J Comb Optim (2008) 15: 315–341
DOI 10.1007/s10878-007-9080-6

An efficient generalized network-simplex-based algorithm for manufacturing network flows

Prahalad Venkateshan · Kamlesh Mathur · Ronald H. Ballou

Published online: 8 June 2007

Abstract Fang and Qi (Optim. Methods Softw. 18:143–165, 2003) introduced a new generalized network flow model called manufacturing network flow model for manufacturing process modeling. A key distinguishing feature of such models is the assembling of component raw-materials, *in a given proportion*, into an end-product. This assembling operation cannot be modeled using usual generalized networks (which allow gains and losses in flows), or using multi-commodity networks (which allow flows of multiple commodity types on a single arc). The authors developed a network-simplex-based algorithm to solve a minimum cost flow problem formulated on such a generalized network and indicated systems of linear equations that need to be solved during the course of the network-simplex-based solution procedure. In this paper, it is first shown how various steps of the network-simplex-based solution procedure can be performed efficiently using appropriate data structures. Further, it is also shown how the resulting system of linear equations can be solved directly on the generalized network.

Keywords Network flows · Manufacturing networks · Generalized network-simplex method

P. Venkateshan (✉)
AmTrust Bank, 1801 East 9th Street, Mail-Code OH99-0307, Cleveland, OH 44114, USA
e-mail: prahalad@gmail.com

K. Mathur · R.H. Ballou
Department of Operations, Weatherhead School of Management, Case Western Reserve University, 10900 Euclid Avenue, Cleveland, OH 44106, USA

K. Mathur
e-mail: Kamlesh.Mathur@case.edu

R.H. Ballou
e-mail: Ronald.Ballou@case.edu

1 Introduction

Network flows have been studied extensively by the operations research community (Ahuja et al. 1993; Bazaraa et al. 1990). A large number of real problems encountered in practice can be accurately formulated as equivalent problems on some underlying network. Network problems arise in applications in areas as diverse as finance (Golden et al. 1979), material production and distribution (Geoffrion and Graves 1974), and vehicle routing (Toth and Vigo 2002), to name a few. Network problems also allow for efficient solution procedures.

The network-simplex method is one such procedure that solves network problems by specializing the simplex algorithm for networks. Various steps of the simplex algorithm have *efficient* counter-parts in the network-simplex method. The generalized network-simplex method is used to solve network flow problems that allow for gains and losses in flows. Multi-commodity network flow problems, likewise, have an equivalent network-simplex-based solution procedure.

Recently, Fang and Qi (2003) introduced a new generalized network flow model called the manufacturing network flow model. Mo et al. (2005) expanded the study of manufacturing network flows by incorporating certain features of ordinary multi-commodity network flow models. A key distinguishing property of all such models is the assembling of component raw-materials, *in a given proportion*, to produce end-products. Such an operation cannot be modeled on usual generalized networks because generalized networks allow for gains and losses in the *total* flow that enters a particular node or travels along an arc, regardless of the *type* of each component of the total flow. Multi-commodity networks are used primarily in situations where multiple commodities share a common distribution network with some resource constraints on arcs and/or nodes. Neither of these two types of networks has the capability to model the manufacturing or assembling operation.

In their paper, Fang and Qi (2003) also outlined the steps of a network-simplex-based solution procedure for a minimum cost flow problem on manufacturing networks. Due to the fact that manufacturing network flows involve the assembly operation, which is fundamentally different from the types of operations encountered in other network flow models, it is not immediately clear whether or how various steps and updates involved in the network-simplex-based algorithm can be carried out with the same efficiency as the usual network-simplex method. Our main contribution in this paper is to show that it is indeed possible to carry out the steps of the network-simplex-based method and the updates involved, with efficiency. More specifically, the main results of this paper are the following:

1. We develop data structures that would aid in efficient implementation of the network-simplex-based solution procedure for manufacturing networks.
2. We show how certain linear systems of equations that arise during the course of the network-simplex-based procedure for manufacturing networks can be efficiently solved.

In the model developed by Fang and Qi (2003), even though the total incoming flow at a node along different arcs had to be in a certain given proportion, the total flow into a node was required to be equal to the total flow out of a node. Lu et al. (2006) study a manufacturing network model in which this assumption is relaxed.

That is, even though the incoming flow at a node along its different incoming arcs had to be in a given proportion, the total flow out of this node was not required to be equal to the total sum of the incoming flows at that node. The methodology developed in our paper is capable of handling this more general type of flow. To aid in discussion of our methodology, it will be useful to describe an application in which a manufacturing network flow model is necessary. In a variety of problem contexts, geographically dispersed customers demand units of an end product each one of which is composed of various components in a fixed and given proportion. The end product is produced (or assembled, as the case may be) at the site of geographically dispersed assemblers by combining the requisite amount of components. The assembler could represent a manufacturer, a distributor, or a retailer, or more generally any entity that assembles multiple components into a single product for onward distribution.

There exist geographically dispersed suppliers who specialize in the production and supply of each component. Multiple suppliers may supply the same component but a supplier is capable of supplying only a single type of component. Assemblers source components from suppliers. Capacity constraints (lower and upper bounds) may exist on how much of a component a supplier can supply, how much of an end product each assembler can assemble and on how much of a commodity (component or end product) can be shipped between a supplier and an assembler and between an assembler and a customer. Shipment of components or end products incurs a cost that varies linearly with the amount of flow. The problem is to find the flow of commodities that satisfies customer demand at least cost.

A pictorial example will help in better understanding of the problem. Figure 1 depicts an example in which each demand cluster has a certain demand of an end product, each unit of which is made up of a certain quantity of component 1 and a certain quantity of component 2. Supplying one unit of the end product of demand cluster 1 from assembler 1 requires the sourcing of appropriate amounts of components from the suppliers of that specific component. As a result of this requirement,

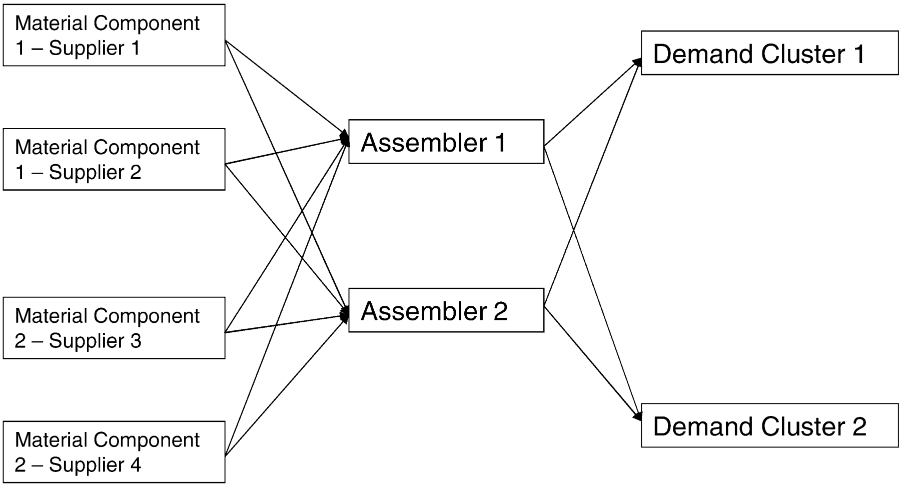

Material Component 1 – Supplier 1
Material Component 1 – Supplier 2
Material Component 2 – Supplier 3
Material Component 2 – Supplier 4
Assembler 1
Assembler 2
Demand Cluster 1
Demand Cluster 2

Fig. 1 Example problem

it is not possible to split the demand of the end product into individual and separate component demands.

The rest of the paper is organized as follows. In Sect. 2, we introduce the notation and the manufacturing network flow model used to solve the minimum cost distribution problem. In Sect. 3, certain graph-theoretical properties of the network flow model are derived. In Sect. 4, we describe the steps of the network-simplex-based algorithm. In Sect. 5 an example is studied which illustrates the proposed methodology.

2 Model

First we list the notation used throughout the paper.

i = index of a customer, $i = 1, \ldots, I$

d_i = non-negative demand of ith customer in units of end-product

j = index of an assembler, $j = 1, \ldots, J$

k = index of a supplier, $k = 1, \ldots, K$

p = index of a component type, $p = 1, \ldots, P$

a_p = positive amount of pth component needed to assemble one unit of end-product

$R(k)$ = index of component supplied by the kth supplier

b_{ji} = non-negative unit shipment cost of the end product from jth assembler to ith customer

d_{kj} = non-negative unit shipment cost of component from kth supplier to jth assembler

$[Q]$ = the set of integers from 1 through Q, defined only for positive integral Q

In addition to the above, let (L_j, U_j) and (l_k, u_k) stand for the non-negative lower and upper bounds (if any) on the amount shipped out of the jth assembler and kth supplier respectively. Non-negative lower and upper bounds on the amount shipped between specific assembler-customer and supplier-assembler pairs are likewise represented by (L_{ji}, U_{ji}) and (l_{kj}, u_{kj}) respectively.

We are now in a position to describe the manufacturing network model for the distribution problem. As illustrated in Fig. 2, in this network, there is a node, S_k, representing the kth supplier, a node A_j, representing the jth assembler and a node, C_i, representing the ith customer. The flow of component p originates from a node r_p. Directed arcs connect r_p to all suppliers supplying the pth component. The assembling operation performed by the jth assembler is represented by a set of nodes and arcs. A node of type A_jr_p is used to conceptually represent the entry point of the pth component at the facility of the jth assembler. All incoming flow at node A_jr_p flows to node B_jr_p. This node is used to represent the contribution of the pth component towards the assembly operation. All nodes of type B_jr_p are also "associated" with node AF_j. This association has a very special meaning in the context of this paper. First, there is no arc between a B_jr_p node and AF_j node. Secondly, feasibility of a solution requires that the total units of the end-product produced by the jth assembler be matched by the exact appropriate amount of each component sourced by that assembler. As will be shown subsequently the special association between the B_jr_p

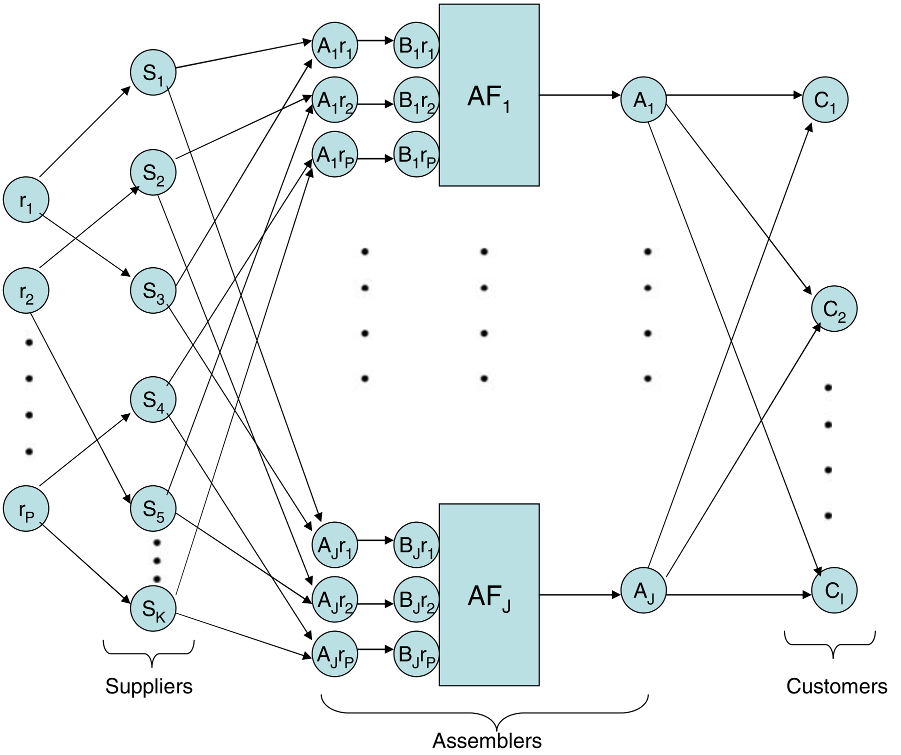


Fig. 2 Example of manufacturing network

nodes and the AF_j node will help achieve this requirement. Thirdly, node AF_j will be shown to be useful both for a mathematical and an intuitive understanding of the network-simplex-based solution procedure. Nodes $B_jr_p, \forall p \in [P]$ and node AF_j together represent the actual *assembly process* at the site of the jth assembler. These $P+1$ nodes are the *nodes associated with the* j*th assembly process*. Nodes AF_j lead into nodes A_j that are in turn connected to customer nodes C_i.

All nodes in this network satisfy the usual flow conservation property (flow in = flow out) except nodes of type B_jr_p and AF_j. More specifically, if $z_{A_jr_p,B_jr_p}$ stands for the flow on arc (A_jr_p, B_jr_p) and z_{AF_j,A_j} stands for the flow on arc (AF_j, A_j), the two are related by the following relationship:

$$z_{A_jr_p,B_jr_p} - a_p z_{AF_j,A_j} = 0, \quad \forall p \in [P],\ \forall j \in [J]. \tag{1}$$

Equation (1) represents the requirement that assembling 1 unit of the end-product requires a_p units of the pth component sourced to an assembler. It is this very requirement that leads to the complication in the model and necessitates the development of a new network-simplex-based solution methodology for manufacturing network flows. If $P=1$, and $a_1=1$, the problem simplifies to the usual minimum cost network flow problem.

Now, the problem of finding the optimal flow on the manufacturing network can be cast in the following general form:

$$\textbf{[MCNF]}\ \text{Minimize}\ \mathbf{c^t z}$$

$$\begin{aligned} \text{s.t.}\quad & \mathbf{Az} = \mathbf{d}, && (2)\\ & \mathbf{Bz} = \mathbf{0}, && (3)\\ & \mathbf{LB} \le \mathbf{z} \le \mathbf{UB} && (4) \end{aligned}$$

in which, each column (variable) represents a unique arc of the manufacturing network. The vector of decision variables, **z**, is of size $K + KJ + JP + J + JI$, corresponding to the total number of arcs in Fig. 2. Vector **c** contains as components the distribution costs, b_{ji} and d_{kj}.

Constraints (2) represent the usual flow balance constraints (flow in = flow out) at all nodes in the network except those of type $B_j r_p$ and AF_j. As a result, matrix **A** is totally unimodular. The number of rows in matrix **A** (and vector **d**) is $K + JP + J + I$, which is equal to the number of nodes in Fig. 2 that satisfy the usual flow balance constraints. Except for components corresponding to customer demand d_i, $\forall i \in [I]$, all other entries of vector **d** are 0. Constraints (3) represent all constraints of type (1). Matrix **B** has JP rows corresponding to the flow balance constraints at all nodes of type $B_j r_p$. Constraints (4) impose lower (**LB**) and upper (**UB**) bounds on the flow.

3 Graph-theoretic properties of bases of [MCNF]

In this section, we will derive certain graph-theoretic properties of the bases of problem **[MCNF]**. These properties will be used in the development of a network-simplex-based algorithm.

For problem **[MCNF]**, let α be the total number of constraints in the constraint set (2) and β be the total number of constraints in the constraint set (3). By adding enough artificial variables, the constraint set can be made to be of full rank. Then, any basis is a square matrix with $(\alpha + \beta)$ rows.

The network-simplex-based method works directly on a graph underlying problem **[MCNF]** such as the one depicted in Fig. 2. Let $\mathcal{B}$ denote a basis of problem **[MCNF]**. The columns of $\mathcal{B}$ correspond to a certain set of arcs of the manufacturing network. These arcs together constitute a sub-graph of the manufacturing network. We now derive a few properties of this sub-graph.

In the graph corresponding to the columns of the constraint matrix of problem **[MCNF]**, the following types of cycles along which flow can be augmented exist:

Type 1: Cycle involving nodes $r_p, S_{k_1}, \ldots, S_{k_q}, A_{j_1} r_p, \ldots, A_{j_{q-1}} r_p$ where $R(k_1) = \cdots = R(k_q) = p$, $k_1 \neq \cdots \neq k_q$ and $j_1 \neq \cdots \neq j_{q-1}$.

Type 2: Cycle involving nodes $S_{k_1}, \ldots, S_{k_q}, A_{j_1} r_p, \ldots, A_{j_q} r_p$ where $R(k_1) = \cdots = R(k_q) = p$, $k_1 \neq \cdots \neq k_q$ and $j_1 \neq \cdots \neq j_q$.

Type 3: Cycle involving nodes $A_{j_1}, \ldots, A_{j_q}, C_{i_1}, \ldots, C_{i_q}$ where $j_1 \neq \cdots \neq j_q$ and $i_1 \neq \cdots \neq i_q$.

Type 4: Cycle involving AF_j nodes: For instance, $AF_{j_1}, A_{j_1}, C_{i_1}, \ldots, C_{i_{q-1}}, A_{j_q}, AF_{j_q}$, where $j_1 \neq \cdots \neq j_q$, $i_1 \neq \cdots \neq i_{q-1}$ and one of the following set of nodes for each $p \in [P]$:

(a) $A_{j_1}r_p, A_{j_q}r_p, A_{j_{l_1}}r_p, \dots, A_{j_{l_{q_p}}}r_p, B_{j_1}r_p, B_{j_q}r_p, B_{j_{l_1}}r_p, \dots, B_{j_{l_{q_p}}}r_p, r_p, S_{k_1}, \dots, S_{k_{q_p}}$, where $k_1 \neq \cdots \neq k_{q_p}$, $j_{l_1} \neq \cdots \neq j_{l_{q_p}} \neq j_1 \neq j_q$, $R(k_1) = \cdots = R(k_{q_p}) = p$ and $q_p = l_{q_p} + 2$.

(b) $A_{j_1}r_p, A_{j_q}r_p, A_{j_{l_1}}r_p, \dots, A_{j_{l_{q_p}}}r_p, B_{j_1}r_p, B_{j_q}r_p, B_{j_{l_1}}r_p, \dots, B_{j_{l_{q_p}}}r_p, S_{k_1}, \dots, S_{k_{q_p-1}}$, where $k_1 \neq \cdots \neq k_{q_p-1}$, $j_{l_1} \neq \cdots \neq j_{l_{q_p}} \neq j_1 \neq j_q$, $R(k_1) = \cdots = R(k_{q_p-1}) = p$ and $q_p = l_{q_p} + 1$.

Since a cycle of type 4 involves arcs corresponding to the end-product as well as the components, we refer to it as a *complex* cycle. We now show that the columns associated with arcs corresponding to each one of the type of cycles are linearly dependent.

Lemma 1 *Columns of the constraint matrix for problem* **[MCNF]** *associated with arcs corresponding to each one of the four types of cycles are linearly dependent.*

Proof Proof of this lemma for cycles of types 1 through 3 is similar to the proof for the usual network-simplex method. Such proofs can be found in Kennington and Helgason (1980), for instance, and are omitted here. The proof for a cycle of type 4 is unique to manufacturing networks. We prove the result for the complex cycle used as an example in the definition of the type 4 cycle. The proof for other complex cycles is similar. Let us define $\mathbf{e}_{f,t}$ to be the column of the constraint set of problem **[MCNF]** corresponding to the directed arc between nodes f and t. If f is not an AF_j type node, this column is a vector of size $\alpha + \beta$ with all entries 0 except an entry of -1 corresponding to the component associated with node f and an entry of $+1$ corresponding to the component associated with node t. If the arc (f, t) corresponds to the arc (AF_j, A_j), $\mathbf{e}_{f,t}$ has all entries 0, except an entry of 1 corresponding to the component associated with node A_j, and an entry of $-a_p$ corresponding to the component associated with node B_jr_p, $\forall p \in [P]$.

We first note that each cycle of type 4, consists, in effect, of P sub-cycles, one for each component type. Each one of these sub-cycles has the same nodes amongst the AF_j, A_j and C_i nodes as part of the sub-cycle. Let these nodes be $AF_{j_1}, \dots, AF_{j_q}$, $A_{j_1}, \dots, A_{j_q}$, $C_{i_1}, \dots, C_{i_{q-1}}$. Now we can define a new vector **rhs** as follows:

$$\mathbf{rhs} = \mathbf{e}_{AF_{j_1},A_{j_1}} + \mathbf{e}_{A_{j_1},C_{i_1}} - \mathbf{e}_{C_{i_1},A_{j_2}} + \cdots - \mathbf{e}_{C_{i_{q-1}},A_{jq}} - \mathbf{e}_{A_{jq},AF_{jq}}.$$

It is easy to see that **rhs** has as entries 0 for all components except those corresponding to nodes $B_{j_1}r_p$ for which the entry is -a_p, and those corresponding to nodes $B_{j_q}r_p$ for which the entry is a_p. We now consider the set of nodes that are unique to each subcycle. For the pth such subcycle (assuming the set of nodes are of type 4b), we define a vector $\mathbf{lhs_p}$ as follows:

$$\mathbf{lhs_p} = -\mathbf{e}_{B_{jq}r_p,A_{jq}r_p} - \mathbf{e}_{A_{jq}r_p,S_{k_{qp}-1}} + \cdots + \mathbf{e}_{S_{k_1},A_{j_1}r_p} + \mathbf{e}_{A_{j_1}r_p,B_{j_1}r_p}.$$

It is easy to see that $\mathbf{lhs_p}$ has as entries 0 for all components except that corresponding to node $B_{j_1}r_p$ for which the entry is $+1$, and that corresponding to node $B_{j_q}r_p$ for which the entry is -1. It then follows that $\mathbf{rhs} + \sum_{p\in[P]} a_p\mathbf{lhs_p} = \mathbf{0}$ establishing linear dependence. □

Corollary 1 *$\mathcal{B}$ does not consist of columns corresponding to the arcs of any type of cycle along which flow can be augmented.*

Proof Follows from the non-singularity of any basis and Lemma 1. □

Let us define arcs *associated with the jth assembly operation* as the following set of $P+1$ arcs: $(AF_j, A_j), (A_jr_1, B_jr_1), \dots, (A_jr_P, B_jr_P)$.

Lemma 2 *$\mathcal{B}$ consists of at least P columns (arcs) associated with the jth assembly operation.*

Proof Consider the first case where column $\mathbf{e}_{AF_j,A_j}$ is not a part of the basis. If there were less than P other columns associated with the jth assembly operations in the basis, it implies that there is at-least one other column $\mathbf{e}_{A_jr_p,B_jr_p}$ not a part of the basis. Hence, all entries of the row in the basis corresponding to node B_jr_p are 0, contradicting the non-singularity of the basis.

The second case is where column $\mathbf{e}_{AF_j,A_j}$ is a part of the basis. If there were less than P other columns associated with the jth assembly operations in the basis, it implies that there are at least two rows in the basis associated with nodes $B_jr_{p_1}$ and $B_jr_{p_2}$ whose only non-zero entries are from the non-zero entries of column $\mathbf{e}_{AF_j,A_j}$. The two rows are, therefore, linearly dependent, again contradicting the non-singularity of the basis. □

For any node n, let u_n represent its node potential (dual variable). Let the cost of unit flow along arc (f,t) in the manufacturing network be denoted by $c_{f,t}$.

Since there is no incoming arc at node AF_j, the constraint set of problem **[MCNF]** does not contain a constraint corresponding to AF_j nodes. None-the-less, it will be convenient to associate a node potential, u_{AF_j}, with node AF_j, that is related to the node potentials for nodes of type B_jr_p, $u_{B_jr_p}$, as follows:

$$u_{AF_j} = \sum_{p\in[P]} a_p u_{B_jr_p}. \tag{5}$$

This association presents no difficulty, conceptually or mathematically, since the reduced-cost of arc (AF_j, A_j), which is $\sum_{p\in[P]} a_p u_{B_jr_p} + c_{AF_j,A_j} - u_{A_j}$ can, conveniently and intuitively, be written as $u_{AF_j} + c_{AF_j,A_j} - u_{A_j}$. Equation (5), therefore, can be considered to represent the complementary slackness conditions for the nodes associated with the jth assembly operation.

Using (1), any lower or upper bound on the flow on arc of type (A_jr_p, B_jr_p) can be translated into an equivalent bound on arc of type (AF_j, A_j). Hence, without loss of generality, the lower and upper bounds on all arcs of type (A_jr_p, B_jr_p) can be taken to be 0 and ∞ respectively. This leads us to the following result.

Theorem 1 *There exists an optimal basis in which all arcs of type (A_jr_p, B_jr_p) are a part of the basis.*

Proof Consider an optimal basis $\mathcal{B}^*$ in which arc (A_jr_p, B_jr_p) is not a part of the basis. The flow on this arc is at its lower bound of zero.

Let u_1 and u_2 be the node potentials associated with nodes A_jr_p and B_jr_p respectively. The optimality conditions imply that the reduced-cost, $c_{A_jr_p,B_jr_p} + u_1 - u_2 \geq 0$.

Now, as a result of Lemma 2, arc (AF_j, A_j) is a part of the basis. Let $u_3 = \sum_{p\in[P]} a_p u_{B_jr_p}$ and let u_4 represent the node potential associated with node A_j. Then, $u_4 = u_3 + c_{AF_j,A_j}$.

Remove arc (AF_j, A_j) from $\mathcal{B}^*$, set the associated flow to its lower bound (of zero), and introduce arc (A_jr_p, B_jr_p) into the basis to obtain the new basis $\mathcal{B}^{**}$. As a result of this modification, the reduced-cost of arc (AF_j, A_j) increases by $a_p(c_{A_jr_p,B_jr_p} + u_1 - u_2) \geq 0$. Since the flow on arc (AF_j, A_j) is at its lower bound, the optimality conditions still hold, and $\mathcal{B}^{**}$ is an optimal basis. (Note that this conclusion is valid only if we can prove that in changing from basis $\mathcal{B}^*$ to basis $\mathcal{B}^{**}$ the potential of node A_j is not altered. The proof for this will follow from Lemma 3 that we prove subsequently.) □

The sub-graph associated with the bases of problem **[MCNF]** can be considered a generalization of the spanning tree associated with the usual network-simplex method. The following definition provides this generalization.

Definition *Generalized Tree*: A generalized tree for a manufacturing network is a connected graph with no cycles along which flow can be augmented. Nodes corresponding to the assembly operation: $(AF_j,\ B_jr_1, \ldots, B_jr_P)$ are considered connected only if at least P arcs amongst $(A_jr_1, B_jr_1), \ldots, (A_jr_P, B_jr_P)$ and (AF_j, A_j) are part of the tree.

For the sake of brevity, we refer to the generalized tree here-after as simply the g-tree.

In the context of the g-tree, the P arcs (A_jr_p, B_jr_p), $\forall p \in [P]$, provide connectivity only to the B_jr_p nodes and the AF_j node. In other words, the presence of these P arcs alone does not provide for the connectivity of the A_jr_p nodes.

Theorem 2

(a) *Every basic solution of* **[MCNF]** *corresponds to a g-tree.*
(b) *Every g-tree corresponds to a basic solution of* **[MCNF]**.

Proof (a) (Modified from Fang and Qi 2003) We will begin the network-simplex-based procedure by creating a special zeroth node and creating artificial arcs connecting this zeroth node to all other nodes in the manufacturing network. As the procedure continues, a few of these artificial arcs will leave the current basis. Since the basis $\mathcal{B}$ associated with a basic solution is non-singular, it follows from Corollary 1 that the sub-graph associated with the basic solution does not contain any cycle along which flow can be augmented. Now assume the graph associated with the basis is not connected. By re-arranging the basis, we can write the basis in form

$$\mathcal{B} = \begin{pmatrix} \mathbf{S} & 0 \\ 0 & \mathbf{V} \end{pmatrix}$$

where **S** corresponds to nodes that are connected to the zeroth node, and **V** corresponds to nodes unconnected to the zeroth node. The rows in **V** are completely in **A** and hence their sum is **0**, contradicting the non-singularity of $\mathcal{B}$. From Lemma 2, we also know that there exist at least P arcs amongst $(A_j r_1, B_j r_1), \dots, (A_j r_P, B_j r_P)$ and (AF_j, A_j) that are part of the g-tree.

(b) The only possible cycles along which flow can be augmented in the graph are the four types that were identified previously. Since the g-tree by definition does not contain any of these, it follows that the g-tree corresponds to a basic solution. □

4 Network-simplex-based algorithm

We now describe, conceptually, the network-simplex-based solution procedure. Associated with any basic feasible solution are arcs in the network which can either

1. be a part of the g-tree (basis), in which case the arc is said to belong to set **T** of arcs, or
2. belong to the set **L** of non-basic arcs whose flows are at their lower bounds, or
3. belong to the set **U** of non-basic arcs whose flows are at their upper bounds.

The network-simplex-based method for the distribution problem proceeds iteratively through the following steps.

Step 1. Find a feasible g-tree structure specified by the sets **T**, **L** and **U** of arcs.
Step 2. Compute the flow and node potentials associated with this g-tree structure.
Step 3. Determine if some arc (k, l) violates the optimality conditions. If no such arc exists, terminate with the current solution being optimal. Otherwise go to step 4.
Step 4. Add arc (k, l) to the g-tree and determine the leaving arc (q, t).
Step 5. Perform a g-tree update and update the flow and node potentials. Go back to step 3.

We now elaborate on how each one of the steps above can be performed efficiently.

4.1 Steps 1 and 2: finding an initial basic feasible solution and computing node potentials

To start the network-simplex-based method, we will construct an all-artificial basis. To do this, we add a new artificial node to the network and call it the zeroth node. *Artificial arcs*, with a lower bound of zero and an upper bound of ∞, connect the zeroth node to all nodes in the network except nodes associated with the assembly operation. The cost of unit flow on these arcs is set to a large positive value, M. The only exception relates to arcs between the zeroth node and the component nodes, r_p, that are not charged any cost. As a result of Theorem 1, all arcs of type $(A_j r_p, B_j r_p)$ are included as a part of the g-tree.

The flows on the original arcs of the manufacturing network are set to their lower bounds. The directions and flows on the artificial arcs are, then, set such that there is flow balance at all nodes of the manufacturing network, including the nodes associated with the assembly process, for which the flow balance requirement is specified by (1).

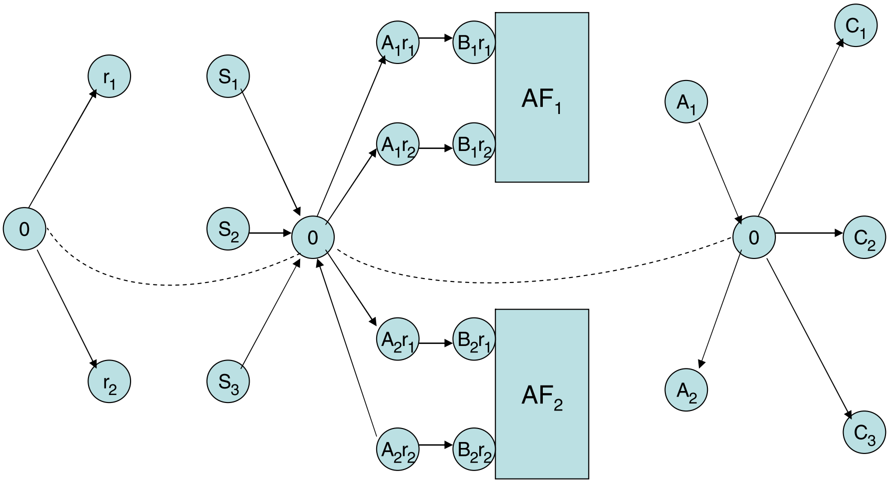


Fig. 3 Example of a starting basis

An example is provided in Fig. 3. Dashed lines are used to indicate that the zeroth node that appears in three different locations refers to the same node. In this example, if $a_1 = 1$, $a_2 = 2$, $d_1 = 5$, $d_2 = 10$, $d_3 = 15$ and $L_1 = 10$, the flows on all basic arcs except the following are zero:

$$\begin{aligned}
z_{0,A_1r_1} &= z_{A_1r_1,B_1r_1} = 10,\\
z_{0,A_1r_2} &= z_{A_1r_2,B_1r_2} = 20,\\
z_{A_1,0} &= 10,\\
z_{0,C_1} &= 5,\\
z_{0,C_2} &= 10,\\
z_{0,C_3} &= 15.
\end{aligned}$$

The node potentials can be calculated using the g-tree structure and complementary slackness conditions. We may assume the node potential for the zeroth node as being equal to 0. For the starting basis shown in Fig. 3, the various node potentials are indicated in Table 1.

4.2 Step 3: testing for optimality

Once the node potentials are calculated, the following procedure is used to determine if the current basis is optimal.

1. Let $\psi_1 = \{(k,l) \in \mathbf{L} : c_{k,l} + u_k - u_l < 0\}$, and let $\psi_2 = \{(k,l) \in \mathbf{U} : c_{k,l} + u_k - u_l > 0\}$.
2. If $\psi_1 \cup \psi_2 = \emptyset$, terminate with the present flows being optimal. Otherwise, select $(k,l) \in \psi_1 \cup \psi_2$ as the arc that violates the optimality conditions.

Table 1 Node potentials for starting basis of Fig. 3

Node, i	Node potential, u_i
r_1	0
r_2	0
S_1	$-M$
S_2	$-M$
S_3	$-M$
A_1r_1	M
A_1r_2	M
A_2R_1	M
A_2R_2	$-M$
B_1R_1	M
B_1R_2	M
B_2R_1	M
B_2R_2	$-M$
AF_1	$3M$
AF_2	$-M$
A_1	$-M$
A_2	M
C_1	M
C_2	M
C_3	M

Before we discuss steps 4 and 5, we need to develop appropriate data structures for the network-simplex-based method. The sections that follow assume familiarity with the data structures developed in the Appendix.

4.3 Step 4: pivoting

On adding an arc (k, l) that violates the optimality conditions to the g-tree, one of the four types of flow-augmenting cycles is formed in the graph. We use the root index along with the predecessor index to find this cycle which we refer to as the *pivoting* cycle.

If $\text{root}(k) = \text{root}(l) \neq 1$, then the cycle formed is of type 1, 2 or 3. By backtracking to the common root-node, the nodes belonging to the cycle can be identified. The maximum value, δ units, of flow that can be sent around this cycle before the flow on one of its component arcs reaches either its lower or upper bounds is found out just as is the case in the usual network-simplex method. Any arc along which the flow reaches this bound is chosen as the leaving arc (q, t). If $\text{root}(k) = 1$ and $\text{root}(l) = 1$, both nodes, on backtracking, could lead to the same AF_j node. In this case, the cycle is of type 3 and can be dealt with similarly.

If, however, $\text{root}(k) \neq \text{root}(l)$, a variety of cases could arise that require careful consideration.

If $\text{root}(k) = 1$ and $\text{root}(l) = 0$, we know that either node k is an AF_j node, or that we can backtrack from node k to an AF_j node. The B_jr_p nodes associated with this

AF_j node all necessarily have a root of 0. Hence, we can backtrack along a path in the g-tree from each one of the B_jr_p nodes to the zeroth node. This path corresponds to a path for component of type p to reach the jth assembler. From node l, we can directly backtrack to the zeroth node. A flow of δ of the end-product on this complex cycle corresponds to a flow of $a_p\delta$ along the path corresponding to the pth component. The leaving arc (q,t) is the arc along which the flow reaches either its lower or upper bound first. This arc could be either of an end-product type, or of the type of any one of the P components. Theorem 1 allows us to disregard consideration of any arc of type (A_jr_p, B_jr_p) as candidates for leaving arc. A similar analysis can be used if root$(k) = 0$ and root$(l) = 1$. An example where such a cycle results is when arc (A_1, C_1) is chosen as the entering arc in the g-tree shown in Fig. 11. If root$(k) = 1$ and root$(l) = 1$, and both nodes lead to different AF_j nodes, a complex cycle of type 4 results. The leaving arc can be found out using a similar analysis.

If the root of node l is B_jr_p, while that of node k is different, and entering arc (k,l) is an arc associated with a component type commodity and not an end-product commodity, we first backtrack from node l to node B_jr_p. From the other B_jr_p nodes associated with the jth assembly operation, we backtrack to their respective roots, which are either zero, or $B_{j_1}r_p$ nodes associated with the j_1 th assembly operation, where $j_1 \neq j$.

The root of the assembler node AF_j associated with B_jr_p could be either zero or a different B_jr_p node. In case of the former, we backtrack from the AF_j node to the zeroth node. In case of the latter, we bactrack from the AF_j node, to AF_{j_2} node, where $j_2 \neq j$. From each node $B_{j_2}r_p$, $p \in [P]$, we backtrack to their respective root-nodes. The above procedure is likewise repeated for node k. After backtracking from node k, we identify the complex cycle in the g-tree. By identifying the maximum possible flow of δ units that may be augmented along the arcs of this complex cycle associated with the end-product, we can find the leaving arc (q,t). A similar analysis results if root of node k is a B_jr_p node, while that of node l is a different node.

An example will help clarify this point. In Fig. 11, if arc (r_2, S_2) were to enter the g-tree, the following complex cycle is created, which can be found by appropriately backtracking to the root indices of the nodes associated with the entering arc.

$$\begin{aligned}
&\text{End-product type:} && AF_1 \to A_1 \to C_3 \to 0,\\
&\text{Component 1:} && B_1r_1 \to A_1r_1 \to S_3 \to r_1 \to 0,\\
&\text{Component 2:} && B_1r_2 \to A_1r_2 \to S_2 \to r_2 \to 0,\\
&\text{Component 3:} && B_1r_3 \to A_1r_3 \to S_4 \to r_3 \to 0.
\end{aligned}$$

If the entering arc is associated with the end-product, a similar procedure of first backtracking to an AF_j node followed by backtracking to the roots of the associated B_jr_p nodes can be employed if the root of node l is a B_jr_p node. This procedure is repeated for node k. The complex cycle is identified as before, and so is the leaving arc (q,t).

For cycle types 1, 2 and 3, if δ units of additional flow were found to be the maximum possible before flow on arc (q,t) attained its bounds, the flow along the various arcs of the cycle is augmented by this amount. In case of a complex cycle, the flow along arcs associated with the pth component is augmented by $a_p\delta$, where δ is the amount of flow that can be augmented along arcs associated with the end-product.

The root index, thus, helps in identifying the pivoting cycle that is formed in the graph when arc (k,l) is added to the g-tree. Note that, similar to the case of the usual network-simplex method, the flows in the network change only for those arcs that are a part of the pivoting cycle. Hence, there is no need to re-calculate the flows along arcs that are not a part of the pivoting cycle. The root index will also play a crucial role in updating the node potentials, which we discuss in the next section.

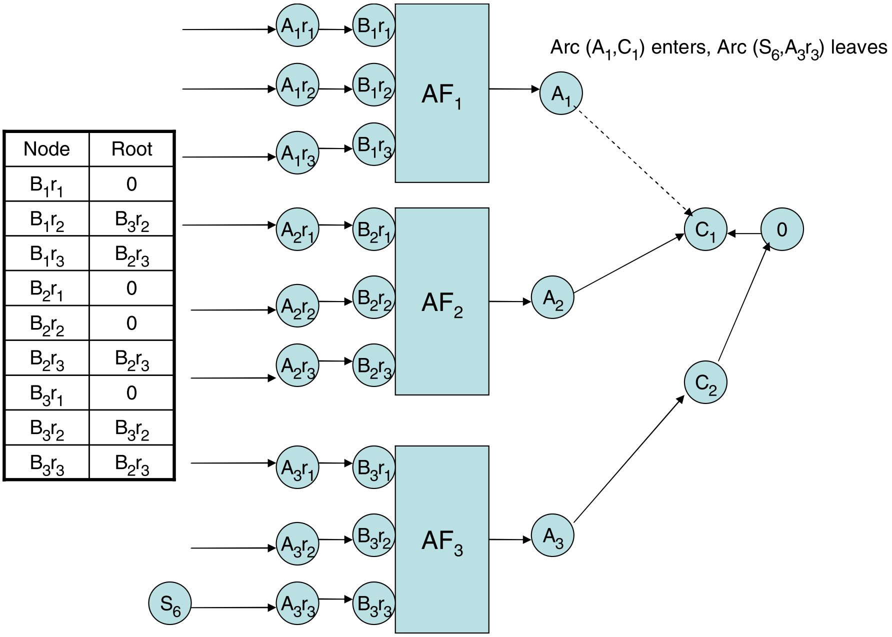
4.4 Step 5: updating g-tree indices and node potentials

The final step that needs clarification is how the g-tree indices and node potentials are updated after pivoting, without having to re-calculate them from scratch for all nodes in the network.

An efficient method to update the indices that is available for the usual network-simplex method is described in Bazaraa et al. (1990). We generalize this method for the case of the manufacturing network as follows.

The method involves first finding out the "stem", which is a special path in the network that connects the entering arc to the leaving arc. The stem consists of those nodes for which the predecessor index would necessarily change as a result of the pivot. The predecessor indices for all other nodes of the network remain unchanged after the pivot. To find out the stem, we begin by noting that either node k or node l or both become disconnected on removal of arc (q,t) from the g-tree. By a node being disconnected, we mean that its potential would have to change as a result of the pivot. The question of which one of the nodes k or l is disconnected can be answered by examining whether the leaving arc is encountered when backtracking from node k (in which case node k is disconnected) or node l (in which case node l is disconnected). This is best illustrated by means of an example.

In Fig. 4, we need to find out whether leaving arc (S_6, A_3r_3) causes the disconnectedness of node A_1 or C_1. Let us begin by finding out whether A_1 is disconnected. Using the predecessor indices, we backtrack from node A_1. We encounter node AF_1. Since node AF_1 is a node associated with an assembly operation, we need to backtrack to the roots of each one of the other nodes associated with the assembly operation. This is so because, changing the potential of any node associated with the assembly operation requires modifying the potential of some other node associated with the same assembly operation to maintain the complementary slackness conditions represented by (5). Thus, from B_1r_1, we can backtrack to its root-node of zero, without having encountered the leaving arc. From B_1r_2, we can backtrack to its root-node, B_3r_2, without having encountered the leaving arc. Since this node (B_3r_2) is associated with another assembly operation, we examine other nodes associated with it. Thus, we examine backtracks from node B_3r_1 to its root-node of zero and do not encounter the leaving arc. If we then examine backtracks from node AF_3 to its root-node of zero, we would still not encounter the leaving arc. However, on backtracking from node B_3r_3 to its respective root-node of B_2r_3, we do encounter the leaving arc. We can thus conclude that node A_1 is indeed disconnected. The stem consists of those nodes encountered along the series of backtracks that led us to the conclusion of the



Node	Root
B_1r_1	0
B_1r_2	B_3r_2
B_1r_3	B_2r_3
B_2r_1	0
B_2r_2	0
B_2r_3	B_2r_3
B_3r_1	0
B_3r_2	B_3r_2
B_3r_3	B_2r_3

Fig. 4 Backtracking to find the stem

disconnectedness of node A_1. For instance, in this example, the stem would consist of the following nodes: $(A_1, AF_1, B_1r_2, A_1r_2, \ldots, A_3r_2, B_3r_2, AF_3, B_3r_3, A_3r_3)$.

Once the stem is identified, we update the indices and potentials of nodes in the stem. The predecessor index is simply reversed for the nodes featuring in the stem. This process is straight-forward except in the case of one type of stem. In case the stem consists of the following set of nodes in order: $(\ldots, A_jr_1, B_jr_1, AF_j, B_jr_2, A_jr_2, \ldots)$, then, pred$(B_jr_1)$ is set to be A_jr_1, the predecessor index of node AF_j is left unchanged, while that of B_jr_2 is set to AF_j.

Once the predecessor indices have been updated, the root indices are updated next. The root index of a node is set to be the same as that of its updated predecessor. The root indices of B_jr_p and AF_j nodes encountered on the stem, are updated based on the rules specified in the Appendix.

We now state a result that will prove useful in subsequent analysis.

Lemma 3 *If the root of an AF_j node is not equal to* 1, *exactly one amongst the P B_jr_p nodes associated with it and the A_j node has its predecessor set to be equal to the AF_j node.*

Proof Note that by construction of the all-artificial starting basis, the roots of all AF_j nodes are 1 at the beginning of the network-simplex-based method. This implies, that to begin with, the root-node of each of the B_jr_p nodes associated with the jth assembly operation is zero. Let us examine when the root of an AF_j node becomes different from 1 for the first time.

We analyze one situation when this change occurs. The analysis is similar for other instances. One instance when the root of an AF_j node becomes different from 1 for the first time occurs when the entering arc is arc (k, l) with root$(k) = 1$, root$(l) \neq 1$ and backtracking from node k using the predecessor indices leads to the AF_j node. The leaving arc (q, t) should be such that it results in the disconnectedness of node k *and* node AF_j. (In this case, after performing the pivot operation, node k would inherit the same root-node as node l and node AF_j would in-turn inherit the same root-node.) For this to happen, it is necessary for the leaving arc (q, t) to be of component type p that lies in path obtained by backtracking from one of the B_jr_p nodes to its root-node of 0. Assuming without loss of generality that pred$(t) = q$, the stem appears as shown: $(k, \ldots, A_j, AF_j, B_jr_p, \ldots, t)$. On updating the indices, we have pred$(k) = l, \ldots,$ pred$(AF_j) = A_j$, pred$(B_jr_p) = AF_j$, and so on. The root indices are updated as shown: root$(k) = \cdots =$ root$(AF_j) =$ root(l), root$(B_jr_p) = \cdots =$ root$(t) = B_jr_p$. The path connecting the other B_jr_p nodes to the zeroth node remains unaffected and the same holds true for the indices of nodes in this path. Thus, the first time that the root of an AF_j node changes from 1, the lemma holds.

To complete the proof of this lemma, it remains to show that any updating operations performed subsequently in the network-simplex-based algorithm do not lead to a violation of the property of the lemma.

Note that any updating operation begins with the creation of the stem. This stem, in general, can be composed of any number of nodes associated with the assembly process. Regardless of the *number* of such nodes, there are just three *types* of sequences of nodes that can feature in the stem:

1. $(\ldots, A_jr_1, B_jr_1, AF_j, B_jr_2, A_jr_2, \ldots)$—in this case, the predecessor of B_jr_1 is set equal to A_jr_1, while that of B_jr_2 is set equal to AF_j. The root of B_jr_1 is set equal to that of A_jr_1, while that of B_jr_2 is set to itself.
2. $(\ldots, A_j, AF_j, B_jr_1, A_jr_1, \ldots)$—in this case, since we have been able to backtrack from node A_j to node AF_j using the original predecessor indices, it follows that before the updating process, pred$(A_j) = AF_j$. The predecessor of all associated B_jr_p nodes are the respective A_jr_p nodes. After the updating process, the predecessor of B_jr_1 is set equal to AF_j, while that of the A_j node is set to the node that precedes it in the stem. The root-node of B_jr_1 is set to itself.
3. $(\ldots, A_jr_1, B_jr_1, AF_j, A_j, \ldots)$—in this case, pred$(B_jr_1)$ is set equal to A_jr_1, root(B_jr_1) is set equal to root(A_jr_1), pred(A_j) is set equal to AF_j and root(A_j) is set equal to root(AF_j).

None of these three cases leads to a situation in which the lemma is violated. Hence, the proof stands complete. □

(We are now in a position to prove a claim that was needed to complete the proof of Theorem 1. Consider the g-tree associated with basis $\mathcal{B}^*$ in the proof of Theorem 1. Since arc (A_jr_p, B_jr_p) was not a part of the g-tree, the root of node B_jr_p must have been set to itself. Therefore, its predecessor would be node AF_j. Consequently, as a result of Lemma 3, the predecessor of node A_j is not node AF_j. However, since arc (AF_j, A_j) forms a part of the g-tree, the predecessor of node AF_j is necessarily node A_j. Thus, the potential of node A_j is unaffected by the new potential of node AF_j in basis $\mathcal{B}^{**}$.)

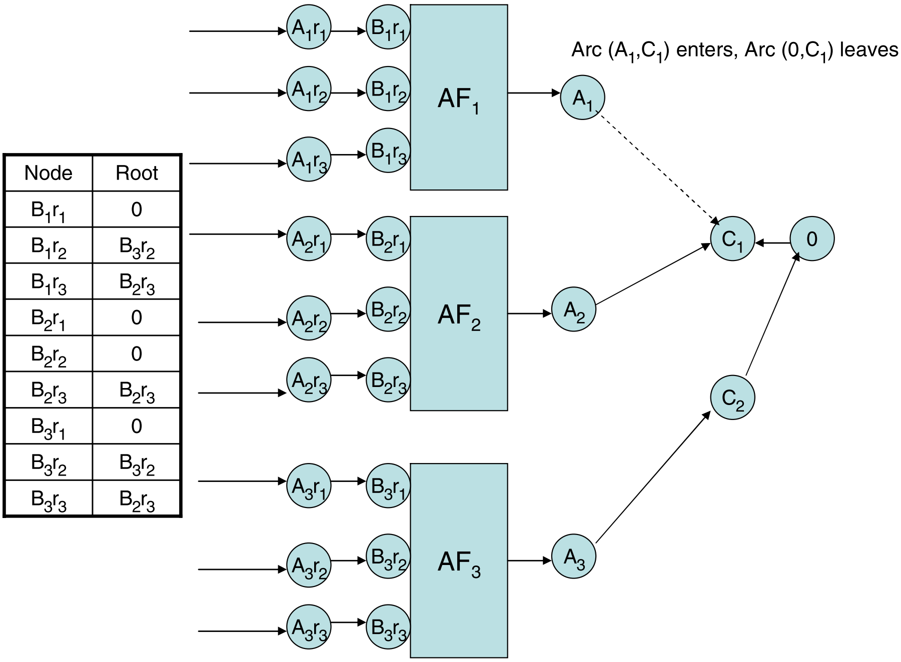
The system of linear equations that are required to be solved during every iteration of the network-simplex-based algorithm relate to the complementary slackness conditions. This set of conditions require assigning node potentials such that the reduced-cost of arcs belonging to **T** is 0. In case of the usual network-simplex method, this system of linear equations can be solved directly on the spanning tree that is associated with the set **T** of arcs belonging to the basis. However, in our case, we have a new set of complementary slackness conditions for the nodes associated with the assembly operation, represented by (5). Despite this complication, we now outline how the node potentials can be assigned using operations performed on the g-tree itself.

The node potentials of the stem are first updated using the updated predecessor indices. Special care is taken when dealing with nodes of the stem that are associated with an assembly operation. For instance, if the nodes encountered in the stem are $(\dots, A_jr_1, B_jr_1, AF_j, B_jr_2, A_jr_2, \dots)$, the node potential of B_jr_1 is updated using the node potential of A_jr_1. Let the *change* (new node potential-old node potential) in node B_jr_1's potential be Δ. The node potential of node AF_j is left unchanged, while that of node B_jr_2 is *decreased* by $\frac{\Delta a_1}{a_2}$. If the nodes encountered in the stem are $(\dots, A_jr_1, B_jr_1, AF_j, A_j, \dots)$, the node potential of B_jr_1 is updated using the node potential of A_jr_1. If the *change* in node B_jr_1's potential is Δ, the node potential of node AF_j is *increased* by Δa_1. If the nodes encountered in the stem are $(\dots, A_j, AF_j, B_jr_1, A_jr_1, \dots)$, the node potential of AF_j is updated using the node potential of A_j. If the *change* in node AF_j's potential is Δ, the node potential of node B_jr_1 is *increased* by $\frac{\Delta}{a_1}$. Note that all these changes in the potentials of nodes associated with the jth assembly operation continue to maintain the complementary slackness relationship represented by (5).

Once the potentials of the nodes in the stem have been updated, we utilize the thread indices to update the potentials of nodes that descend from the nodes in the stem. In doing so, we might update the potential of say, node B_jr_1, whose predecessor is A_jr_1. This necessitates updating the potential of some other node associated with the jth assembly operation in order to maintain the relationship represented by (5). If the root of the AF_j node is not equal to 1 and its predecessor is A_j, it implies that its node potential is fixed independently of the node potential of B_jr_1. Lemma 3, however, guarantees that in such a case, there exists exactly one other B_jr_p node whose root is equal to itself. If this node is node B_jr_2, its potential is *decreased* by an amount equal to $\frac{\Delta a_1}{a_2}$, where Δ is the change in the node potential of B_jr_1, in order to maintain the relationship represented by (5). The thread index is then used to update the descendents of node B_jr_2. Note that this process is valid since these descendents derive their node potentials from their root-node, which in this case is B_jr_2.

In this process of updating the node potentials of descendents of the stem, it could so happen that the node potential of a predecessor of a stem node is updated. Figure 5 illustrates one such case.

When arc (A_1, C_1) enters the g-tree and arc $(0, C_1)$ leaves the g-tree, node C_1 is disconnected. This node constitutes the stem. The potential of the node C_1 is updated to $u_{A_1} + c_{A_1,C_1}$. Using the thread indices, we now update the potentials of nodes that descend from the stem. Thus, we update the potentials of the following nodes: $A_2, AF_2, B_2r_3, A_2r_3, \dots, A_1r_3, B_1r_3, AF_1, A_1$. We seem to have reached a problem in that the updated potential of node C_1, on being used to update the potentials of its



Node	Root
B_1r_1	0
B_1r_2	B_3r_2
B_1r_3	B_2r_3
B_2r_1	0
B_2r_2	0
B_2r_3	B_2r_3
B_3r_1	0
B_3r_2	B_3r_2
B_3r_3	B_2r_3

Fig. 5 Updating node potentials

descendents, leads to a possible change of the potential of the node, A_1, that was used to update the potential of node C_1 in the first place. (This is the special case in which both nodes associated with the entering arc are disconnected.) However, Theorem 3 below establishes that this problem can be resolved and appropriate node potentials can be found using operations performed on the g-tree itself.

Theorem 3 *There exists a three-step procedure that can be implemented on the g-tree itself by which we can find appropriate node potentials satisfying the complementary slackness conditions for the stem and its descendents even when this results in an update of the potential of one of the predecessors of a stem node.*

Proof Without loss of generality assume that the first node in the stem is node l associated with the entering arc (k,l). On updating the potentials of the stem nodes, u_l is set the value of $u_k + c_{k,l}$. Let us define a new function $f_k(u)$ as the node potential of k on beginning with a potential of u for node l and using this value to update the potentials of the stem, and its descendents, eventually leading to the potential $f_k(u)$ for node k. Note that function $f_k()$ is well-defined since there is a sequence of nodes whose potentials are updated along the stem and the stem's descendents. The thread index ensures that this sequence itself is well-defined.

Observe that in obtaining the potential for node k from the potential for node l, we merely perform additions, subtractions, multiplication and division by rational numbers. Hence, $f_k(u)$ can be expressed as a linear function of u. If $f_k(u) + c_{k,l} = u$, we have found the appropriate node potentials. Since we know that the linear system of equations $u\mathcal{B} = c_{\mathcal{B}}$, where $c_{\mathcal{B}}$ is the vector of cost coefficients of variables featured in the basis has a solution for all bases encountered during the network-simplex-based method, we know that the linear function $f_k(u) - u + c_{k,l}$ has a zero. The zero of this linear function, u_l^*, can be found using at-most two function evaluations. For instance, if $f_k(u_1) - u_1 + c_{k,l} = k_1 \neq 0$, and $f_k(u_2) - u_2 + c_{k,l} = k_2 \neq 0$, by linear interpolation, $u_l^* = u_1 - k_1(\frac{u_2-u_1}{k_2-k_1})$. Each function evaluation requires beginning with a unique value for u_l, and performing potential updates for the nodes encountered on the stem and its descendents. This operation can be performed directly using the various data structures defined on the g-tree itself.

The third and final step involves actually using the value of u_l^* found to update the node potentials of the stem and its descendents. □

It is interesting to note that the procedure outlined in the proof of Theorem 3 is also one of the methods to update the node potentials of augmented trees (also called one-trees, which are spanning trees with one additional arc, thereby forming a cycle) associated with bases in the generalized network-simplex method. This method is described in Ahuja et al. (1993). Yet, Theorem 3 is an important result in the context of the generalized network-simplex-based method for manufacturing network flows for the reason that the set of nodes and arcs that lead to the update of the potential of a predecessor of a stem node do not form an augmented tree based on its usual definition, due to the presence of nodes representing the assembly operation. So, it is not immediately clear if there exists a procedure that is capable of solving the resultant system of linear equations efficiently. Theorem 3 indicates that there does exist such a procedure.

Once the node potentials are updated, the leaving arc (q,t) is removed from **T** and added to either **L** or **U** depending on whether the arc left the pivoting cycle in step 4 at either the lower or the upper bound respectively. Arc (k,l) is added to **T**. The depth index of every node in the stem is set to be one more than its predecessor. The thread index is again used to update the depths of descendents of the stem node. Finally, updating the thread indices can be performed as in the case of the usual network-simplex method based on the new predecessor and depth indices. This completes step 5 of the network-simplex-based method.

5 Example problem

In this section, we present the steps that the network-simplex-based algorithm goes through in solving an example problem to optimality. This example is presented on a 3 customer, 3 assembler, 6 supplier, 3 component problem. The demand of the three customers is 10, 15 and 20 units, respectively. The amount of components needed to produce 1 unit of the end product are given by the vector $\mathbf{a}^t = [1,2,4]$. Suppliers 1 and 2 provide component 1, suppliers 3 and 4 provide component 2, and the other suppliers provide component 3. The lower and upper bounds on various arcs are depicted in Fig. 6. For arcs not depicted in the figure the lower bounds are zero, while the upper bounds are infinite. There is a cost of 2 units for the unit flow of any commodity to and from assembler 3, while there is a cost of 1 unit for any unit flow to and from other assemblers. There is no cost for flow on any other arc in the

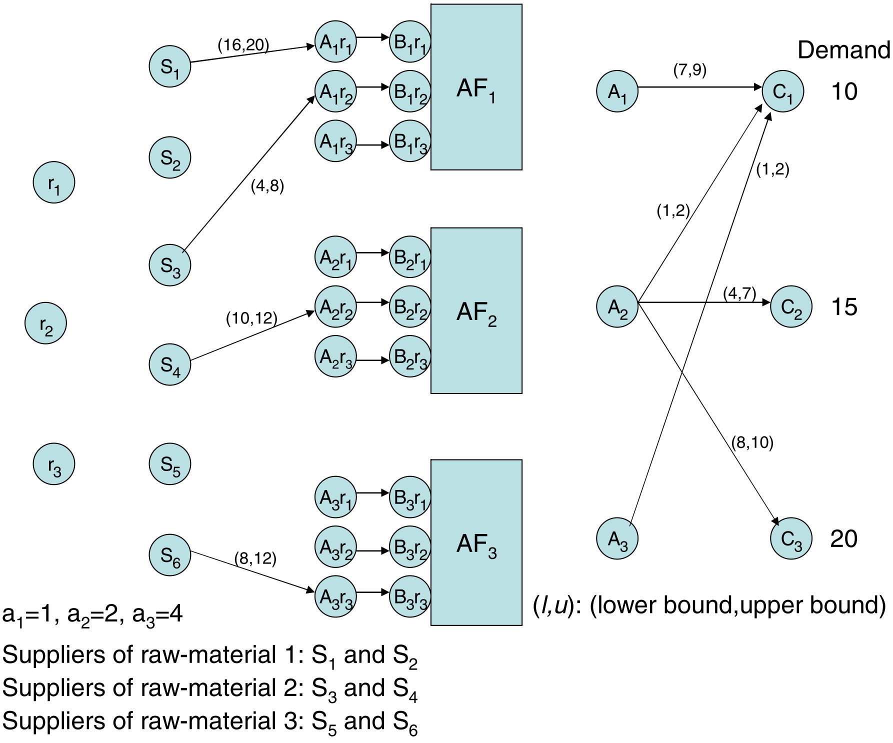


Fig. 6 Bounds for example problem

network. The cost of unit flow from the zeroth node to nodes other than r_p nodes is set at 1000 units. If there are multiple candidates for either the entering or leaving arcs, different rules have been studied in literature to select one amongst them. In this example, we select the arcs to illustrate the various concepts developed in this paper, and not according to any particular rule.

Figure 7 depicts the starting g-tree, in which the flows on all original arcs is set at the lower bound associated with the arc. The flow and orientation on other artificial arcs is set so as to balance the flow in the network.

We outline in Table 2 the various iterations of the network-simplex-based method. In the table, the term EA stands for the entering arc during that iteration, while LA stands for the leaving arc. The leaving arc can leave the basis and enter either **L** or **U** depending on whether it leaves the basis at the lower or upper bound respectively.

Figure 8 depicts the steps of iteration number 2. The stem is composed of the following nodes: (A_3, AF_3, B_3r_3, A_3r_3). We first update the potential of node A_3 to be $u_{A_3} = u_{c_1} - 2 = 998$. The potential of node A_3 changes to 998 from the earlier potential of -1000. This constitutes a change of $\Delta = 998 - (-1000) = 1998$. Since the next node in the stem corresponds to component 3, its potential likewise *increases* by $\frac{\Delta}{a_3} = 499.5$ from its earlier potential of -1000, finally becoming $u_{B_3r_3} = -500.5$. It can now be verified that the new node potentials still satisfy the complementary slackness condition denoted by (5). The predecessor of B_3r_3 is made AF_3, and the root of node B_3r_3 is set to itself. That B_3r_3 is the only node amongst nodes $B_3r_p, \forall p \in [P]$

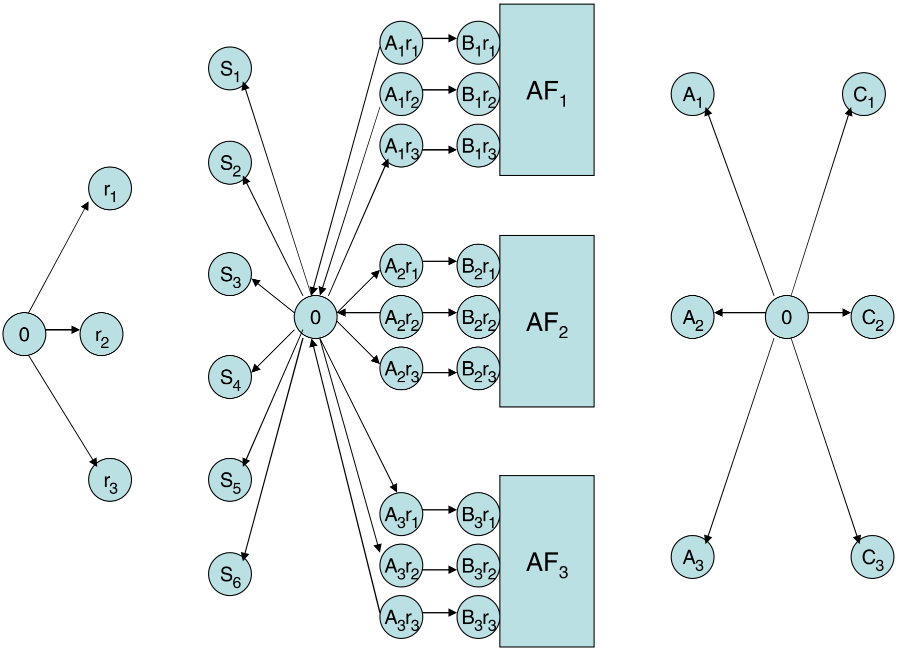


Fig. 7 Starting G-tree

to have its root set to itself is due to Lemma 3. This fact will become important in the node potential updates of a subsequent iteration.

In iteration number 12, depicted in Fig. 9, arc (S_1, A_3r_1) enters the basis, while arc $(0, A_3r_1)$ leaves. This causes the potential of node A_3r_1 to be updated to $u_{A_3r_1} = u_{S_1} + 2 = 2$. The potential of node B_3r_1 changes to 2 from the earlier potential of 1000. This constitutes a change of $\Delta = 2-(1000) = -998$. However, root$(AF_3) = 0$, implying that its potential cannot be changed since it is derived directly from the zeroth node. Yet, from Lemma 3, we know that there exists some other B_3r_p node whose root is set to itself. To maintain the complementary slackness conditions, the potential of this node should be changed. In our example, root$(B_3r_3) = B_3r_3$. The potential of this node is *decreased* by $\frac{a_1\Delta}{a_3} = -249.5$ from its earlier potential of -500.5, finally becoming $u_{B_3r_3} = -251$. Note that the complementary slackness condition depicted by (5) still hold. We now update the potential of node A_3r_3 to be -251.

In iteration number 19, depicted in Fig. 10, arc (S_1, A_2r_1) enters the basis, while arc $(0, A_2r_1)$ leaves. This causes the potential of node A_2r_1 to be updated to $u_{A_2r_1} = u_{S_1} + 1 = 1$. The potential of node B_2r_1 changes to 1 from the earlier potential of 1000. This constitutes a change of $\Delta = 1 - (1000) = -999$. Now, root$(AF_2) = 1$, implying that its potential can be changed when the potential of one of the B_2r_p nodes changes. Thus, the potential of node AF_2 *increases* by $a_1\Delta$ from its present potential of 3000, finally becoming $u_{AF_3} = 2001$. It can be observed that this change retains the relationship depicted by (5).

Table 2 Iterations for the example problem

Iteration number	Details
1	EA:(AF_3, A_3), LA:$(0, A_3)$ to **L**
2	EA:(A_3, C_1), LA:$(A_3r_3, 0)$ to **L**
3–8	EA:(r_1, S_1), LA:$(0, S_1)$ to **L**. EA:(r_1, S_2), LA:$(0, S_2)$ to **L**. EA:(r_2, S_3), LA:$(0, S_3)$ to **L** EA:(r_2, S_4), LA:$(0, S_4)$ to **L**. EA:(r_3, S_5), LA:$(0, S_5)$ to **L**. EA:(r_3, S_6), LA:$(0, S_6)$ to **L**
9	EA:(S_5, A_1r_3), LA:$(A_3r_3, 0)$ to **L**
10	EA:(AF_1, A_1), LA:$(0, A_1)$ to **L**
11	(S_3, A_1r_2) switches from **L** to **U**
12	EA:(S_1, A_3r_1), LA:$(0, A_3r_1)$ to **L**
13	EA:(S_4, A_3r_2), LA:$(0, A_3r_2)$ to **L**
14	EA:(S_4, A_1r_2), LA:$(0, A_1)$ to **L**
15	EA:(A_1, C_1), LA:$(0, C_1)$ to **L**
16	EA:(A_1, C_2), LA:$(A_1r_1, 0)$ to **L**
17	EA:(S_2, A_1r_1), LA:$(0, C_2)$ to **L**
18	EA:(A_1, C_3), LA:$(0, C_3)$ to **L**
19	EA:(S_1, A_2r_1), LA:$(0, A_2r_1)$ to **L**
20	EA:(S_5, A_2r_3), LA:$(0, A_2r_3)$ to **L**
21	EA:(AF_2, A_2), LA:$(A_2r_2, 0)$ to **L**
22	EA:(S_3, A_2r_2), LA:$(0, A_2)$ to **L**, Optimal Solution

EA: Entering Arc. LA: Leaving Arc

6 Conclusion

In this paper, we presented a graph-theoretic characterization of the bases associated with manufacturing network flow problems. We developed data structures to assist in a network-simplex-based solution methodology for manufacturing network flow problems. We also showed how a system of linear equations that arises during the solution procedure can be solved using the g-tree underlying the network-simplex-based method itself.

The methodology developed in this paper assumes a single type of end-product. The mathematical model for the problem with multiple types of end-products, each requiring the combination of component raw-materials in different proportions, exhibits a block-diagonal structure, with a set of common linking constraints and a set of constraints specific to each end-product type. Efficient solution procedures for problems exhibiting such structure decompose the problem into a master problem and many sub-problems. Each sub-problem is specific to a single end-product type and the methodology developed in this paper can be used to efficiently solve such sub-problems.

We have shown theoretically satisfactory update procedures for the network-simplex-based method for manufacturing network flows. In codes that implement network-simplex solution methods, indices in addition to the augmented thread indices are utilized for computational advantages. For the usual network-simplex

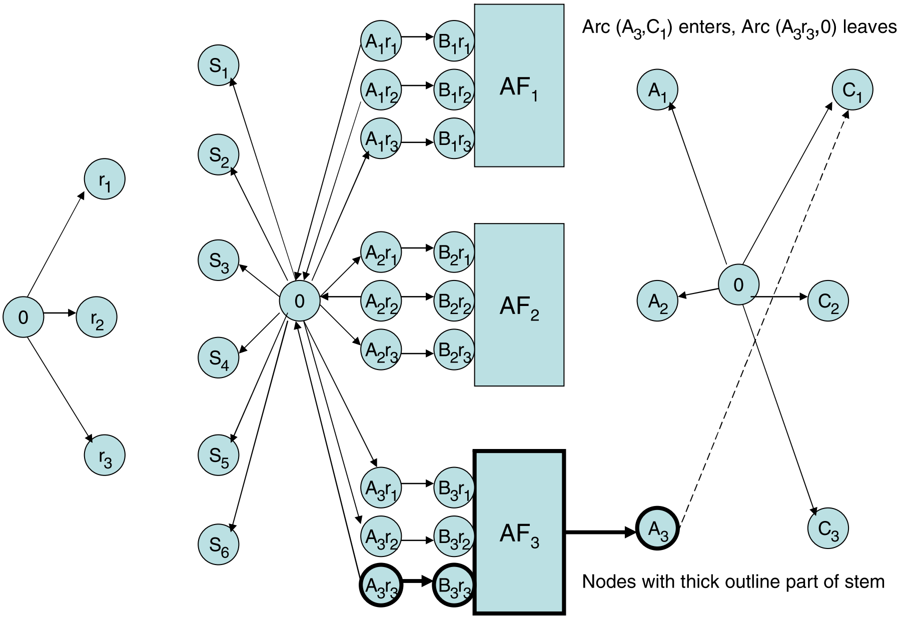


Fig. 8 Iteration 2

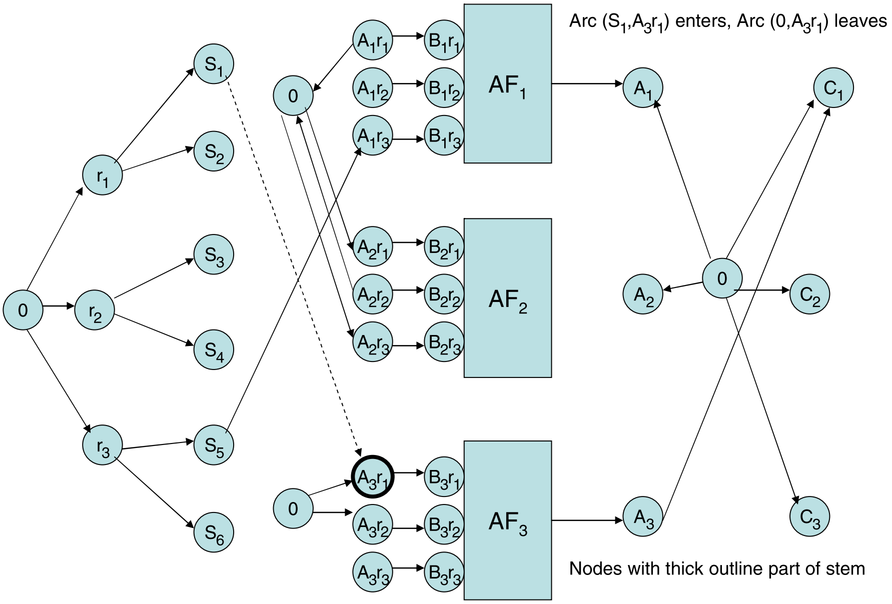


Fig. 9 Iteration 12

method, these include the *final* index of a node, the reverse thread index and the size of the sub-tree rooted at every node (Bazaraa et al. 1990). In light of this, certain

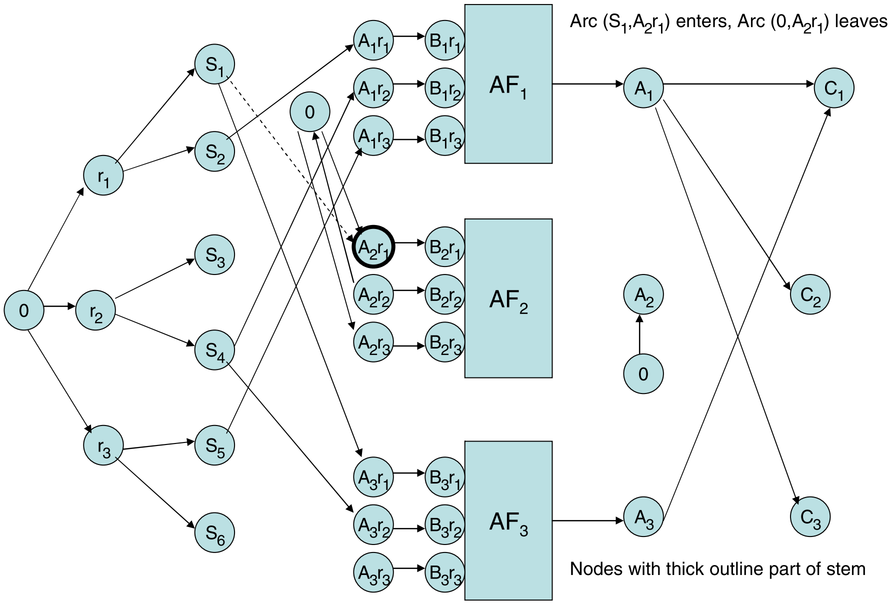


Fig. 10 Iteration 19

additional data structures may need to be developed specifically for manufacturing network flows for effective implementations of the network-simplex-based method. All these additional data structures can be derived from the basic data structures developed in this paper. We hope that these data structures can serve as a starting point for computational implementations of the network-simplex-based method.

Acknowledgements The authors thank two anonymous referees for their useful comments on an earlier version of the paper.

Appendix Data structures

A variety of data structures have been developed for efficient implementation of the network-simplex method for generalized non-manufacturing type networks. Excellent reference sources about these data structures include Ahuja et al. (1993), Bazaraa et al. (1990) and Kennington and Helgason (1980). One of the widely used data structures are the augmented thread indices first proposed by Johnson (1966) for the network-simplex procedure and subsequently generalized by Glover et al. (1974) for the generalized network-simplex procedure. In this paper, we show that the augmented thread indices (that consist of the well-known predecessor, depth and thread indices) along with a new data structure called as the root indices can be used for efficient implementation of the generalized network-simplex-based algorithm for manufacturing network flows.

Each node i in the manufacturing network flow model has a unique path connecting it to its root. This path can be traversed using the predecessor indices. There is

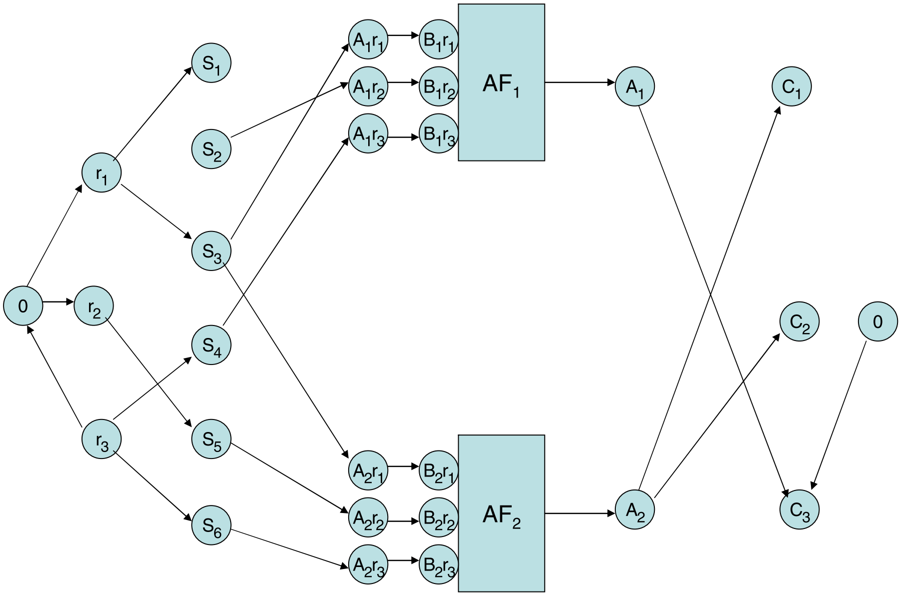


Fig. 11 Example of a G-tree

no single root in the network, but each node, i, has its own specific root, denoted by root(i). The depth of a node is the number of nodes in the path connecting it to its root plus the depth of its root-node. On starting with an all-artificial basis, each node (other than those associated with the assembly operation) is connected directly to the special zeroth node whose depth is taken to be equal to 0.

The root of node i is zero if there is a path connecting it to the zeroth node that does not involve an AF_j node. Root of an AF_j node is zero, if arc (AF_j, A_j) is a part of the g-tree, and root$(A_j) = 0$. If arc (AF_j, A_j) is not a part of the g-tree, the indices for node AF_j are set as follows:

If the root of every $B_j r_p$ node associated with node AF_j is zero,

1. Choose any $B_j r_p$ node associated with node AF_j, say, $B_j r_{p^*}$.
2. Set depth$(AF_j) = 1 +$ depth$(B_j r_{p^*})$. Set pred$(AF_j) = B_j r_{p^*}$.
3. Set root$(AF_j) = 1$. (Note: the root index of 1 is symbolic, and this value of 1 does not stand for any particular node in the network.)

Otherwise,

1. Amongst the $B_j r_p$ nodes associated with node AF_j whose roots are not zero, choose any $B_j r_p$ node, say, $B_j r_{p^*}$.
2. Set depth$(AF_j) = 1 +$ depth$(B_j r_{p^*})$. Set pred$(AF_j) = B_j r_{p^*}$.
3. Set root$(AF_j) =$ root$(B_j r_{p^*})$.

If the root of a $B_j r_p$ node cannot be set to zero due to the lack of a direct path leading to the zeroth node, its root is set in one of the following two possible ways:

1. If pred$(B_j r_p) = AF_j$, the root of a $B_j r_p$ node is set to itself.

Table 3 Indices for the G-tree in Fig. 11

Node, i	Pred(i)	Depth(i)	Thread(i)	Root(i)
r_1	0	1	S_1	0
r_2	0	1	S_5	0
r_3	0	1	S_4	0
S_1	r_1	2	S_3	0
S_2	A_1r_2	6	0	B_1r_2
S_3	r_1	2	A_1r_1	0
S_4	r_3	2	A_1r_3	0
S_5	r_2	2	A_2r_2	0
S_6	r_3	2	A_2r_3	0
A_1r_1	S_3	3	B_1r_1	0
A_1r_2	B_1r_2	5	S_2	B_1r_2
A_1r_3	S_4	3	B_1r_3	0
A_2r_1	S_3	3	B_2r_1	0
A_2r_2	S_5	3	B_2r_2	0
A_2r_3	S_6	3	B_2r_3	0
B_1r_1	A_1r_1	4	A_2r_1	0
B_1r_2	AF_1	4	A_1r_2	B_1r_2
B_1r_3	A_1r_3	4	S_6	0
B_2r_1	A_2r_1	4	AF_2	0
B_2r_2	A_2r_2	4	r_3	0
B_2r_3	A_2r_3	4	C_3	0
AF_1	A_1	3	B_1r_2	0
AF_2	B_2r_1	5	A_2	1
A_1	C_3	2	AF_1	0
A_2	AF_2	6	C_1	1
C_1	A_2	7	C_2	1
C_2	A_2	7	r_2	1
C_3	0	1	A_1	0

2. Otherwise, its root is the first B_jr_p node encountered in backtracking using the predecessor indices.

The root of all other nodes in the network is set to be that of its predecessor.

Figure 11 shows the g-tree associated with a hypothetical network basis. For this g-tree, the values for various indices is shown in Table 3.

Given the predecessor and depth indices, the thread index can be constructed using a depth-first-search process in linear time (Ahuja et al. 1993). The thread indices provide a convenient means of visiting all of node i's descendents before visiting any other node. Beginning at node i, we visit thread(i), followed by a visit to thread(thread(i)), and so on, until we encounter a node whose depth is lesser than or equal to that of node i.

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows. Prentice-Hall, New Jersey

Bazaraa MS, Jarvis JJ, Sherali HD (1990) Linear programming and network flows. Wiley, New York

Fang SC, Qi L (2003) Manufacturing network flows: a generalized network flow model for manufacturing process modeling. Optim Methods Softw 18:143–165

Geoffrion AM, Graves G (1974) Multicommodity distribution system design by Benders' decomposition. Manag Sci 5:822–854

Glover F, Klingman D, Stulz J (1974) Extensions of the augmented predecessor index method to generalized network problems. Transp Sci 7:377–384

Golden BL, Liberatore M, Lieberman C (1979) Models and solution techniques for cash flow management. Comput Oper Res 6:13–20

Johnson E (1966) Networks and basic solutions. Oper Res 14:89–95

Kennington JL, Helgason RV (1980) Algorithms for network programming. Wiley, New York

Lu H, Yao E, Qi L (2006) Some further results on minimum distribution cost flow problems. J Comb Optim 11:351–371

Mo J, Qi L, Wei Z (2005) A network simplex algorithm for simple manufacturing network model. J Ind Manag Optim 1:251–273

Toth P, Vigo D (eds) (2002) The vehicle routing problem. Society for Industrial and Applied Mathematics, Philadelphia