $topMost(G_M) = (\ ,(S_2,\ ))$. The search resumes from the D-cut (D, T).

When there is no un-marked arc of the desired type adjacent to $topMostNode(G_M)$, the following *retreat phase* is applied. The strategy is to pop out the saturated $topMost(G_M)$ depending on whether $preTopNode(G_M)$ is a D-node. If it is indeed a D-node, all the arcs in G_M generated after that D-node should be released due to the proportional constraint.

The retreat phase

Step 1.: If $preTopNode(G_M) \notin N_D$, mark and pop out $topMost(G_M)$. Go to the advance phase. If $G_M = \emptyset$, stop and report the current flow is maximal.

Step 2.: If $preTopNode(G_M) = l \in N_D$, keep popping out the entries in G_M one by one until we find $topMostNode(G_M) = l$. Go to Step 1.

Note that, marking the removed $topMostArc(G_M)$ at Step 1 of the retreat phase is a tracer for arcs that have been visited and discarded from $topMostNode(G_M)$. The tracer is reset when this $topMostNode(G_M)$ is to be released ("un-mark" at Step 1 of the advance phase). The reset is necessary since this $topMostNode(G_M)$ could be later included into a different G_M.

When a D-node is to be released, we can replace the capacities $\{u_{ij}|\ i \in N_D, j \in L(i)\}$ with the current flow values on $\{(i,j)|\ i \in N_D, j \in L(i)\}$. The updated capacities have now the same proportions as k_{ij}. By this, if any arc of $\{(i,j)|\ i \in N_D, j \in L(i)\}$ is saturated, so are the others. The same D-node hence will not be processed repeatedly by way of different entries to it.

3. **Cycles.** In searching G_M, we could generate cycles by adding an arc of the form (M, M). Unlike the classical maximum flow problem where one has choices to generate (e.g. the pre-flow push method [1]) or not to generate (e.g., the labeling

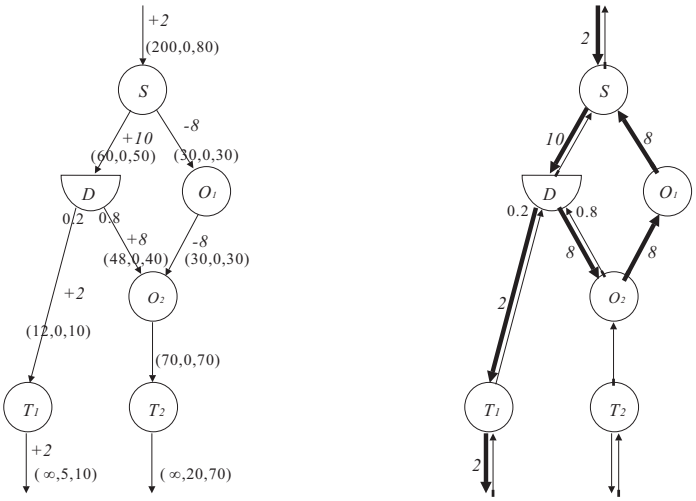

FIGURE 9. The cycle in a distribution network

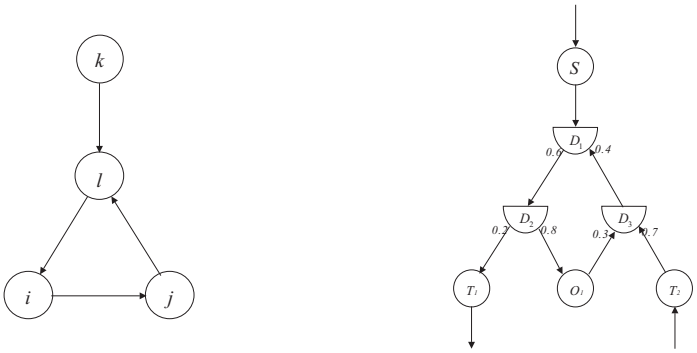

FIGURE 10. Cycles that are topologically invalid

method) cycles, we may not have an option in the distribution network. This can be best illustrated by the example in Figure 9. Observe that (O_2, T_2) is already saturated. To increase the flow, (D, T_1) is the only way. By the proportional constraint, G_M must contain both (S, D) and (D, O_2). At O_2, the only arc of the right type is (O_2, O_1) (in the residual network $\mathcal{G}_x$). Then, (O_1, S) is included and the cycle must be formed.

Nevertheless, cycles can have an obvious defect in the configuration, which we call *topologically invalid*. In Figure 10, the cycle (l, i, j, l) consists of only O-nodes. Due to the conservation law, the cycle must have the 0-flow and then becomes useless. On the other hand, the directed cycle $(D_1, D_2, O_1, D_3, D_1)$ is topologically invalid since the arc (D_3, D_1) is of the right type for D_3 but is of the wrong type for D_1. See Figure 11 for more examples of invalid cycles. On the left, the cycle (O_1, O_2, D, O_1) can be thought as an integrated node with only two incoming arcs but no outgoing arcs. The conservation law will certainly make the flow in the cycle be 0-increment. On the right of Figure 11, the cycle is topologically legitimate but

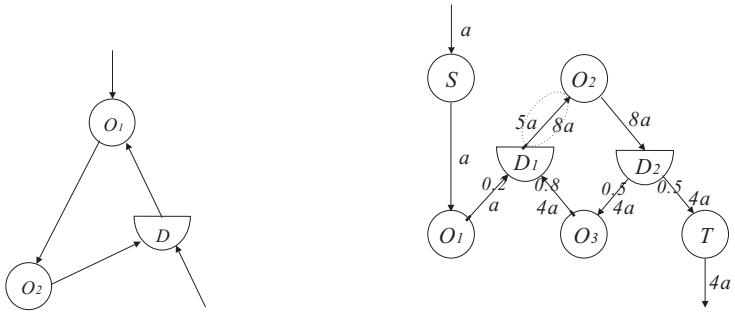

FIGURE 11. Cycles that are topologically invalid and numerically invalid

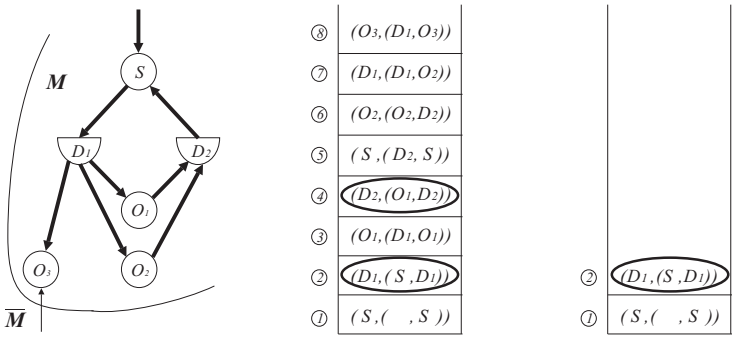

FIGURE 12. Circle the D-node to indicate the first time it entered G_M

numerically invalid. Look at (D_1, O_2). The flow out of D_1 is $5a$, but it needs $8a$ to meet the need at O_2. This happens because several D-nodes of contradicting ratios are grouped together by the search algorithm.

In general, we are not able to identify all patterns of invalid cycles. However, we observe that the invalid cycles, topologically or numerically, result in 0-increment. Our strategy is to always form the cycles and continue the search from the (nearest) D-cuts until a complete G_M is generated. The validity of G_M can be determined after the flow increment on a complete G_M is computed. (next section)

In Figure 12, suppose we first formed the cycle $\rightarrow S \rightarrow D_1 \rightarrow O_1 \rightarrow D_2 \rightarrow S$. Then, use the D-cut (O_2, D_2) and (D_1, O_2) to form the second cycle. Continue from the D-cut (D_1, O_3) but there was no arc of the right type ahead. Then, the retreat phase discarded all the arcs and nodes related to D_1 until $topMostNode(G_M) = D_1$. The circled entry in the stack G_M indicates the very place where the D-node was first included. There are two D-nodes in this example.

4. **Determining the flow value on G_M.** Suppose now we have a complete G_M that reaches at least one T-node with an A_x^+-type pseudo-arc. In this section, we show how to compute the flow increment on G_M.

Recall the $s - t$ augmenting path of the labeling method. Each node, except for the beginning and the ending node, has exactly two adjacent arcs. Knowing the flow increment on any arc of the path implies knowing that on all the others. For G_M, however, there are O-nodes having more than two adjacent arcs. See nodes S,

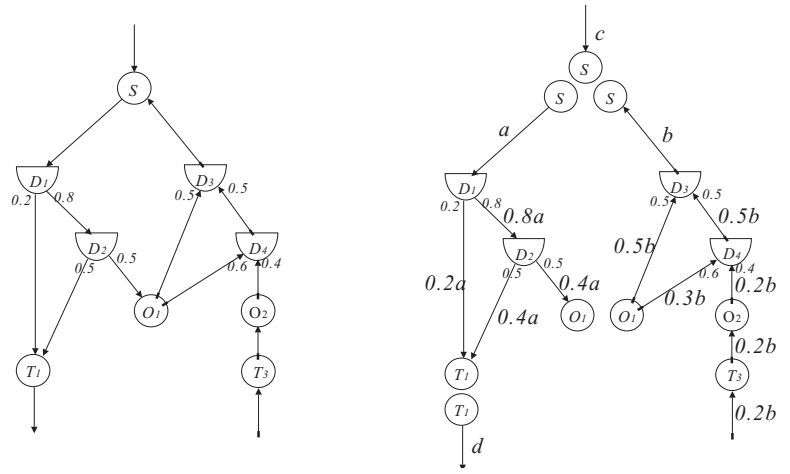

FIGURE 13. The residual network (left) and its compatible components (right)

T_1, and O_1 on the left of Figure 13. We call S, O, T-nodes having more than two adjacent arcs in G_M to be *dividing*, while those with exact two be *regular*.

To better understand the structure, we need more definitions. An undirected simple path $(u_1, u_2, \ldots, u_n)$ is said to be *compatible* if (i) u_1, u_n are the dividing node or the pseudo node; (ii) $u_i, i \neq 1, n$, are D-nodes or regular nodes. The compatible path can be viewed as a generalization of the conventional $s-t$ augmenting path. In particular, knowing the flow value on any arc of a compatible path implies knowing the values on all the other arcs. For instance, the path $P_1 : (S, D_3, D_4, O_2, T_3, \ )$ is compatible, while the path $P_2 : (S, D_3, O_1, D_2, T_1)$ is not since O_1 is a dividing node. On P_1, suppose the flow on (D_3, S) is b, then the flow on (D_4, D_3) and (O_2, D_4) will be $0.5b$ and $0.2b$, respectively. However, on P_2, assuming the flow on (D_3, S) to be b implies $0.5b$ on (O_1, D_3). The information is insufficient to determine the flow on (D_2, O_1) since there is another arc $(O_1, D_4) \notin P_2$ connecting to the dividing node O_1.

More generally, we shall call two arcs (i_1, j_1) and (i_2, j_2) *compatible*, denoted by $(i_1, j_1) \sim (i_2, j_2)$, if they are on the same compatible path. It is easily seen that the compatibility "$\sim$" defines an equivalence relation on A_M, which is thus partitioned into several *compatible components*. See the right picture in Figure 13 where there are four compatible components. In each component, if the flow value is determined for one arc, so are the others. For this reason, the flows on each component can be expressed in terms of a single independent variable (label), so we call it the *multiple labeling method*.

In Figure 14, the four components join with each other at three dividing nodes. By the conservation law, at S we have $c + b = a$. At O_1, we have $0.4a = 0.5b + 0.3b$ and at T_1, we have $0.6a = d$. They can be further simplified to become $b = c = 0.5a$ and $d = 0.6a$ with which the four components are represented by just one variable a multiplied by a constant, say $r_{ij}a$. Determine the value a by solving $r_{ij}a \leqslant u_{ij}$ with the residual capacity u_{ij}. Taking the minimum of $\frac{u_{ij}}{r_{ij}}$ yields the largest possible flow that can be increased on G_M. In this example, $a = 20$.

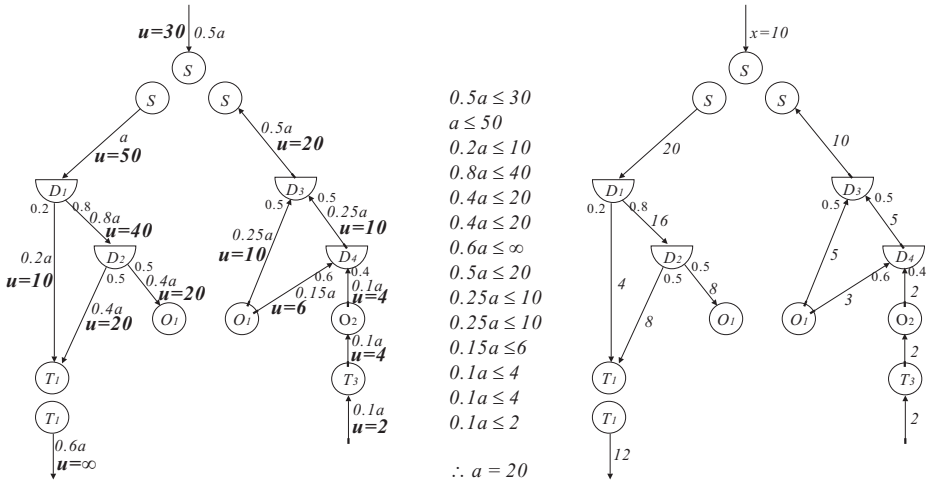

FIGURE 14. All arcs in G_M can be expressed by $r_{ij}a$ (left) and take the minimum of $\frac{u_{ij}}{r_{ij}}$ to get x_{ij} (right)

In general, if there are n compatible components and m dividing nodes, the conservation law at the dividing nodes gives a system of m linear homogeneous equations with n non-negative unknowns. This system has at least one trivial solution $(0, 0, \ldots, 0)$. But in order for G_M to be valid and useful, we need the system to have a positive solution. To this end, it is necessary that $m \leqslant n - 1$, providing that the system is of full rank.

Suppose that $m \geqslant n$. Then, choosing n linear independent equations out of m and solving the n unknowns will give a unique zero solution. Figure 15 is such an example in which $m = n = 2$. The two dividing nodes are S and O_1 and the two equations are $a + b = a$ and $0.4a = 0.6a$. The unique solution is $a = b = 0$.

Figure 15 also provides an interesting example for the dividing node which we call *degenerate*. From the definition of compatibility, two compatible components must be connected at dividing nodes. This example shows the converse is not necessarily true. The dividing node O_1 in Figure 15 is not the junction node of different components. The conservation constraint at O_1 forces the flow to be 0.

If $m = n - 1$, the system has one degree of freedom and the flow on G_M can be expressed by one single variable. Unfortunately, the non-negativity restrictions could still be violated. Figure 16 provides such an example. On the left is the original network, whereas on the right is the G_M. Node S is the only dividing node and there are two components. The equation at S is $b = -5a$. Only $(a, b) = (0, 0)$ satisfies the non-negative restrictions, so the G_M is not valid.

Finally, we briefly discuss the case $m \leqslant n - 2$ which gives us at least two degrees of freedom to determine the flow increment. Using our strategy of searching, we can not find a particular G_M having this phenomenon, nor can we prove that it is not possible. We can only conjecture that it is very unlikely since we form the cycle at the dividing node without penetrating to go on. Besides, if we do have two degrees of freedom from the equations, it might indicate that there are two chunks of components that are independent of each other and can be treated separately.

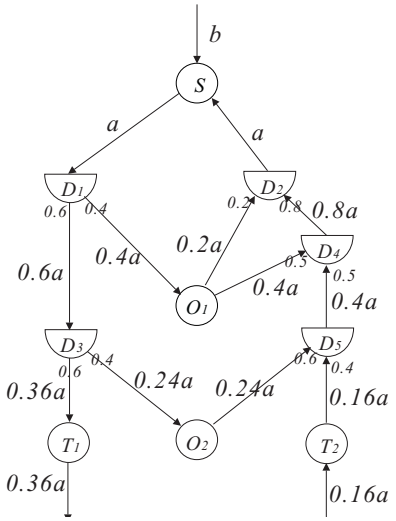

FIGURE 15. An example of degenerate dividing nodes

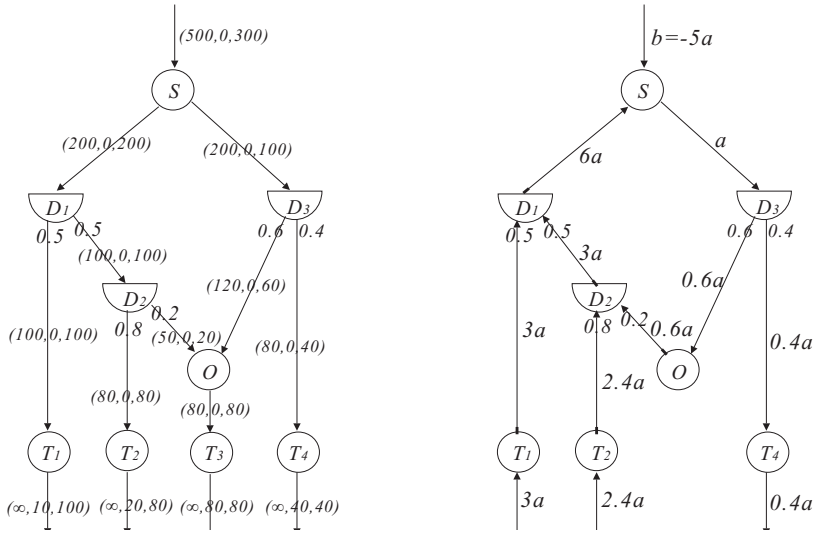

FIGURE 16. An example of the negative total flow increment

5. **Searching orders among D-cuts.** In the advance phase when an O-node is encountered, every adjacent arc in $\mathcal{G}_x$ has to be checked and marked to guarantee a unique visit. However, when meeting a D-node, we arbitrarily choose one adjacent arc to continue and leave the others as D-cuts which must be all checked in whatever order. It is surprising to see that some valid G_M can be obtained *only if* a particular order among the D-cuts is followed. To illustrate the importance of searching orders, let us work out an eccentric example as shown in Figures 17 - 19. The numbers circled nearby the arcs indicate the searching order in constructing the G_M.

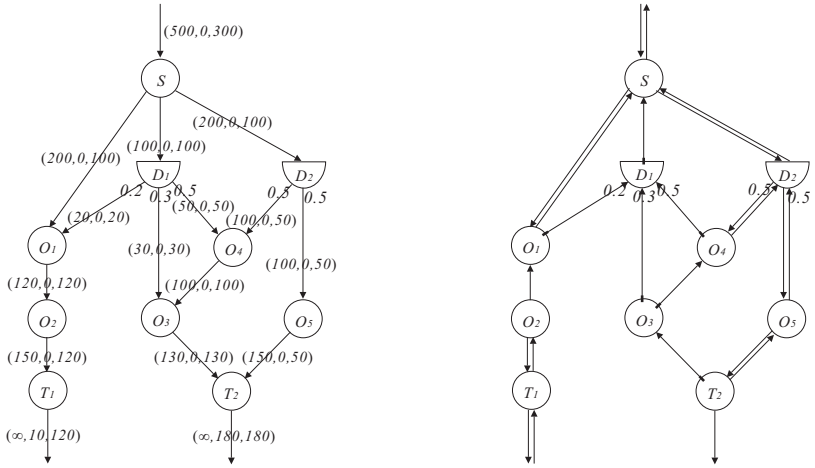

FIGURE 17. An example for the importance of the searching order of D-cuts

Figure 17 is the target problem together with its residual network. Suppose the search began with $\rightarrow S \rightarrow D_2 \rightarrow O_4$. It follows that there are only two G_M's as shown in Figure 18 that could result from this condition. Both are invalid. In particular, the left one in Figure 18 contains no pseudo arc of A_x^+, while the right one even fails to reach any pseudo-arc.

In fact, a valid G_M must include either $(T_1, \ )$ or $(T_2, \ )$. However, since (O_2, O_1) is of the wrong type, there is no way to reach $(T_1, \ )$. To get the other pseudo arc $(T_2, \ )$, one must enter T_2 from O_5 rather than from O_3 because the two arcs (O_5, T_2) and $(T_2, \ )$ are of the same direction whereas the other pair (T_2, O_3) and $(T_2, \ )$ are not. See Figure 19. Similarly, suppose G_M commences from $\rightarrow S \rightarrow O_1 \rightarrow D_1$. There are three options at D_1: $(D_1, S), (O_4, D_1)$ and (O_3, D_1). To reach T_2 from O_5, (O_4, D_1) must go ahead of (O_3, D_1). The reverse order, i.e., (O_3, D_1) in advance of (O_4, D_1), will not do.

Therefore, when retreating a D-node from a complete G_M, it is necessary to enumerate all possible searching orders among the D-cuts. For example, the D-node in Figure 20 has three D-cuts. The six combinations have to be totally examined before we can safely discard it in the retreat phase. In general, if a D-node l has ε adjacent arcs, there are $\varepsilon - 1$ D-cuts and $(\varepsilon - 1)!$ searching orders to explore.

6. **The algorithm.** Our search method is a depth-first-search-based algorithm. It claims the maximum flow x after exploiting every possible G_M on $\mathcal{G}_x$. The data structure of G_M is a stack. Each stack, when finished, is like a path from the root to a leaf in a depth-first-search tree. Starting from $G_M = (s, (\ , s))$ (the root), the *advance phase* repeatedly searches for an unsaturated arc until it hits either a T-node, or a S-node, or forms a cycle. If there is a D-cut, the search has not reached a leaf yet and has to be continued from any one of the D-cuts. Otherwise, the search is complete and a path from the root to a leaf is found.

The *retreat phase* is equivalent to the backtracking mechanism of the depth-first-search algorithm. It applies either when the top most entry in G_M is saturated or when it is already completed but the resulting G_M is invalid. To distinguish, we set the former to $Flag = UC$, while the latter $Flag = C$.

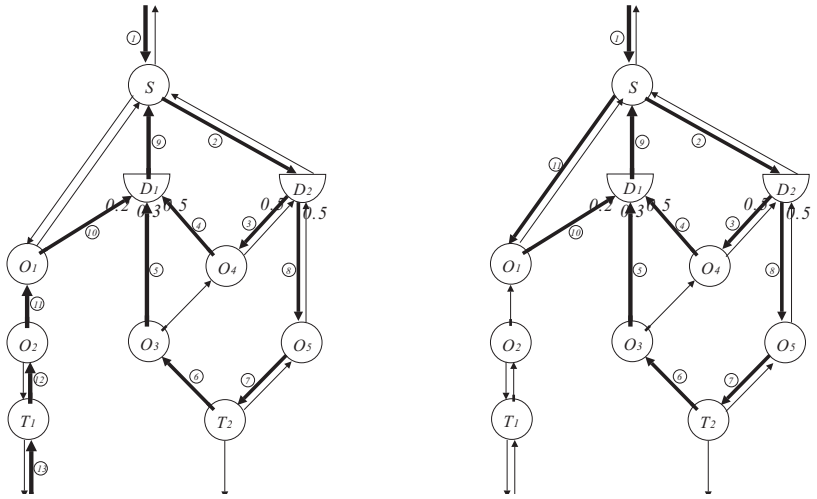

FIGURE 18. From (D_2, O_4), we can only obtain invalid G_M's

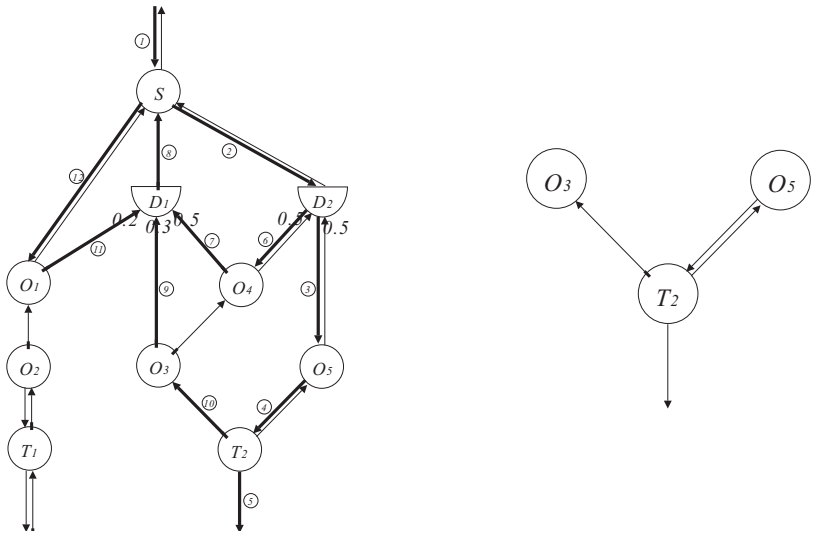

FIGURE 19. The only way to get $(T_2, \ )$ is to come from O_5, not O_4

In the retreat phase, if the $preTopNode(G_M)$ is not a D-node, it pops out the $topMost(G_M)$ and starts off the advance phase from the earliest possible un-marked arcs of the right type (Step 3 of the following algorithm). If $preTopNode(G_M)$ is a D-node, it pops out every entry in G_M all the way to the place where the D-node was first visited. Then, if it is of $Flag = C$, we have to check all other orders among the D-cuts and resume the advance phase (Step 5). Otherwise, for $Flag = UC$, we continue the retreat phase by going back to Step 3. The reason is that, if one searching order could not successfully lead to a leaf, it must be saturated at some

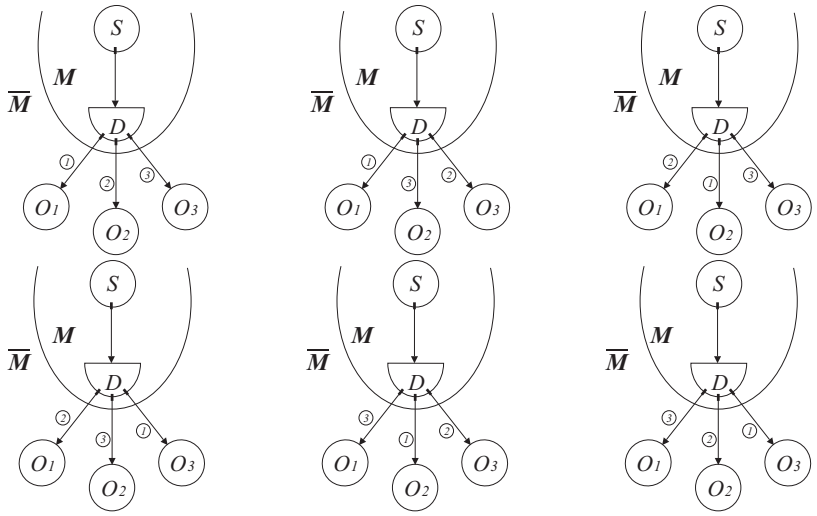

FIGURE 20. All searching orders among the D-cuts

place. The saturation can not be got away by trying other searching orders among the D-cuts.

Below is the step-by-step procedure of our algorithm.

The Algorithm

Step 0.: (Initialization) Un-mark all the arcs. For each $d \in N_D$, set $Flag(d) = UC$. Enumerate all searching orders among the incident arcs of d. Determine any order, say $\pi(d)$, to begin with. Let $s \in N_s$ and $(\ , s)$ be a pseudo-arc of type A_x^+. Push $(s, (\ , s))$ into the stack G_M.

Step 1.: (Searching repeatedly for unsaturated arcs) If $topMostNode(G_M) \in N_S \bigcup N_O$, select any *un-marked* arc according to Cases (1)-(2) and push it into G_M. If $topMostNode(G_M) \in N_D$, select and push the arc following the pre-determined order as well as Cases (3)-(6). Redo Step 1. If all the arcs incident to $topMostNode(G_M)$ are either marked or of the wrong type, un-mark (release) the arcs adjacent to $topMostNode(G_M)$. Go to Step 3.

Step 2.: (Searching from a D-cut) If $topMostNode(G_M) \in N_{PS}$ or $topMostNode(G_M) \in M$, look for the D-cuts using the pre-determined order. If the next D-cut is of the right type by Cases (3)-(6), push it into G_M. Go to Step 1. If there is no D-cut, a complete G_M is obtained. Go to Step 6.

Step 3.: (Backtracking from non-D-cuts) If $preTopNode(G_M) \notin N_D$, mark and pop out $topMost(G_M)$. Go to Step 1. If $G_M = \emptyset$, stop and report the current flow is maximal.

Step 4.: (Backtracking from D-cuts) If $preTopNode(G_M) = l \in N_D$, keep popping out the entries in G_M one by one until we find $topMostNode(G_M) = l$. If $Flag(l) = UC$, Go to Step 3. Otherwise, go to Step 5.

Step 5.: (Backtracking from D-cuts using different orders) If there is any unused order for the current D-node, pick one and go to Step 1. Otherwise, go to Step 3.

Step 6.: (Compute the flow increment) Use the multi-labeling method in Section 4 to determine the flow increment on G_M. If the value is positive, update the current flow to come out with a new residual network. Go to Step 0. If the value is 0, set every D-node in G_M with $Flag = C$ and go to Step 3.

The backtracking strategies in Steps 3 - 5 carefully check backward for a new search off the nearest available node in the stack under different situations. The maximum flow x is found when the search eventually ends with G_M being an empty set. This guarantees a complete search for all the subgraphs on the residual network and thus provides the correctness for the algorithm.

In general, we are unable to guarantee that the same G_M will never be produced again since the same cycle could be formed through different orientations (G_M's). The depth-first-search is thus in a sense of the data structure, not in a sense of the configuration.

7. **Initial flow.** In this section, we address the *phase-one* procedure that finds an initial flow on a distribution network. It is not a maximum flow problem. The difficulty occurs especially in the positive lower bounds $d_t > 0$. Take Figure 21 as an example. The maximum flow is $a = 50$ but T_2 receives nothing from this allocation whereas T_1 and T_3 obtain something more than enough ($d_{T_1} = d_{T_3} = 20$). If they were otherwise supplied by an amount between $20 \leqslant a \leqslant 25$, T_2 would be satisfied as well. This shows that the flow balance among T-nodes, rather than a larger flow value, is more critical in the phase-one.

Let us begin with the 0-flow and suppose some positive lower bounds on the T-nodes are violated. Add dotted arcs to those T-nodes and associate each T-node with a negative number $-n, n > 0$ to indicate that the current flow on the T-node is short of n. See nodes T_1, T_2 and T_3 in Figure 21 for an example.

The strategy is to run the searching algorithm "upside-down" starting from any T-node having a dotted arc. It means to use $(T, (T, \ ))$ as the first entry and search for a complete G_M (compared with $(S, (\ , S))$ when starting from S). See Figure 21 for an example in which the flow increment on G_M is $a = 50$.

After updating the flow by $a = 50$, we found T_2 is the only T-node with insufficient supply. See the dotted arc on the leftmost graph of Figure 22. Starting from T_2, we apply the advance phase to obtain the path $\leftarrow T_2 \leftarrow O_1 \leftarrow T_1 \leftarrow$ with $a = 30$, as shown in the middle graph of Figure 22. In fact, this path $\leftarrow T_2 \leftarrow O_1 \leftarrow T_1 \leftarrow$ reallocates 30 from T_1 to T_2 so as to achieve a feasible flow (the right most graph). It could not be accomplished if the search had started from the S-node, since the bottle neck arc (O_1, D) is not of the right type. This example shows the necessity of searching from the bottom for finding the initial flow.

8. **Conclusion.** In this paper, we present a search algorithm to solve the maximum flow problem in the distribution network. It is of the depth-first-search type which explores all searching orders so as to identify every valid G_M. Although the search strategy may not be very efficient, the examples we provide are the most representative ones showing the necessity to enforce this kind of complete enumeration on graphs. The algorithm also comes with various graph techniques such as computing the flow increment and finding an initial flow, making it ready for implementation. We believe that the study conducted here is very original and illustrates important graph properties with the appearance of the D-nodes and the cycles.

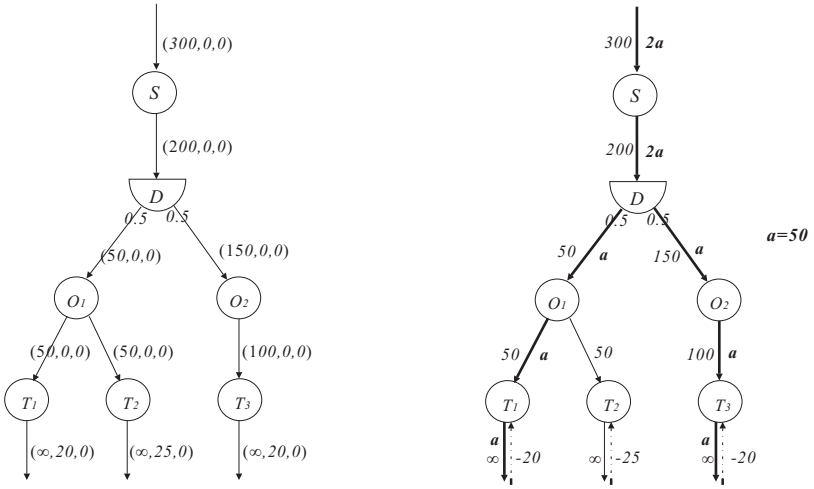

FIGURE 21. Node T_1 is supplied by more than enough flows

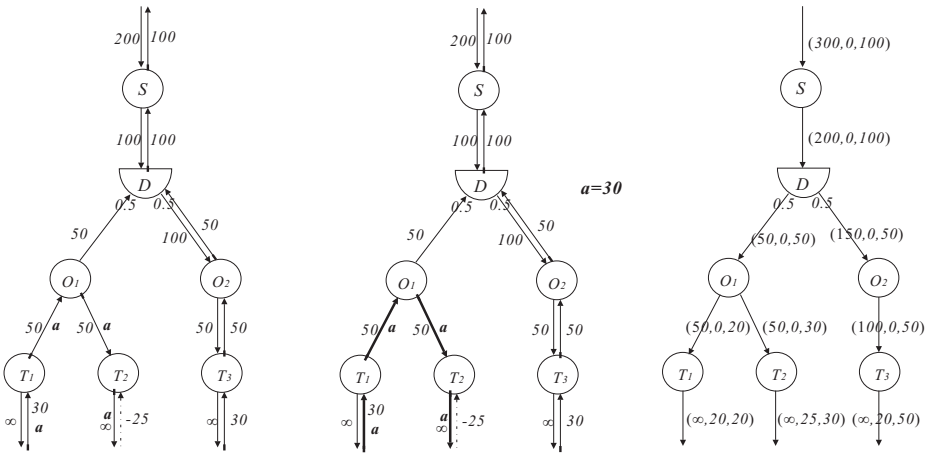

FIGURE 22. Reallocation of outputs between T_1 and T_2

There are several interesting research directions afterward. Finding a more intelligent algorithm to avoid the complete enumeration is definitely at the top of the list. Establishing the fundamental graph relation of the maximum flow - minimum "cut" (with a new definition for the "cut") is also primary among many others.

REFERENCES

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

[2] R.G. Askin and C.R. Standridge, Modelling and Analysis of Manufacturing Systems, John Wiley & Sons, New York, USA, 1993.

[3] M.S. Bazaraa, J.J. Jarvis and H. Sherall, Linear Programming and Network Flow, John Wiley & Sons, New York, USA, 1990.

[4] E. Cohen and N. Megiddo, *Algorithms and complexity analysis for some flow problems*, Algorithmica, 11 (1994) 320–340.

[5] S.C. Fang and L. Qi, *Manufacturing network flows: A generalized network flow model for manufacturing process modelling*, Optim. Methods Softw., 18 (2003) 143–165.
[6] L.R. Ford and D. Fulkerson, Flows in Networks, Princeton University Press, Princeton, USA, 1962.
[7] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, USA, 1976.
[8] C. Leondes, Computer Integrated Manufacturing, CRC Press, New York, USA, 2001.
[9] D.G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley, Canada, 1984.
[10] J. Mo, L. Qi and Z. Wei, *A network simplex algorithm for simple manufacturing network model*, Journal of Industrial and Management Optimization, 1 (2005), 251–275.
[11] K.G. Murty, Network Programming, Prentice Hall Englewood Cliffs, New Jersey, 1992.
[12] C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1982.

Received September 2005; revised December 2005.

E-mail address: rsheu@math.ncku.edu.tw