Optimization Methods and Software, 2003, Vol. 18, No. 2, April, pp. 143–165

Taylor & Francis
Taylor & Francis Group

MANUFACTURING NETWORK FLOWS: A GENERALIZED NETWORK FLOW MODEL FOR MANUFACTURING PROCESS MODELLING

SHU-CHERNG FANG[a] and LIQUN QI[b,†,*]

[a]*Department of Industrial Engineering and Graduate Program in Operations Research, North Carolina State University, Raleigh, North Carolina, USA;* [b]*Department of Applied Mathematics, Hong Kong Polytechnic University, Kowloon, Hong Kong*

(Received 10 October 2002; In final form 28 January 2003)

Network flow models have been shown to be theoretically interesting and practically useful. However, an ordinary network flow has its limitation in modelling more complicated manufacturing scenarios, in particular the synthesis of different materials to one product and/or the distilling of one material to many different products. In this paper, we present a generalized network model called *manufacturing network flow* (MNF) for this purpose. The underlying structure and dual properties of the so-called minimum distribution cost problem is specifically studied to outline a network simplex method for solving this problem.

Keywords: Network flow; Manufacturing; Process and product management; Mathematical modelling

1 INTRODUCTION

Network flow models have been shown to be theoretically interesting and practically useful [1,3–5,7]. However, an ordinary network flow has its limitation in modelling more complicated manufacturing scenarios, in particular the synthesis of different materials to one product and/or the distilling of one material to many different products [2,6]. In this paper, we present a generalized network model called *manufacturing network flow* (MNF) for this purpose. In such a model, multiple products may be produced by using multiple materials via a combination of synthesis and distilling operations.

In order to properly model various manufacturing scenarios, we introduce six different kinds of nodes in Section 2. The ordinary nodes are for transition of flows, source nodes for providing raw materials, termination nodes for collecting final products, intermediate nodes which allow to use excess in-process materials or semi-products from the last period, or to store excess in-process materials or semi-products for the next period, distribution nodes

†The work of this author was done during his visit to the Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, New Territory, Hong Kong.
E-mail: fang@eos.ncsu.edu

*The work of this author was supported by the Research Grant Council of Hong Kong.
E-mail: maqilq@polyu.edu.hk

ISSN 1055-6788 print; ISSN 1029-4937 online
DOI: 10.1080/1055678031000152079

for distribution or distilling operations, and combination nodes for assembly or synthesis operations. The most frequently studied "maximum flow" and "minimum cost" problems in ordinary network flows are generalized in this framework. Linear programming formulations are presented in Section 3. This allows us to apply linear programming methodologies to solve related optimization problems in manufacturing networks. However, we are more interested in the possibility of finding efficient algorithms, in particular, strongly polynomial-time algorithms, for solving these problems.

Due to the complexity of a manufacturing network, we introduce two simplified versions, namely, the distribution and assembly network flow problems in Sections 4 and 5, respectively. The underlying structure and dual properties of the so-called minimum distribution cost problem is specifically studied to outline a network simplex method for solving this problem in Sections 6, 7 and 8. More specifically, we show in Section 6 that, for a connected general network, the subgraph corresponding to a basic feasible solution of the minimum distribution cost problem is connected. This subgraph is called a "basic feasible graph". In Section 7, we develop well-defined procedures to compute the primal basic feasible solution and dual basic solution corresponding to a given basic feasible graph. These computational procedures are almost as efficient as their counterparts are in the ordinary network flow problem. Based upon these procedures and some optimality conditions, a network simplex method is outlined in Section 8. Various pivoting schemes are also discussed.

It is our hope that the proposed framework can be applicable to industrial applications and can be of interests to researchers for theoretical development. Some final remarks are given in the last section.

2 NODES AND ARCS

Let $G = (N, A)$ be a general network with N denoting the node set and A the arc set. For an arc $(i, j) \in A$, we say i is the tail node and j the head node. Also, we denote the flow value on arc (i, j) by x_{ij} with a given upper bound $u_{ij} > 0$, i.e.,

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A. \tag{2.1}$$

In our model, we allow multiple materials (products) flow through the network, but only one material (product) on an individual arc.

For a node $i \in N$, define

$$E(i) := \{j \in N: (j, i) \in A\}$$

to be the set of all "entering" nodes to node i, and

$$L(i) := \{j \in N: (i, j) \in A\}$$

the set of all "leaving" nodes from i. There are six kinds of nodes in the generalized network model.

1. O-nodes are ordinary nodes for transition. Such a node may have several arcs coming in and going out, but only one kind of material/semi-product flowing through with balance, i.e.,

$$\sum_{j \in E(i)} x_{ji} = \sum_{j \in L(i)} x_{ij} \quad \forall_i \in N_O, \tag{2.2}$$

where N_O represents the set of all O-nodes. We may use an (O-shape) disk to represent an O-node. See Fig. 1 for an O-node. A transportation or transshipment operation in manufacturing can be a good example.

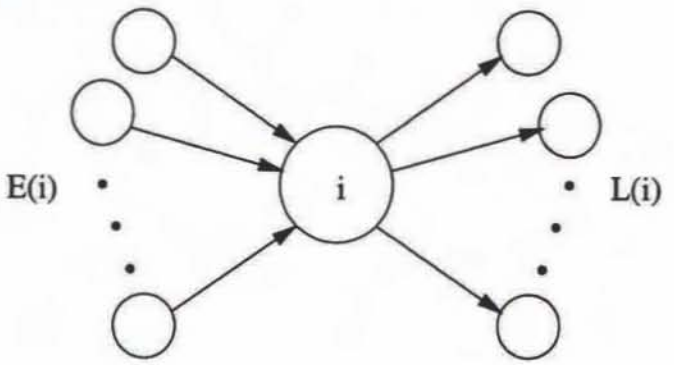

FIGURE 1 An *O*-node.

2. *S*-nodes are source nodes for raw materials. A manufacturing network G may have one or several source nodes, each representing one different raw-material. Let N_S be the set of all *S*-nodes. For each $i \in N_S$, $E(i) = \emptyset$ and there associates an input flow of x_i such that

$$x_i = \sum_{j \in L(i)} x_{ij} \quad \forall i \in N_S. \tag{2.3}$$

See Fig. 2 for an *S*-node. Also note that only one kind of raw-material is allowed at an *S*-node.

3. *T*-nodes are termination nodes for final products. A manufacturing network G may have one or several termination nodes, each representing one different final product. Let N_T be the set of all *T*-nodes. For each $i \in N_T$, $L(i) = \emptyset$ and there associates an output flow of x_i such that

$$x_i = \sum_{j \in E(i)} x_{ji} \quad \forall i \in N_T. \tag{2.4}$$

See Fig. 3 for a *T*-node. Again, only one kind of final product is allowed to flow in a *T*-node.

Notice that *O*-nodes, *S*-nodes, and *T*-nodes are commonly used in ordinary network problems [1,3–5,7,8]. Following are three kinds of new nodes.

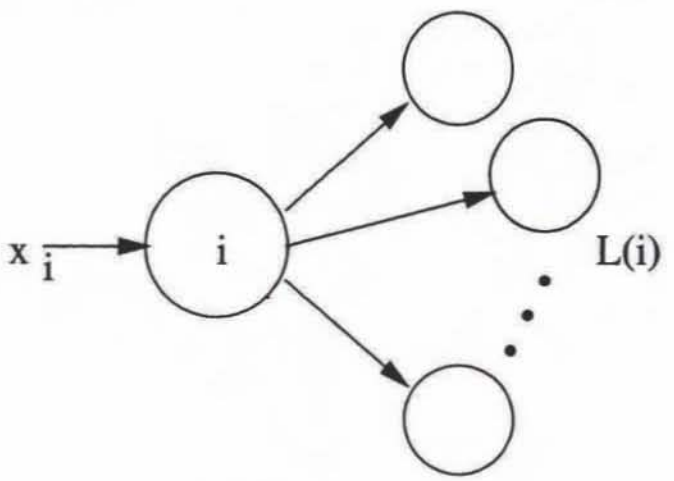

FIGURE 2 An *S*-node.

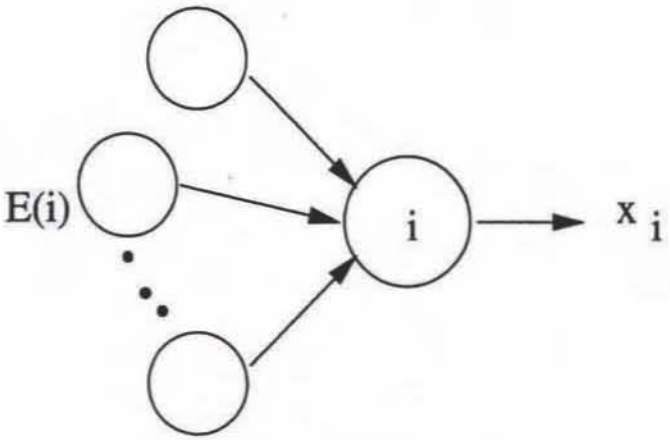

FIGURE 3 An *T*-node.

4. *I*-nodes are intermediate nodes for in-process materials/semi-products. *I*-nodes are like *O*-nodes, but in such a node, the total in-flow value and the total out-flow value can have a difference x_i, i.e.,

$$\sum_{j\in E(i)} x_{ji} = x_i + \sum_{j\in L(i)} x_{ij} \quad \text{and} \quad l_i \le x_i \le u_i, \quad \forall i \in N_I, \tag{2.5}$$

where $l_i \le 0$, $u_i \ge 0$, $x_i > 0$ means the excess quantity which can be used in the next period, $x_i < 0$ means the excess quantity from the last period which can be used now, l_i and u_i are the lower and upper limits of x_i and N_I represents the set of all *I*-nodes.

See Fig. 4 for an *I*-node in which only one kind of in-process material/semi-product can flow through, with an excess quantity x_i stored in the node.

5. *D*-nodes are distillation nodes. A *D*-node has only one arc in (for one kind of material) but multiple arcs out (for different kinds of products), and the flow value on an output arc is proportional to flow value on the input arc. Let N_D be the set of all *D*-nodes. For each $i \in N_D$ we have

$$E(i) = \{i^*\},$$

and

$$x_{ij} = k_{ij}x_{i^*i} \quad \forall j \in L(i), \tag{2.6}$$

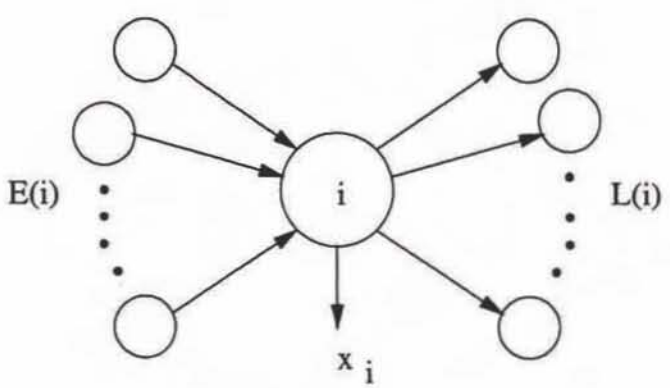

FIGURE 4 An *I*-node.

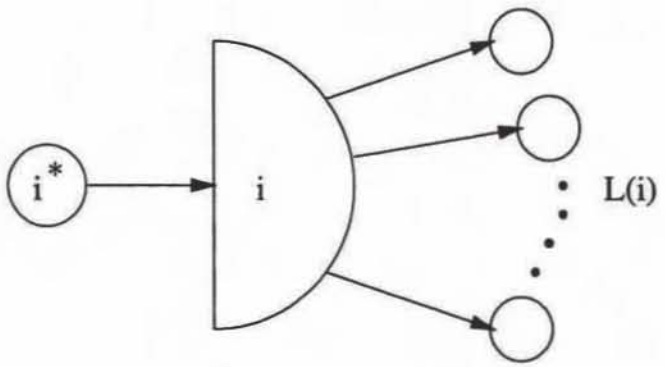

FIGURE 5 A *D*-node.

where k_{ij} is a positive real number. We may use a (D-shape) right-half disk to represent a *D*-node with only one arc (i^*, i) pointing to the center of the half disk on the flat side and several arcs going out from the half-circle boundary of the disk.

See Fig. 5 for a *D*-node. A distilling operation that decomposes one material into several products with fixed ratios can be a good example.

6. *C*-nodes are combination nodes. A *C*-node has only one arc out (for one kind of product) and multiple arcs in (for different kinds of materials), and the flow value on an input arc is proportional to the flow value of the output arc. Let N_C be the set of all *C*-nodes. For each $i \in N_C$, we have

$$L(i) = \{i^*\},$$

and

$$x_{ji} = h_{ji}x_{ii^*} \quad \forall j \in E(i), \tag{2.7}$$

where h_{ji} is a positive real number. We may use a (C-shape) left-half disk to represent a *C*-node with only one arc (i, i^*) going out from the center of the half disk on the flat side and several arcs pointing to the half-circle boundary of the disk.

See Fig. 6 for a *C*-node. A synthesis unit that assembles several materials with fixed ratios to form a new product can be a good example. Also notice that due to the proportion

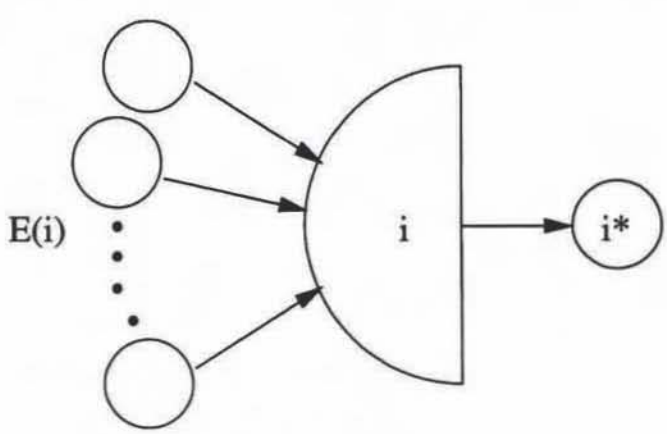

FIGURE 6 A *C*-node.

requirement, an input arc to a C-node is usually coming from an I-node that stores the unused material/semi-product.

By chaining C-nodes and D-nodes together with other nodes, we can describe a manufacturing scenario involving multiple materials and products without much difficulty.

3 GENERAL MNF OPTIMIZATION PROBLEMS

Given a manufacturing network $G = (N, A)$, a feasible flow x is a mapping from A to $\Re_+$ such that the general network constraints (2.1)–(2.7) are satisfied. Let F be the set of feasible flows. We are interested in the following two MNF optimization problems.

A. Maximum flow value problem. Assume that there is an upper bound of $u_i > 0$ for each available resource material, and a unit value (weight) of w_j for each final product. The maximum flow value problem finds a feasible flow that maximizes the total production value, i.e.,

$$\begin{aligned} \max \quad & \sum_{j \in N_T} w_j x_j \\ \text{subject to} \quad & x \in F, \\ & x_i \le u_i, \quad \forall i \in N_S. \end{aligned} \tag{3.1}$$

B. Minimum cost flow problem. Assume that there is a market demand of $d_j > 0$ for each final product, and a unit cost of c_i for each raw material. The minimum cost flow problem finds a feasible flow that minimizes the total cost of raw materials, i.e.,

$$\begin{aligned} \min \quad & \sum_{i \in N_S} c_i x_i \\ \text{subject to} \quad & x \in F, \\ & x_j \ge d_j, \quad \forall j \in N_T. \end{aligned} \tag{3.2}$$

These two problems are both linear programming problems with a general network flow structure. But they are more complicated than the classical network problems [1,3–5,7,8]. An interesting issue is to investigate whether they can be solved more efficiently than a general linear programming problem. In particular, is it possible to develop strongly polynomial-time algorithms [1,5,8] for solving such problems?

To simplify the situation, we present two special cases, namely the distribution network flow and assembly network flow problems, in the next two sections for further studies.

4 DISTRIBUTION NETWORK FLOW PROBLEMS

Consider a general supplier who wants to distribute certain amount of a particular product from an S-node s to several retailers at different T-nodes through a distribution network in which O-nodes are used for transition and D-nodes are used for proportional distribution at middle-man's places. For simplicity, we assume that

$$\sum_{j \in L(i)} k_{ij} = 1 \quad \forall i \in N_D, \tag{4.1}$$

and $w_j = 1$, $\forall j \in N_T$.

In this scenario, since no C-nodes, and hence I-nodes, involved, we let F_D be the set of all flows that satisfies the constraints (2.1)–(2.4), (2.6), and (4.1). Then we face the following problem:

C. Maximum distribution flow problem.

$$\begin{aligned} \max \quad & \sum_{j \in N_T} x_j \\ \text{subject to} \quad & x \in F_D, \\ & x_s \le u_s, \end{aligned} \tag{4.2}$$

where u_s is the upper bound of the available resource, which may be infinity.

Assuming that in a distribution network there is no directed cycle going through any D-node and all nodes in the network are reachable in the sense that for each T-node, there is a directed path from s to it; for each O-node, there is a directed path from s to it, and a directed path from it to a T-node; for each D-node, there is a directed path from s to it; and for each output node of the D-node, there is a directed path from it to a T-node. In this case, (4.2) is feasible and the maximum flow value will be affected by the upper bound on each arc. Furthermore, there exists a partial order among the D-nodes in such distribution networks. We may draw a corresponding manufacturing network like Fig. 7 with s on the top and multiple termination nodes at the bottom.

By adding a minimum demand of $d_j > 0$ at the termination node j for the corresponding retailer, a slightly more general setting becomes

$$\begin{aligned} \max \quad & \sum_{j \in N_T} x_j \\ \text{subject to} \quad & x \in F_D, \\ & x_j \ge d_j, \quad \forall j \in N_T, \\ & x_s \le u_s. \end{aligned} \tag{4.3}$$

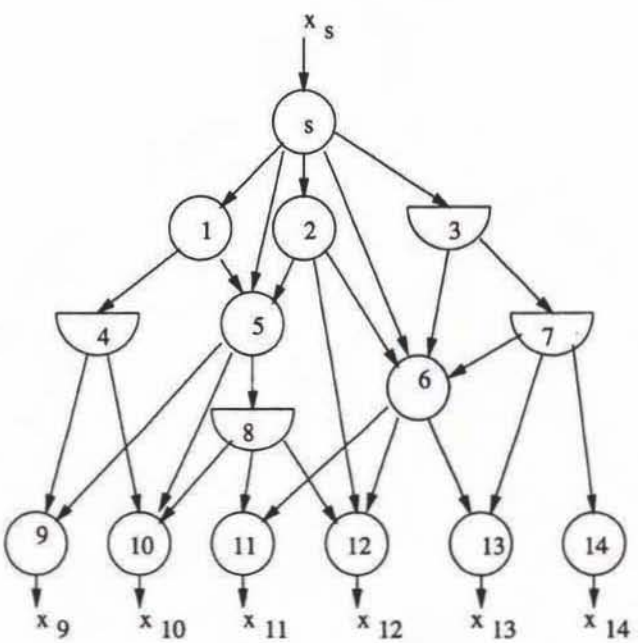

FIGURE 7 A distribution network.

Notice that problem (4.3) may be infeasible. Thus a feasibility problem exists before finding an optimal solution. Also notice that the objective function may be modified to form the following minimization problem:

D. Minimum distribution cost problem.

$$\begin{aligned} \min \quad & c_s x_s + \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{subject to} \quad & x \in F_D, \\ & x_j \geq d_j, \quad \forall j \in N_T, \\ & x_s \leq u_s. \end{aligned} \tag{4.4}$$

where c_s is the unit cost of the raw material and c_{ij} is the cost (such as transportation and material handling fees) per unit flow going through arc (i, j).

The maximum distribution flow problem can be viewed as a generalization of the classical maximum flow problem. Similarly, because of (4.1), the minimum distribution cost problem can be treated as a classical minimum cost network flow problem with slack variables at T-nodes and additional linear constraints (2.6). We are interested in knowing if the classical theory of the maximum flow value and minimum cost flow problems can be generalized here, and if there exist strongly polynomial-time algorithms in this setting.

5 ASSEMBLY NETWORK FLOW PROBLEMS

An assembly network looks like a reversed distribution network. Consider a manufacturer who is producing one final product at the termination node t by assembling various parts and components obtained at different source nodes $i \in N_S$ through a manufacturing network G in which same kind of parts/components transporting through O-nodes satisfying the balance equation (2.2), and different kinds of parts are assembled at C-nodes to form a semi-product/product according to fixed ratios given by (2.7).

Note that C-nodes are connected to I-nodes to use the excess materials or semi-products from the last period or to store them for the next period, so we let F_C be the set of all flows that satisfies the constraints (2.1)–(2.5) and (2.7). Then we face the following problem:

E. Maximum assembly flow problem.

$$\begin{aligned} \max \quad & x_t \\ \text{subject to} \quad & x \in F_C, \\ & x_i \leq u_i, \quad \forall i \in N_S. \end{aligned} \tag{5.1}$$

Assume that in an assembly network there is no directed cycle going through any C-node and all nodes in the network are reachable in the sense that for each S-node, there is a directed path from it to t; for each O-node, there is a directed path from an S-node to it, and a directed path from it to t; for each C-node, there is a directed path from it to t; and for each input node of a C-node, there is a directed path from an S-node to it. In this case, problem (5.1) has nontrivial solutions. Furthermore, there exists a partial order among the C-nodes. We may draw such an assembly network like Fig. 8 with t at the bottom and part suppliers on the top.

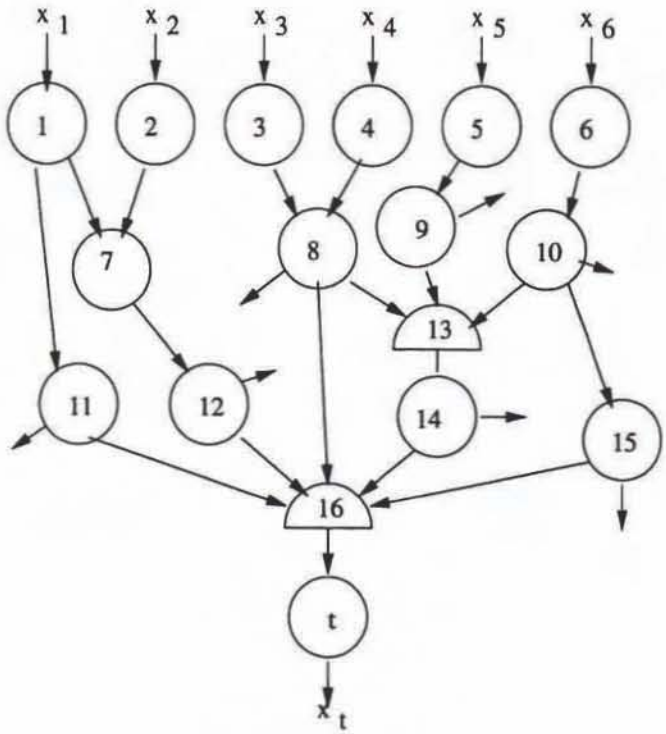

FIGURE 8 An assembly network.

Like in the distribution network, we can modify the objective function to consider the following minimization problem:

F. Minimum assembly cost problem.

$$\begin{aligned} \min \quad & \sum_{i \in N_S} c_i x_i + \sum_{(i,j) \in A} c_{ij} x_{ij} \\ \text{subject to} \quad & x \in F_C, \\ & x_i \leq u_i, \quad \forall i \in N_S, \\ & x_t \geq d_t, \end{aligned} \tag{5.2}$$

where c_i is the unit cost of the ith raw material, c_{ij} is the unit transportation cost on the arc (i, j) and d_t is the minimum demand for the final product.

Again, we are interested in knowing if the classical theory of the maximum flow value and minimum cost flow problems can be generalized here, and if there exist strongly polynomial-time algorithms in this setting.

To answer part of the questions raised in this and previous sections, we focus on the minimum distribution cost problem (4.4) to outline a network simplex method as a starting point for future research.

6 BASIC FEASIBLE GRAPH

For a minimum distribution cost problem (4.4) defined on a general network $G = (N, A)$, let x be a basic feasible solution. The subgraph of G corresponding to x is called a *basic feasible graph* of (4.4). Since this subgraph plays a key role in designing a network simplex method, in this section, we focus on studying its network structure.

First we show that a basic feasible subgraph of (4.4) is connected, if the underlying network G is connected. For easy analysis, we include an arc at the source node (for x_s) in the arc set A (for solution vector x) and assume $u_s = \infty$ and $u_{ij} = \infty$ for all $(i, j) \in A$. Let M be the corresponding node–arc incidence matrix. Because of (4.1), the flow balance Eq. (2.2) not only holds at every O-node, but also at every D-node. By adding a slack variable t_j at each T-node j, we have

$$\sum_{i \in E(j)} x_{ij} - t_j = d_j \quad \text{and} \quad t_j \geq 0 \quad \text{for all } j \in N_T.$$

Consequently, (4.4) becomes

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Mx - Qt = d \\ & Dx = 0 \\ & x, t \geq 0, \end{aligned} \tag{6.1}$$

where x is a flow vector (including x_s), c a cost vector, M the node–arc incidence matrix, and t a slack vector for T-nodes. For the demand vector d, we have $d_i = 0$ for all $i \notin N_T$. We assume that $c_s > 0$ and $c_{ij} \geq 0$ for all (i, j). Moreover, $Q = \begin{pmatrix} 0 \\ I \end{pmatrix}$ with I corresponding to T-nodes and 0 corresponding to non-T nodes, and $Dx = 0$ corresponding to (2.6) of D-nodes with slight modification.

Note that for each $i \in N_D$, because of (4.1), there is a redundant equation in (2.2) and (2.6) ($\forall j \in L(i)$). Hence we omit one equation of (2.6) for each $i \in N_D$ from $Dx = 0$ to avoid the degeneracy of D. In this way we know the matrix

$$\bar{M} = \begin{pmatrix} M & -Q \\ D & 0 \end{pmatrix}$$

has full row rank.

Let the number of nodes be n, the number of arcs (including the general supply arc) be m, the number of T-nodes be p, and the number of output arcs of D-nodes minus the number of D-nodes be q. Then the dimensions of M, Q, D and $\bar{M}$ are $n \times m$, $n \times p$, $q \times m$ and $(n + q) \times (m + p)$, respectively.

We may partition x into two parts:

$$x = \begin{pmatrix} x_N \\ x_D \end{pmatrix}$$

where x_D corresponds to arcs connected with D-nodes, x_N corresponds to other arcs. Correspondingly, we may partition $M = (M_N, M_D)$ and $D = (D_N, D_D)$. Since $Dx = 0$ corresponds to (2.6), we know $D_N = 0$. Therefore, problem (6.1) becomes

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Mx - Qt = d \\ & D_D x_D = 0 \\ & x, t \geq 0. \end{aligned} \tag{6.2}$$

The dual problem of (6.1) becomes:

$$\begin{aligned} \max \quad & d^T y = \sum_{j \in N_T} d_j y_j \\ \text{subject to} \quad & M^T y + D^T w \le c, \\ & y_T \ge 0, \end{aligned} \tag{6.3}$$

where y is the dual vector corresponding to the constraint $Mx - Qt = d$ in (6.1), u is the dual vector corresponding to the constraint $Dx = 0$ (same as $D_D x_D = 0$) in (6.1), and y_T are dual variables corresponding to the T-nodes.

As an illustration, in Fig. 7, we have

$$x_N = (x_s, x_{s,1}, x_{s,5}, x_{s,2}, x_{s,6}, x_{1,5}, x_{2,5}, x_{2,12}, x_{2,6}, x_{5,9}, x_{5,10}, x_{6,11}, x_{6,12}, x_{6,13}),$$
$$x_D = (x_{s,3}, x_{3,6}, x_{3,7}, x_{1,4}, x_{4,9}, x_{4,10}, x_{7,6}, x_{7,13}, x_{7,14}, x_{5,8}, x_{8,10}, x_{8,11}\ x_{8,12}),$$

$$M_N = \begin{pmatrix} 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$M_D = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$Q = \begin{pmatrix} 0 \\ I \end{pmatrix}, \quad D_N = 0,$$

$$D_D = \begin{pmatrix} k_{3,6} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{4,9} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{7,6} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & k_{7,13} & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{8,10} & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{8,11} & 0 & -1 & 0 \end{pmatrix},$$

where $n = 15$, $m = 27$, $p = 6$, $q = 6$, the first row of M corresponds to the node s, the other rows correspond to nodes 1 to 14.

Let x be a basic feasible solution of (6.1). We say a node is *idle*, if no positive flow goes through it. Otherwise, it is *active*. Some properties relating to a basic feasible solution are described in the following result:

THEOREM 6.1 (Network Structure of Basic Feasible Graphs) *Let x be a basic feasible solution of* (6.1) *and G_B be the basic feasible graph corresponding to x. Then,*

(i) *the node s and all T-nodes are active;*
(ii) *G_B spans all nodes, i.e., G_B is connected;*
(iii) *any cycle of G_B includes at least one D-node;*
(iv) *for a D-node, either all the arcs connected with it are basic arcs, or all the arcs connected with it except one arc are basic arcs. In the latter case, this D-node is idle;*
(v) *G_B is the union of a spanning tree of G, and $q - p_B$ additional arcs. The spanning tree has a root at s and p_B exiting arcs at T-nodes (corresponding to those basic slack variables).*

Proof (i) Since $d_j > 0$ for all $j \in N_T$, by (2.4), all T-nodes are active. In addition, since the positive flow must come from s, the node s is also active. (ii) Because the positive flow comes from s, all active nodes are connected with s in G_B. Now, assume that some idle nodes are not connected with s in G_B. By rearranging the rows and columns, the basic matrix corresponding to x can be written as

$$B = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix},$$

where B_1 corresponds to the nodes and basic arcs connected with s while B_2 corresponds to those not connected with s. From (i), we know that B_2 does not include the rows corresponding to node s and any demand nodes. In this way, some rows of B_2 are totally in M such that the sum of such rows is zero. This contradicts the nonsingularity of B. Hence B_2 must be empty and all nodes in G_B are connected with s. (iii) Again, let B be the basic matrix corresponding to x. By (6.2), the common part of B and M_N must have full column rank. According to the theory of network simplex method [1,3,7], any subgraph of G_B that does not include any D-node, contains no cycles. (iv) From the structure of M, it can be seen that a nonsingular $(n+q) \times (n+q)$ submatrix needs to contain either all the arcs connected with a D-node, or all the arcs, except one arc, connected with that D-node. In the latter case, an arc connected with that D-node is nonbasic and the flow on that arc is zero. By (2.4), we further know that the flows on all arcs connected with that D-node are zero. Consequently, this D-node is idle. (v) follows from (ii) and the dimension of the basic matrix. ■

As an example, a basic feasible graph is displayed in Fig. 9.

One interesting observation is that if we delete node s and arc x_s from G_B in Theorem 6.1, then we can partition the remaining graph into several connected subgraphs of G_B. We call

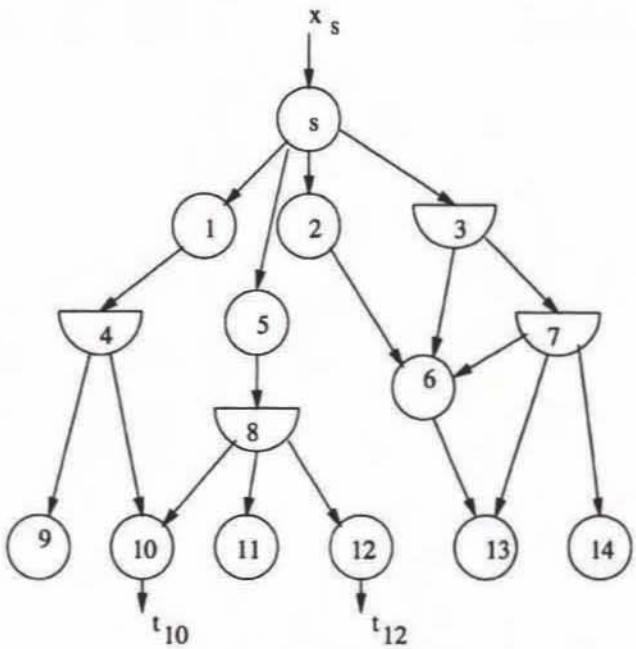

FIGURE 9 A basic feasible graph.

each of them a *branch* of G_B. For example, the basic feasible graph G_B in Fig. 9 has two branches. Adding the node s and arc x_s to a branch of G_B forms an *expanded branch.*

THEOREM 6.2 (Branches of a Basic Feasible Graph) *Let x be a basic feasible solution of* (6.1), *B the corresponding basic matrix, and G_B the corresponding basic feasible graph. Then,*

(i) *after rearranging rows and columns, B can be written in the form*

$$B = \begin{pmatrix} 1 & e_1^T & e_2^T & \cdots & e_l^T \\ 0 & B_1 & 0 & \cdots & 0 \\ 0 & 0 & B_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & B_l \end{pmatrix},$$

where the first column of B corresponds to x_s, the first row of B corresponds to node s, and square matrix B_i corresponds to the ith branch of G_B;

(ii) *for each i, $i = 1, \ldots, l$, B_i and*

$$\bar{B}_i = \begin{pmatrix} 1 & e_i^T \\ 0 & B_i \end{pmatrix}$$

are nonsingular.

(iii) *the ith expanded branch of G_B is the union of a tree rooted at s, and $q_i - p_i$ additional arcs, where p_i is the number of basic slack variables associated with the branch and q_i is the number of output arcs of D-nodes minus the number of D-nodes in the branch.*

Proof (i) follows from the definition of branches. (ii) follows from the nonsingularity of B. (iii) follows from the dimension of B_i for each i. ■

In Fig. 9, if we take the left branch as branch 1 and the right branch as branch 2, then we have $p_1 = 2$, $p_2 = 0$, and $q_1 = q_2 = 3$. Also note that we may delete 1 $(=3-2)$ arc from the *expanded branch* 1 and 3 $(=3-0)$ arcs from the *expanded branch* 2 to form trees.

7 PRIMAL BASIC FEASIBLE SOLUTION AND DUAL BASIC SOLUTION

Using the dual form (6.3) and Theorem 6.1, we can derive the optimality conditions for problem (6.2). Let B be an optimal basis. For an arc, $(i,j) \in B$ means the variable x_{ij} is a basic variable. Similarly, for a node $i \in N_T$, $i \in B$ means t_i is a basic variable. Remember that for each D-node $i \in N_D$, we have omitted one equation (2.4) from $D_x = 0$. Denote the arcs in $L(i)$, except the arc corresponding to the omitted equation, as $\bar{L}(i)$. Moreover, let

$$L_D = \bigcup_{i \in N_D} \bar{L}(i),$$

$$E_D = \bigcup_{i \in N_D} E(i),$$

$$A_O = A \backslash (L_D \cup E_D \cup \{s\}).$$

The optimality conditions consist of three parts:
(Primal) On the primal side, we need

$$x_{ij} = 0 \quad \forall (i,j) \notin B; \tag{7.1}$$

$$t_i = 0 \quad \forall i \in N_T,\ i \notin B; \tag{7.2}$$

$$x_{ij} = k_{ij} x_{i^* i} \quad \forall j \in L(i),\ i \in N_D, \tag{7.3}$$

where $E(i) = \{i^*\}$;

$$\sum_{j \in E(i),(j,i) \in B} x_{ji} = \sum_{j \in L(i),(i,j) \in B} x_{ij} \quad \forall i \in N_O; \tag{7.4}$$

$$\sum_{j \in E(i),(j,i) \in B} x_{ji} = d_i \quad \forall i \in N_T,\ i \notin B; \tag{7.5}$$

$$x_s = \sum_{j \in L(s),(s,j) \in B} x_{sj}; \tag{7.6}$$

$$t_i = \sum_{j \in E(i),(j,i) \in B} x_{ji} - d_i \quad \forall i \in N_T,\ i \in B. \tag{7.7}$$

(Dual Basic) On the dual side for basic variables, we need

$$y_s = c_s; \tag{7.8}$$

$$y_i = 0 \quad \forall i \in N_T \cap B; \tag{7.9}$$

$$y_i - y_j = c_{ij} \quad \forall (i,j) \in A_O \cap B; \tag{7.10}$$

$$y_i - y_j + \sum_{k \in \bar{L}(j)} k_{jk} u_{jk} = c_{ij} \quad \forall (i,j) \in (E_D \backslash L_D) \cap B; \tag{7.11}$$

$$y_i - y_j - u_{ij} = c_{ij} \quad \forall (i,j) \in (L_D \backslash E_D) \cap B; \tag{7.12}$$

$$y_i - y_j + \sum_{k \in \bar{L}(j)} k_{jk} u_{jk} - u_{ij} = c_{ij} \quad \forall (i,j) \in L_D \cap E_D \cap B. \tag{7.13}$$

(Dual Nonbasic) On the dual side for nonbasic variables, we need

$$y_i \geq 0 \quad \forall i \in N_T \backslash B; \tag{7.14}$$

$$y_i - y_j \leq c_{ij} \quad \forall (i,j) \in A_O \backslash B; \tag{7.15}$$

$$y_i - y_j + \sum_{k \in \bar{L}(j)} k_{jk} u_{jk} \leq c_{ij} \quad \forall (i,j) \in (E_D \backslash L_D) \backslash B; \tag{7.16}$$

$$y_i - y_j - u_{ij} \leq c_{ij} \quad \forall (i,j) \in (L_D \backslash E_D) \backslash B; \tag{7.17}$$

$$y_i - y_j + \sum_{k \in \bar{L}(j)} k_{jk} u_{jk} - u_{ij} \leq c_{ij} \quad \forall (i,j) \in (L_D \cap E_D) \backslash B. \tag{7.18}$$

With the primal optimality conditions (7.1–7.7), we may calculate the primal basic feasible solution x and t easily. To explain the procedure, we say a T-node is a *leaf* of G_B, if it is connected with only one arc in G_B. We also say a D-node is a *leaf* of G_B, if it is connected with a nonbasic arc. For example, in Fig. 9, nodes 9, 11 and 14 are leaves.

If a leaf of G_B is a T-node, then we may use (7.5) to calculate the flow value on the unique basic arc connected with this node. If a leaf is a D-node, Theorem 6.1 tells us that the flow values on all the basic arcs connected with this node are zero. By identifying these leaves and calculating related flow values, for the uncalculated part of G_B, we say an O-node or a T-node is a leaf if only one uncalculated basic arc is connected with it. Similarly, we say a D-node is a leaf if there is a calculated basic arc connected with it.

The following computational procedure tells us how to compute a primal basic feasible solution x and t.

ALGORITHM 7.1 (Calculate Primal Basic Feasible Solution) *Let G_B be a basic feasible graph corresponding to a basic feasible solution x.*

Step 1 *Identify the leaves of T-nodes and D-nodes of G_B. Use* (7.5) *to calculate the flow values of basic arcs connected with T-node leaves and set the flow values of basic arcs connected with D-node leaves to be zero.*

Step 2 *Identify additional leaves in the uncalculated part of G_B. Use* (7.3), (7.4) *and* (7.5) *to calculate the flow values of basic arcs connected with D-node, O-node and T-node leaves, respectively. Repeat this step until all basic feasible variables x_{ij} have been calculated.*

Step 3 *Use* (7.6) *to calculate x_s and* (7.7) *to calculate the values of basic slack variables.*

The following is an example to illustrate the Algorithm 7.1. In Fig. 9, nodes 9, 11 and 14 are leaves and we may have

$$x_{4,9} = d_9, \quad x_{8,11} = d_{11}, \quad x_{7,14} = d_{14}.$$

By (7.3), we have

$$x_{1,4} = \frac{x_{4,9}}{k_{4,9}}, \quad x_{5,8} = \frac{x_{8,11}}{k_{8,11}}, \quad x_{3,7} = \frac{x_{7,14}}{k_{7,14}}, \quad x_{s,3} = \frac{x_{3,7}}{k_{3,7}},$$

and

$$x_{4,10} = k_{4,10} x_{1,4}, \quad x_{8,10} = k_{8,10} x_{5,8}, \quad x_{8,12} = k_{8,12} x_{5,8},$$
$$x_{7,13} = k_{7,13} x_{3,7}, \quad x_{7,6} = k_{7,6} x_{3,7}, \quad x_{3,6} = k_{3,6} x_{s,3}.$$

From (7.5),

$$x_{6,13} = d_{13} - x_{7,13}.$$

From (7.4),

$$x_{s,1} = x_{1,4}, \quad x_{s,5} = x_{5,8}, \quad x_{2,6} = x_{6,13} - x_{3,6} - x_{7,6}, \quad x_{s,2} = x_{2,6}.$$

From (7.6),

$$x_s = x_{s,1} + x_{s,5} + x_{s,2} + x_{s,3}.$$

From (7.7),

$$t_{10} = x_{4,10} + x_{8,10} - d_{10}, \quad t_{12} = x_{8,12} - d_{12}.$$

To make sure that Algorithm 7.1 is a well-defined computational procedure, we introduce a concept called *computable path* in a recursive manner. Note that this concept is used only for analysis, not implementation. In a basic feasible graph G_B, a path is called a *computable path* if the following conditions are met:

(i) it ends at a leaf of G_B;
(ii) a node or an arc appears in such a path only once;
(iii) if it passes through an O-node or a T-node, all other basic arcs connected with such a node start some computable paths.

Note that in Fig. 9, arcs (6, 7), (7, 14) form a computable path, arcs (6, 3), (3, 7), (7, 14) form another computable path, hence arcs (s, 2), (2, 6), (6, 13), (13, 7), (7, 14) form a computable path.Also note that a computable path from arc (s, 2) may not be unique.

THEOREM 7.1 *Let G_B be a basic feasible graph corresponding to a basic feasible solution x. Then Algorithm* 7.1 *is a well-defined computational procedure. In particular,*

(i) *for a non-leaf D-node, there exists exact one basic arc that connects with it and starts some computable paths;*
(ii) *for each non-leaf non-D node, all but except one arcs connected with it start some computable paths.*

Proof Let B be the basic matrix associated with G_B. Assume that there exist two basic arcs connected with a non-leaf D-node and some computable paths start from each of these two arcs. Take one computable path for each case. From the definition of the computable path, we know it is possible to construct another computable path from each basic arc that is not in these two paths, but connected with these two paths at some non-D nodes.We repeat this process until there are no such basic arcs. Deleting the overlapping part, we see that the rows of B corresponding to nodes in these computable paths are linearly dependent, as we may eliminate all of them one by one. This causes a contradiction and proves (i). Similarly, we can prove that for each non-leaf non-D node, all but except one arcs connected with it start some computable paths. These facts assure us that Algorithm 7.1 will not generate two different values for one primal basic variable, no matter which order it takes.

Note that throughout the computation procedure of Algorithm 7.1, the (node–arc) difference between the number of uncalculated basic arcs and the number of nodes associated with

them is held to be $\bar{q} - \bar{p}_B$, where $\bar{q}$ is the number of uncalculated output arcs of the remaining D-nodes minus the number of the remaining D-nodes and $\bar{p}_B$ is the number of uncalculated basic slack variables. This is true in the beginning of the procedure because of Theorem 6.1. When we calculate a basic arc connected with a non-D node, we disassociate that node immediately. Therefore, the (node–arc) difference in the un-computed part of G_B remains unchanged. On the other hand, when we calculate arcs associated with a D-node, exact one arc associated with it has been calculated before. This again does not change the (node–arc)difference.

Because of this (node–arc) difference, throughout the computational procedure, as long as there exists a basic arc x_{ij} to be computed, there is at least one leaf. Hence, the computational procedure in Algorithm 7.1 is well-defined. ■

After computing a primal basic feasible solution, we may use the optimality conditions (7.8)–(7.13) to calculate a dual basic solution of y and u. What follows is a computational procedure for this purpose.

ALGORITHM 7.2 (Calculate Dual Basic Solution) *Let G_B be a basic feasible graph corresponding to a given basic feasible solution x.*

Step 1 *Calculate y_s and y_i for $i \in N_T \cap B$ according to* (7.8) *and* (7.9).

Step 2 *Calculate the remaining dual basic variables by using* (7.10)–(7.13). *Possibly some small systems of linear equations need to be solved.*

For example, in Fig. 9 (7.8) and (7.9) result in

$$y_s = c_s, \quad y_{10} = 0, \quad y_{12} = 0.$$

By (7.10), we have

$$y_1 = y_s - c_{s,1}, \; y_5 = y_s - c_{s,5}, \; y_2 = y_s - c_{s,2}, \; y_6 = y_2 - c_{2,6}, \; y_{13} = y_6 - c_{6,13},$$

and

$$y_4 = y_{10} + c_{4,10}, \quad y_8 = y_{12} + c_{8,12}.$$

From (7.11) and (7.12),

$$u_{4,9} = \frac{y_4 - y_1 + c_{1,4}}{k_{4,9}}, \quad y_9 = y_4 - u_{4,9} - c_{4,9}.$$

Similarly,

$$u_{8,10} = y_8 - y_{10} - c_{8,10}, \quad u_{8,11} = \frac{y_8 - y_5 + c_{5,8} - k_{8,10}u_{8,10}}{k_{4,9}},$$

$$y_{11} = y_8 - u_{8,11} - c_{8,11}.$$

Moreover, we have a system of linear equations:

$$\begin{aligned} -y_3 + k_{3,6}u_{3,6} &= c_{s,3} - y_s, \\ y_3 - u_{3,6} &= c_{3,6} + y_6. \end{aligned} \tag{7.19}$$

Since $k_{3,6} < 1$, this system is nonsingular. We can solve it to find y_3 and $u_{3,6}$. By (7.11) and (7.12), we have the following system of linear equations:

$$\begin{aligned} -y_7 + k_{7,6}u_{7,6} + k_{7,13}u_{7,13} &= c_{3,7} - y_3, \\ y_7 - u_{7,6} &= c_{7,6} + y_6, \\ y_7 - u_{7,13} &= c_{7,13} + y_{13}. \end{aligned} \tag{7.20}$$

Note that this system is nonsingular, so we can solve it for y_7, $u_{7,6}$ and $u_{7,13}$. Finally, using (7.10), we have

$$y_{14} = y_7 - c_{7,14}.$$

It is not by chance that the coefficient matrices of (7.19) and (7.20) are both nonsingular. They are guaranteed by the following theorem.

THEOREM 7.2 *The coefficient matrices of systems of linear equations encountered in Algorithm* 7.2 *are always nonsingular.*

Proof From the theory of linear programming, we know the solution process involved in Algorithm 7.2 is to solve a system of linear equations, whose coefficient matrix is B^T. No matter how we break the procedure into solving smaller systems of linear equations, as long as B is nonsingular, the coefficient matrices encountered in the procedure are always nonsingular. ■

8 A NETWORK SIMPLEX METHOD

Using Algorithms 7.1, 7.2 and the optimality conditions (7.1)–(7.18), we may outline a network simplex method for solving (6.1).

ALGORITHM 8.1 (Network Simplex Method) *Let G_B be a basic feasible graph corresponding to a given basic feasible solution x and t.*

Step 1 *Calculate the dual basic solution y and u by using Algorithm* 7.2.

Step 2 *Check dual feasibility conditions* (7.14)–(7.18). *If they are satisfied, then stop. (We have reached optimality.) Otherwise, pick a nonbasic arc (i, j) or a non-basic slack variable t_i that violates the dual feasibility conditions to enter the basis as a new basic variable.*

Step 3 *Do pivoting by adding the new basic variable to current basis, find a basic variable to leave the basis, and update the primal basic feasible solution x and t. Return to Step* 1 *for the next iteration.*

Note that both Step 1 and Step 2 of the proposed network simplex method are quite straightforward. But Step 3 requires efficient ways to identify a basic variable leaving the basis and to update x and t.

An observation can be made here. The nonbasic arc chosen in Step 2 (to enter the basis) connects two nodes. Either they are in the same branch of G_B or they sit in two different branches of G_B. Consequently, the leaving basic arc will be found either in the corresponding

branch or two branches. This means that we do not have to worry about the basic arcs in other branches. Hence a lot of work can be saved in solving large-scale problems.

In each pivot, a subgraph of G_B together the nonbasic arc chosen in Step 2, on which flows need to be updated, is called a *pivoting graph*. The pivoting graph consists of only part of one or two branches of G_B as described above. Recall that in the ordinary network flow problem, the pivoting graph is a cycle. In such cycle, arcs have the same direction as the nonbasic arc entering the basis are called *forward arcs*. Other arcs in the cycle are called *backward arcs*. The flow on each forward arc will be increased by a nonnegative amount w, while the flow on each backward arc will be decreased by w. A backward arc whose flow will reach zero first is then identified to be the leaving basic variable of the current basis. If the cycle does not contain any backward arc, the network flow problem has an unbounded value.

For problem (6.1), the pivoting graph may be more complicated than a cycle and the updated amount on each arc in the pivoting graph may be a fraction or multiple of w. One simple way to identify a leaving basic arc is to take w as a parameter and apply Algorithm 7.1 to the one or two branches that the entering nonbasic arc connects, plus the entering nonbasic arc. If flows on some basic arcs are increased by a fraction or multiple of w, these arcs are forward arcs. Similarly, if flows on some basic arcs are decreased by a fraction or multiple of w, these arcs are backward arcs. A backward arc on which flow will reach zero first is identified to be the basic arc leaving the basis. Since problem (6.1) is bounded, there is at least one backward arc.

The above process can be further improved if we know how to identify the pivoting graph. Let us discuss this issue case by case.

In case that the nonbasic arc chosen in Step 2 (entering the basis) forms a cycle with some basic arcs, and such a cycle does not include any D-node, according to Theorem 6.1, a basic arc in this cycle must leave the basis to break the cycle. Then this cycle is the pivoting graph, and we may follow the normal network pivoting procedure along this cycle to find a basic arc that will leave the basis. Consequently, we can update x. For example, in Figs. 7 and 9, if a nonbasic arc (1, 5) is chosen to enter the basis, then the basic arc (s, 5) will leave the basis. Let $w = x_{s,5}$. We may drop the arc (s, 5) from the basis, and increase the flows in arcs (s, 1) and (1, 5) by w. If the arc (2, 5) or (s, 6) is chosen to enter the basis, similar analysis follows.

Otherwise the nonbasic arc chosen in Step 2 (entering the basis) may form one or several cycles with some basic arcs, and each of these cycles contains at least one D-node.We call a cycle *dead* if it contains two output arcs of the same D-node. For example, consider the basic solution corresponding to Fig. 9. If the nonbasic arc (5, 9) is chosen to enter the basis, then it forms a *dead cycle* with basic arcs (4, 9), (4, 10), (8, 10), (5, 8) as shown in Fig. 10 (broken line for the entering arc), because arcs (4, 9) and (4, 10) come from the same D-node 4. Note that dead cycles are useless in pivoting flows, and hence can be ignored in identifying basic variables leaving the current basis. Now, for all those non-dead cycles, we put the entering arc together with all the basic arcs and related nodes involved. Also, for all D-nodes in this subgraph, we add all the basic arcs and related nodes that lead the D-nodes to final demands. For example, for Fig. 9, if the nonbasic arc (6, 11) is chosen to enter the basis, the resulted subgraph is shown in Fig. 11. Arcs (8, 10), t_{10}, (8, 12), t_{12} and (7, 14) do not form cycles with (6, 11), but they lead D-nodes 7 and 8 to demands. Thus, they are included in the resulted subgraph. Also note that nodes 10, 12 and 14 are related nodes to be included in the resulted subgraph.

In the resulted graph, if some arcs are "leaves", we may apply the method described in Algorithm 7.1 to calculate the flows in these and related arcs from (7.5), (7.3) and (7.4). By deleting all such arcs and related nodes, some cycles will break. Such cycles are called *false cycles*. By eliminating all false cycles and related arcs and nodes from

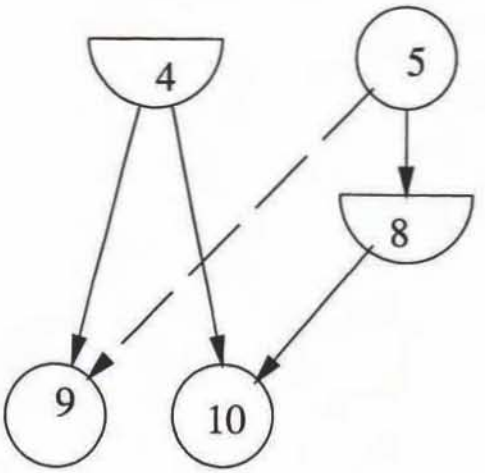

FIGURE 10 A dead cycle.

the resulted graph, we get the pivoting graph. For example, consider the resulted graph of Fig. 11, node 14 is a leaf. From it we can calculate $x_{7,14}$, $x_{3,7}$, $x_{7,6}$, $x_{7,13}$, $x_{s,3}$ and $x_{3,6}$. Delete all these arcs and related nodes 14, 7 and 3. Some false cycles are broken. By deleting the related arc (6, 13) and node 13 in these false cycles, we have a pivoting graph of Fig. 12.

In the pivoting graph, assume the flow on the entering arc is increased by w. Then we may apply the method described in Algorithm 7.1 to calculate the changes on the basic arcs of the

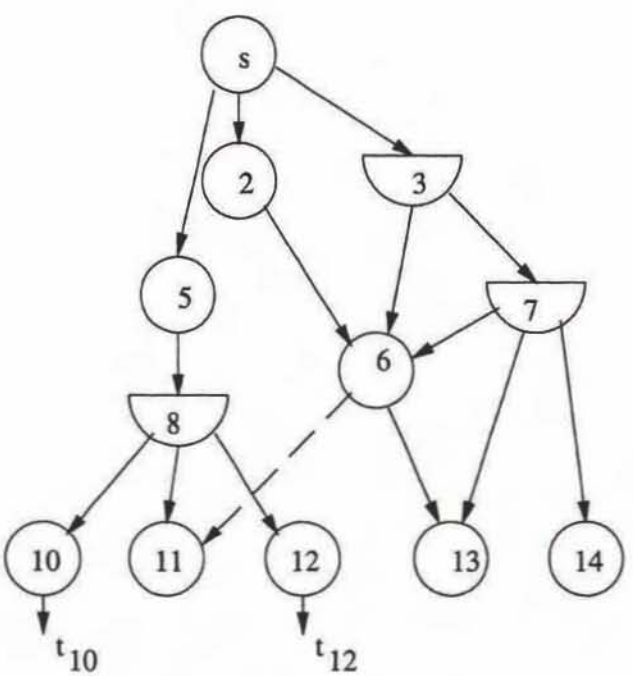

FIGURE 11 Resulted subgraph.

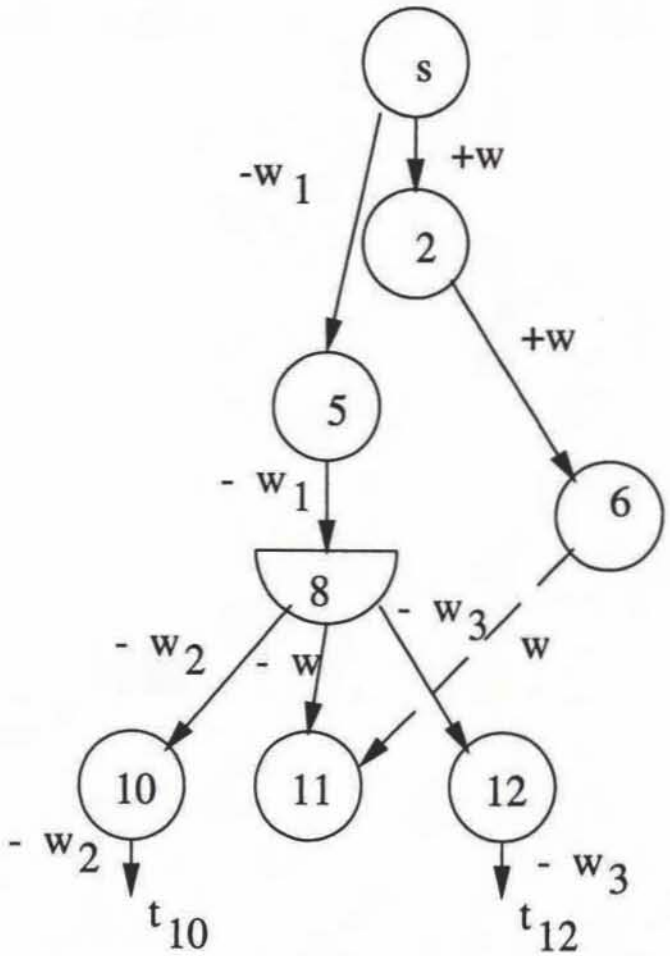

FIGURE 12 The pivoting graph.

pivoting graph, with w being a parameter. An example is shown in Fig. 12, where

$$w_1 = \frac{w}{k_{8,11}}, \quad w_2 = w_1 k_{8,10}, \quad w_3 = w_1 k_{8,12}.$$

Knowing the original flow values, we can determine which basic arc to leave the basis. In Fig. 12, either the slack variable t_{10} or the slack variable t_{12} will be the basic arc to leave the basis, depending upon $t_{10} - w_2$ or $t_{12} - w_3$ becomes zero first when w increases. $x_{s,5}$, $x_{5,8}$, $x_{8,10}$, $x_{8,11}$ and $x_{8,12}$ will not leave the basis in this pivot because t_{12} becomes zero before they do, by noting the facts that

$$t_{12} = x_{8,12} - d_{12} \leq x_{8,12},$$

and $x_{s,5}$, $x_{5,8}$, $x_{8,10}$ and $x_{8,11}$ become zero at the same time as $x_{8,12}$ does.

Notice that unless the entering arc corresponds to a nonbasic t variable, after deleting the dead and false cycles, there remains at least one more cycle containing this entering arc, for

the nondegenerate case. For example, in Fig. 9, assume that arc (1, 10) is chosen to enter the basis and the flow will be increased by w. Then we may apply the method described in Algorithm 7.1 to calculate the changes on the basic arcs of the pivoting graph, with w being a parameter. It is not difficult to check that there is no leaving arc. Moreover, in order to minimize the sum of all flows in all the arcs, we see $w = 0$. This implies that arc (1, 10) cannot be chosen in Step 2 at the first place. As to the degenerate case, the situation becomes more complicated. We may be able to find a suitable basic arc in the idle part of G_B as the leaving arc, but no flow changes in pivoting.

The final case is that the entering arc is a nonbasic slack variable. In this case, we need to include in the pivoting graph all the basic arcs and related nodes, including arc x_s, which come from the general supply node s to this demand node, or which lead to some other final demand nodes from some D-nodes in the above supplying "paths". Figure 13 displays the pivoting graph when t_{14} is chosen as the entering arc. In this example, the pivoting graph is the whole expanded branch of G_B.

In Fig. 13, we have

$$w_1 = \frac{w}{k_{7,14}}, \quad w_3 = w_1 k_{7,13}, \quad w_4 = w_1 k_{7,6},$$

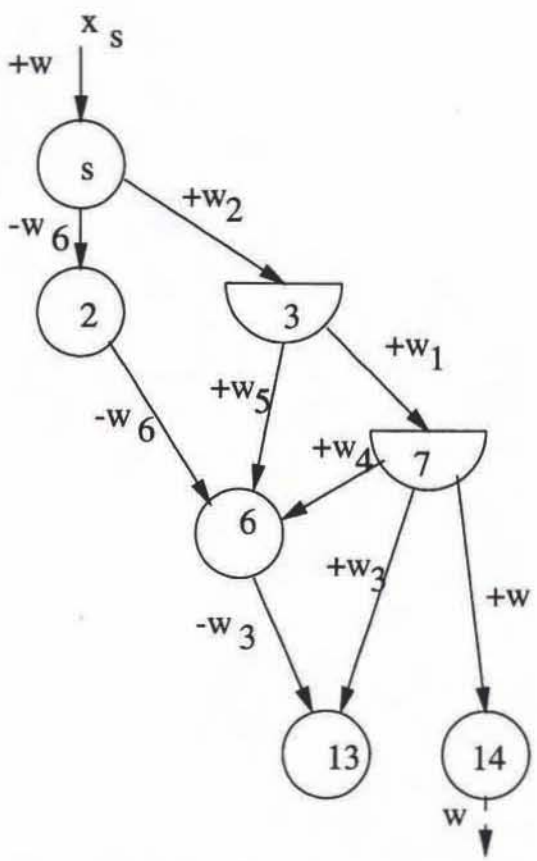

FIGURE 13 The entering arc is a slack variable.

and

$$w_2 = \frac{w_1}{k_{3,7}}, \quad w_5 = w_2 k_{3,6}, \quad w_6 = w_4 + w_5 + w_3.$$

Again, with the original flow values, we can determine which basic arc to leave the basis.

To obtain an initial basic feasible solution of (6.1), we may add artificial arcs from s to $\mathcal{T}$-nodes. The two-phase or big-M strategies then applies.

9 FINAL REMARKS

In this paper, we have presented a generalized network flow framework to model more complicated manufacturing scenarios. Two simplified versions are specifically proposed for further investigations on the following two fundamental issues:

(1) Can some classical duality and optimality results of the ordinary network flow problems be extended in the new setting?
(2) Can some strongly polynomial-time algorithms be developed for solving the related optimization problems in the new framework?

We have explored the duality and optimality conditions of the minimum distribution cost problem and outlined a network simplex method for solving it. But more research work is needed to give a complete answer of the above two questions.

Acknowledgments

The authors would like to thank Chung-yee Lee, Chung-lun Li and two referees for their comments.

References

[1] R.K. Ahuja, T.L. Magnanti and J.R. Orlin (1993). *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, New Jersey.
[2] R.G. Askin and C.R. Standridge (1993). *Modeling and Analysis of Manufacturing Systems.* John Wiley & Sons, New York.
[3] M.S. Bazaraa, J.J. Jarvis and H.D. Sherall (1990). *Linear Programming and Network Flow,* 2nd ed. John Wiley & Sons, New York.
[4] L.R. Ford and D.R. Fulkerson (1962). *Flows in Networks.* Princeton University Press, Princeton.
[5] E.L. Lawler (1976). *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart & Winston, New York.
[6] C. Leondes (Ed.) (2001). *Computer Integrated Manufacturing.* CRC Press, New York.
[7] K.G. Murty (1992). *Network Programming.* Prentice Hall, New Jersey.
[8] C.H. Papadimitriou and K. Steiglitz (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, New Jersey.