This discussion presents a unified method for organizing the configuration data of manufacturing files, and for generating and retrieving essential quantities from the files. The required processing operations, which include various requirements and engineering-change computations, are explained with the aid of matrix algebra.

An important objective of the method is to permit a reasonably optimal balance between the bulk-storage requirements and the amount of time required for processing.

Fabrication and assembly operations

Part III Matrix methods for processing configuration data

by P. G. Loewner

The manufacturing process of a complex product made up of various components includes a series of assembly and fabrication steps. In each step, several discrete parts are used in the construction of another part. Each part is identified by a unique number. Some parts, known as details, are purchased directly from other firms and can be considered raw materials. Parts constructed from other parts are known as assemblies. Those assemblies that represent the firm's final products are sometimes known as final assemblies; other assemblies represent intermediate steps in the manufacturing process and are therefore called subassemblies.

Strict control over inventories, schedules, costs, and other items of interest to management requires that information about the structure of a part be retrieved rapidly. For example, a sales forecast or a customer order generates a demand for final assemblies. This demand in turn generates a demand for subassemblies at each step in production. To compute the demand for parts at any step, complete structural information about each step of the manufacturing process is needed. On the other hand, if an alteration of a part is contemplated, it is essential to gauge in advance the effect of that change on the manufacturing process. To do this, it is

necessary to know every part that the assembly in question enters into. Hence, complete structural information is required.

This discussion describes a structural data processing method that is capable of performing various functions associated with the assembly of discrete parts. Among the functions are simple and computed references (simple and total bills of materials, where-used traces, detailed and accumulated costs, reorder points and quantities, etc.) and requirement computations (including netted, reorder, and time-dependent requirements). Consideration is given throughout to engineering changes.

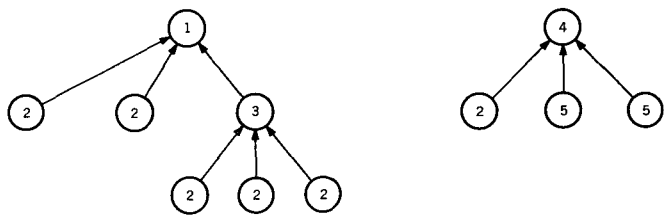
The matrix algebra approach to the bill of materials and other structural problems, based on the work of Vazsonyi,[1] is presented as introductory background information. The description of file organization and planned capabilities of the data processing system includes a definition of algorithms for each computation. The discussion is restricted to non-cyclic systems; i.e., systems in which no part is used at any level in the assembly of itself. Typical applications are shown for structural data references and requirement computations.

Although matrix algebra is used for descriptive purposes, matrices are not stored as regular arrays. List processing methods are used in the implementation to economize on storage space.

Matrix algebra approach

To illustrate the concept of structural information, consider a simple manufacturing process as represented by the tree in Figure 1. The process makes two products; the products are labelled part 1 and part 4. In order to manufacture part 1, a subassembly must be made first. Notice that there are two details (part 2 and part 5), one subassembly, and two final assemblies. The tree serves to describe not only the number of units needed to make a final assembly, but also the order in which they must be assembled.

Figure 1 Structure of a simple manufacturing process



An arrow in the tree will be called a *branch* and a circle a *node*. A node has one (and only one) directed path to a final assembly. If node b lies above node c in a given path, such that n branches intervene between c and b, then node c is said to be at *distance* n from b. Moreover, any node is at distance zero from itself.

next-assembly matrix

The information given in the tree can also be presented in a set of matrices. Obviously, in order to assemble any part, we must

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 5 & 1 & 3 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix}.$$

The diagonal has 1 in each entry, representing the fact that the zero distance is included in this matrix. In our example, $T = N^0 + N^1 + N^2$, and, in general,

$$T = \sum_{k=0}^{\delta} N^k = I + N + N^2 + \cdots + N^{\delta}, \tag{2}$$

where δ is the largest distance in the structure. Since

$$T = \sum_{k=0}^{\delta} N^k = I + \sum_{k=1}^{\delta} N^k$$

and, because $N^{\delta+1} = 0$,

$$N \cdot T = \sum_{k=1}^{\delta+1} N^k = \sum_{k=1}^{\delta} N^k,$$

then

$$T - N \cdot T = (I - N)T = I.$$

Hence

$$T = (I - N)^{-1}. \tag{3}$$

In practice, (2) is more useful than (3) as a way of solving for T.

triangularity

Another property of the N matrix is that it can be triangularized. In a triangular matrix, no non-zero entries exist on or above the main diagonal. An obvious way of triangularizing the matrix above is to permute rows and columns. If we define the *level* of a part as $d + 1$, where d is the maximum distance of the part from a final product, one way to obtain triangularity is to sort the part numbers on levels. Since parts 1 and 4 are final products, we put them first. Next we put part 3, and finally parts 2 and 5. Thus, we arrive at the triangular matrix N_T,

$$N_T = \begin{array}{c} \begin{matrix} 1 & 4 & 3 & 2 & 5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} 1 \\ 4 \\ 3, \\ 2 \\ 5 \end{matrix}$$

where part numbers are written on the top and side to indicate the column and row corresponding to them. The technique of sorting on levels is only one of the many ways of triangularizing N.

The N matrix is sparse—only a very small proportion of the entries are non-zero. This turns out to be a crucial factor in planning suitable processing methods.

typical matrix operations

Later we see that matrix multiplications of the form $V \cdot T$ and $T \cdot V$, where V is a vector variable, can yield vectors of information about each part. Assume, for the example depicted in Figure 1, an order for 5 units of part number 1, and 10 units of part number 4. We can write this demand as a column vector, say D, of the form

$$D = \begin{bmatrix} 5 \\ 0 \\ 0 \\ 10 \\ 0 \end{bmatrix},$$

where the demand for each part appears as a component in the vector. Since t_{ij} units of part i are needed to make one unit of part j, the total requirement for part i in fulfilling the demand d_j for part j is $t_{ij}d_j$. Hence, the total requirement r_i in fulfilling all demands is $\sum_j t_{ij}d_j$. A requirements vector R is thus specified by the matrix product $T \cdot D$.

In the above example, performing the multiplication gives

$$R = \begin{bmatrix} 5 \\ 35 \\ 5 \\ 10 \\ 20 \end{bmatrix}.$$

The vector R specifies the unit quantity of each part needed to meet the order. In addition to giving the number of each detail needed, it also yields the number of subassemblies required at each level of the assembly.

The requirements multiplication would be a simple matter if T could be kept in storage. Because T is much more dense than N, it is more feasible to store N then T. For this reason, we will later describe algorithms that perform the indicated multiplication using N rather than T.

File organization

The data files consist of three groups of data: the product structure file, the level file, and the parts data file.

product structure file

The product structure file (PSF) consists of the next-assembly quantity matrix N stored in two forms: ordered by columns, and ordered by rows. Both forms are kept in memory in order to facilitate rapid retrieval by row or by column. To take advantage of the sparseness of N, only the non-zero elements are stored. Each element is stored with a row or column index, as appropriate.

Moreover, a row or column begins with its own index. Thus, for row form, we store each row ${}_iN$ as

$$\{i, (j, n_{ij}) \mid n_{ij} \neq 0 \text{ for all } j\}.$$

If all $n_{ij} = 0$, the entire row, including its index, is omitted. Similarly, for column form, we store each column ${}^jN$ as

$$\{j, (i, n_{ij}) \mid n_{ij} \neq 0 \text{ for all } i\}.$$

The column and its index are omitted if all entries are zero. The compactness of this representation can be increased by making use of variable-length fields.

For the matrix obtained from Figure 1, the actual information stored would be: {2, (1, 2), (3, 3), (4, 1); 3, (1, 1); 5, (4, 2)} for N in row order, and {1, (2, 2), (3, 1); 3, (2, 3); 4, (2, 1), (5, 2)} for N in column order. Punctuation has been added only for purposes of illustration. The n_{ij} and its associated index have been grouped together by parentheses. A number standing by itself is a column or row index.

From the above discussion, it can be seen that, although matrix algebra is used to formalize the subject, the procedures are implemented with the aid of list processing methods. Since the matrices are large and very sparse, the elimination of zero elements sharply reduces the amount of memory required.

It is shown later that very efficient algorithms could be formulated by processing the N matrix if it were feasible to store the matrix in triangular form. The process of triangularization, which requires a great deal of sorting, might be justified if N did not change as frequently as it actually does. Changes that alter the level structure of the manufacturing process and, hence, change the triangular order of N, may occur frequently. To achieve some of the efficiency of triangularization without incurring a great deal of sorting, we work with a *semitriangular* N.

We define the row semitriangular matrix N_{RT} as an N matrix with its rows permuted such that, were the columns of N ordered by the same permutation, the resulting matrix would be a triangular matrix N_T. The column semitriangular matrix N_{CT} is the counterpart of N_{RT}, ordered by columns. For the matrix of Figure 1, we would have the following two matrices:

$$N_{RT} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{array}\right] \end{array} \begin{matrix} 1 \\ 4 \\ 3 \\ 2 \\ 5 \end{matrix} \qquad N_{CT} = \begin{array}{c} \begin{matrix} 1 & 4 & 3 & 2 & 5 \end{matrix} \\ \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{array}\right] \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix}.$$

Much sorting time is saved by working with a semitriangular matrix. A change to N that alters the level order of a part requires

a rearrangement of the columns of N_{CT} and the rows of N_{RT}. However, such a change does not require a rearrangement of the individual elements within the rows of N_{RT} or the columns of N_{CT}. It is shown later that the algorithms for N_T can be applied to N_{RT} or N_{CT} fairly easily. They are nearly as efficient when applied to the semitriangular matrix.

level file

The level file (LF) has a record for each level of the manufacturing process, listing all parts that enter the manufacture at that level for the first time. The top level, the final product level, is designated as level 1. The levels preceding the top level are numbered two, three, etc. For the manufacturing process of Figure 1, the file is shown in Table 1. In this unique-level system, detail number 2 is considered in level 3 only, although it also enters the assembly process at the second level: it is assigned to level 3 because it enters that level first. The parts in Table 1, said to be in *level order*, are ordered by level, and within level by part number. Methods of level computation are discussed later.

Table 1 Example of level-file order

Level	*Parts*
1	1, 4
2	3, 5
3	2

The purpose of the level file is to make the N matrix effectively semitriangular, even though both of the forms in which it is stored are in haphazard order. For example, if it is necessary to perform some operation with N_{RT}, by consulting the level file one can select the rows of N in level order and thus effectively perform the operation as if N were semitriangular. The same argument holds for an operation with N_{CT}.

parts data file

Associated with each part is a list of numbers describing important characteristics of the part. The lists for all parts are known jointly as the parts data file (PDF). For each part i, the PDF has a record with the items listed in Table 2 (other items may also appear). In this table, the references to rows and columns are an aid to locating a desired row or column. If the files are kept on

Table 2 Typical items in a parts data file

c_i	*Local cost.* Price of purchases for details, or one-level fabrication cost for assemblies. The one-level fabrication cost includes only the cost incurred in putting the immediate components together to make a part, and does not include the cost of the components.
s_i	*Standard labor element.* Amount of labor time required to assemble a part from its immediate components.
b_i	*Lead time.* Amount of time that elapses between the start of manufacture of a part and the availability of that part (within one level).
a_i	*Availability.* Number of units of part i available in stock.
k_i	*Reorder point.* Inventory threshold at which a reorder for part i should be placed.
q_i	*Reorder quantity.* Minimum number of units of part i that should be reordered when $a_i \leq k_i$.
$\chi_{(i)}$	*Part identification.* The manufacturer's identifier for part i; this is stored in PDF for reference purposes.
$r(R)_i$	*Reference to row.* Address of the stored row of N corresponding to part i.
$r(C)_i$	*Reference to column.* Address of the stored column of N corresponding to part i.

disk storage, the reference is a disk address, and both representations of N can be stored in random order. However, if the files are kept on tape, the only way of referencing N in triangular order is to store its column-by-column representation in semitriangular order of columns and its row-by-row representation in semitriangular order of rows, thus minimizing the number of tape passes when searching. In this case, the reference might be the record number on the tape.

It is useful to visualize each class of information in Table 2 as a vector with an entry for each part. Thus the vector C of detailed costs, given as

$$C = [c_1, c_2, c_3, \cdots, c_n],$$

where n is the total number of parts in the system, has a component for each part i. In this respect, PDF can be thought of as a set of vectors, each representing different characteristics for the parts. The information for each part is stored as a list.

System capabilities

A configuration data processor may be envisioned as a group of programs that are adaptable to a wide range of application environments. The underlying objective is to provide a set of programs from which the user can select as required by his special needs. The number of functional programs should be as small as possible without sacrificing the versatility of the system. The computational system is capable of quite different kinds of structural data processing: file maintenance (engineering changes), and basic computational retrieval functions.

engineering changes

Up to this point, we have discussed a manufacturing process as though it were represented by a single N. In practice, however, engineering changes force one to consider multiple N's. From its conception as a proposed product to its manufacture, a product undergoes numerous changes at the design development, testing, and manufacturing stages; in fact, products are sometimes released into the market while still undergoing changes. Each change, in effect, creates a new process that has a distinct N associated with it.

Engineering changes may also occur in satisfying a demand for custom-made products; plants producing large and costly products often tailor standard designs to customer specifications. Hence each article is described by a distinct N. Each of the N's may be recorded and retained because maintenance, repair, and modifications demand that the structural data of any product be known in exact detail.

To economize space, we can store the various N's as either pivotal or delta matrices. A pivotal N is simply a complete N that serves as a reference point in a stream of engineering changes, whereas a ΔN is the set of changes that must be applied to the previous stream to produce an updated N. To retrieve N, it may therefore be necessary to go through a series of delta N's. In some circumstances, it is desirable to permit a stream of changes to

diverge; if this is permitted, a particular desired N can be obtained by following a path in a tree of changes.

As the chain progresses, pivotal matrices may be stored as necessary to achieve a balance between storage economy and processing time. This procedure also applies to the parts data file and the level file. For each engineering change, the PDF, PSF, and LF are given identifying header records. These records specify the effective date of the change and/or its serial number; other descriptive information may also be included.

As an example of an engineering change, assume that we wish to change the matrix representing Figure 1 as follows:

$$N = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix} \quad \text{to} \quad M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 3 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix}.$$

The only change is in $n_{2,1}$ which is changed to 3. Thus, we would store ΔN as $\{2, (1, 3)\}$. Deletions can be made by entering the row and column index with a zero element.

In this system, new rows and columns can be added by entering them in the previously defined form. Since the row and column indices are not in the N being modified, the rows and columns are added without replacing any indices. Therefore, files are automatically created by applying an engineering change, together with the initial files, to a set of files containing only zero elements. The zero files do not take much space because of our file organization scheme.

This procedure applies to the parts data file as well. With the level file, however, we have a somewhat different problem. The level file has a list of parts for each level. When a change is made to a level, that change consists of the insertion or deletion of a part. Thus, we indicate a change by listing the level, followed by the list of parts, each with a positive or negative sign indicating additions or removals, respectively. For example, consider that we want to change the level file given in Table 1. Assume that part number 3 is removed from level 2 and placed in level 3, and that a new part (part number 6) is added to level 2. The ΔLF is shown in Table 3.

Table 3 Example of a delta level file

Level	*Parts*
2	−3, 6
3	3

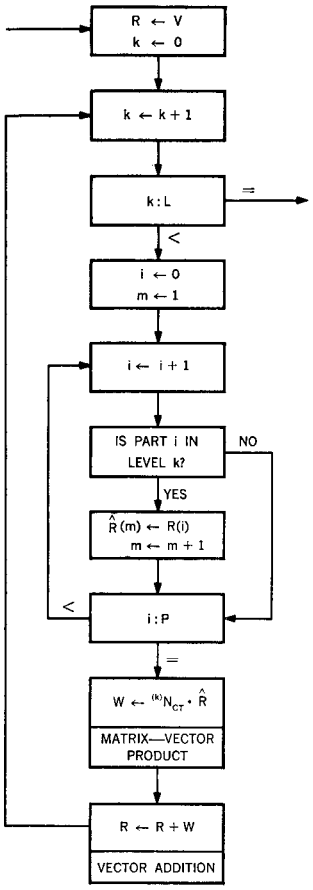
basic functions

The basic functions in the method are dependent upon the matrix operations $V \cdot N$ and $N \cdot V$, where V is a row or column vector as dictated by compatibility considerations. Given N stored by rows, it is obvious how to perform a multiplication $R = N \cdot V$ by operations of the form

$$r_i = \sum_j n_{ij} v_j. \tag{4}$$

If an entire row of N is zero, we can skip that step and set r_i to zero. Given N stored by columns, we can initially set the vector R

Figure 2 Flow chart for obtaining R = T • V



EXPLANATION	
L	NUMBER OF LEVELS
P	NUMBER OF PARTS
M[k]	NUMBER OF PARTS IN LEVEL k
i	PART INDEX
k	LEVEL INDEX
m	WITHIN-LEVEL PART INDEX
V	INPUT VECTOR
R(i)	ith ELEMENT OF R VECTOR
R̂(m)	mth ELEMENT OF R̂ VECTOR
$^{(k)}N_{CT}$	MATRIX OF M[k] COLUMNS AND P ROWS CONSISTING OF THE COLUMNS OF N_{CT} IN LEVEL k.

to zero and iterate the following equation,

$$R = R + {}^{i}N\, v_i \qquad i = 1, 2, \cdots, m, \tag{5}$$

where ${}^{i}N$ is the ith column of N, v_i is the ith element of V and m is the dimension of V. This step can be omitted whenever ${}^{i}N = 0$. This method can be very fast if V has many zero elements. Notice that whenever $v_i = 0$, we may also omit the step. Thus, the sparseness of both N and V serves to hasten the computation. Similar reasoning applies to $R = V \cdot N$.

A second useful pair of functions are $V \cdot T$ and $T \cdot V$. To perform the indicated multiplication by using N rather than T and making only one pass over N, a semitriangular N is applied. The use of N_{CT} or N_{RT} creates a few problems, since the order of the rows in the above matrices is in general not the same as the order of the columns. If we want to perform the multiplication $N_{RT} \cdot V$, then V must be in the same order as the columns of N_{RT} and, therefore, in natural (or customer) order. Note that the resulting vector is in the order of the rows of N_{RT} . To avoid rearranging the result vector, the following algorithm is used.

Let ${}^{(k)}N_{CT}$ be the N_{CT} matrix with all columns not in level k replaced by zeros. Let $R^{(k)}$ be a column vector with all entries not in level k replaced by zeros. Let $\hat{R}$ denote R ordered in level order, L be the number of levels, and set R_1 equal to V in the following recursion equation (see Figure 2 for a flow-charted explanation).

$$\left.\begin{aligned} &\text{Generate } \hat{R}_k^{(k)} \text{ from } R_k \\ &R_{k+1} = R_k + {}^{(k)}N_{CT}\hat{R}_k^{(k)} \end{aligned}\right\} \quad k = 1, 2, \cdots, L-1. \tag{6}$$

We can generate $\hat{R}^{(k+1)}$ at the same time we generate R_{k+1} by testing each element of R_{k+1} , as it is produced, to determine whether it belongs in level $k + 1$. The last R_{k+1} is the desired vector R . This algorithm works very well if Equation 5 is used and the columns of N are ordered within each level in the same order as the rows. An equivalent algorithm can be formulated for $R = V \cdot T$ by letting ${}_{(k)}N_{RT}$ be the N_{RT} matrix with all rows not on the kth level replaced by zeros.

It is interesting to note that ${}^{(k)}N_{CT}$ can be replaced by ${}_{(k)}N_{RT}$, and vice versa, provided that the placement of the circumflex is suitably rearranged.

checking of results

One of the advantages of matrix methods for solving bill-of-material problems is that one can check the results by back solutions. From Equation 3, $T \cdot (I - N) = I$. It follows that the general algorithm $R = T \cdot V$, using N, has the back solution:

$$R = V + N \cdot R.$$

To speed up checking, one can collapse the matrix N by using only its checksums: row sums, column sums, and total sum. To describe the checking formally, let $\bar{U}$ be a row or column vector consisting of only 1's. Then $\bar{U} \cdot N$ and $N \cdot \bar{U}$ represent the two

checksum vectors of N, and $\bar{U} \cdot N \cdot \bar{U}$ represents the total checksum. The general algorithm is checked by the identity

$$\bar{U} \cdot R = \bar{U} \cdot V + (\bar{U} \cdot N) \cdot R.$$

The other computations are checked by similar identities.

Structural data references

computed references

The structural data references are sets of information that can be obtained from the given files. Simple references, such as those cited in Table 4, are directly retrievable from the files.

Table 4 Directly retrievable file references

Reference	*Notation*	*Remarks*
Bill of Materials	${}^{i}N$	The B/M for a part is the corresponding column of N.
Where-Used Trace	${}_{i}N$	The W/U trace for a part is the corresponding row of N.
Boolean B/M	${}^{i}\bar{N}$	The Boolean or skeleton B/M is the corresponding column of N with only 0 and 1 entries.
Boolean W/U Trace	${}_{i}\bar{N}$	The Boolean or skeleton W/U trace of a part is the corresponding row of N with only 0 and 1 entries.
Detailed Cost	c_i	Directly obtainable from PDF.
Standard Labor	s_i	Directly obtainable from PDF.
Lead Time	b_i	Directly obtainable from PDF.
Reorder Point	k_i	Directly obtainable from PDF.
Reorder Quantity	q_i	Directly obtainable from PDF.
Basic Levels		There are three basic levels; details, subassemblies, and final assemblies. The basic level of a part can be obtained by searching the row and column reference in PDF. If the part has a row, but no column in N, it is a detail. If it has a column, but no row, it is a top assembly. If it has both, it is a subassembly.

The following references require some computation:

Total Bill of Materials. This reference is column ${}^{i}T$ of T and can be computed from N by letting U be a column vector consisting of 1 in the ith place and 0's elsewhere. Then

$${}^{i}T = T \cdot U,$$

in which the algorithm for $T \cdot V$ described previously can be used.

Total Where-Used Trace. This reference is the corresponding row of T. It can be computed by an algorithm equivalent to the preceding one, where U now represents a row vector with 1 in the ith place and 0's elsewhwere. Then

$${}_{i}T = U \cdot T,$$

for which the algorithm for $V \cdot T$ can be used.

Total Boolean Bill of Materials. From a column ${}^{i}T$, this reference yields a vector ${}^{i}\bar{T}$ in which each entry is 0 or 1 when t_{ij} is zero or non-zero, respectively.

Total Boolean Where-Used Trace. This reference is similar to the previous one except that it refers to a row of T.

Selective Reference. The selective reference for a part i is the bill of materials for that part, exploded level by level for each part used in the manufacture of i. It is a useful compromise that often avoids the need for a complete tree, which is tedious to generate and bulky in volume. Thus, the selective reference for part i in Figure 1 is:

① uses two ②, one ③;
③ uses three ②.

No entry is needed for part 2, because it is a detail and uses no parts. The selective reference for part 4 is:

④ uses one ② and two ⑤

To obtain a selective reference, we list the columns of N indicated by ${}^{i}\bar{T}$. To obtain a selective reference for several parts and for a restricted number of levels, we generate a Boolean vector $\bar{V}$ to identify the parts wanted. Then we obtain $\bar{T}$ from $\bar{N}$ by using a Boolean version of (6), carried out to L iterations only, where L is the number of levels involved.

Accumulated Cost. This reference yields the total cost of a part, including the cost of details and fabrication. If C is the vector of detailed costs, the vector of accumulated cost, V, is found by $V = C \cdot T$. Since c_i is the cost of part i, and v_j the accumulated cost of part j, we have

$$v_j = \sum_i c_i t_{ij},$$

and therefore

$$V = C \cdot T.$$

Accumulated Labor Element. If S denotes a vector of labor elements, the accumulated labor element vector H is given by $S \cdot T$. The derivation is similar to the one for accumulated cost.

Level Identification. There can be various definitions of level; in many contexts, a part is not assigned a unique level. However, for our purposes, we consider only one unique-level definition. The level of a part is determined by its position in the level file.

unique level computation

The algorithms for the unique-level computation follow: let $\bar{U}_0$ be a column vector of all 1's, then

$$\bar{U}_{k+1} = \bar{N} \cdot \bar{U}_k \quad \text{for} \quad k = 1, 2, \cdots \text{until } \bar{U}_{k+1} = 0.$$

Letting the last k = L,

$$V = \sum_{k=0}^{L} \bar{U}_k,$$

where the summation is non-Boolean. The resulting vector V has an entry for each part, listing its level in terms of the longest path from any final product.

At engineering-change time, levels can be recomputed by using a faster algorithm that takes advantage of delta matrices, wherever possible, to avoid unnecessary scanning of the complete N.

Requirements computations

The requirements computation makes it possible to determine the quantity of each part that must be purchased or manufactured to meet a given demand. It is important to understand that the demand may be for any part, although a demand for details is not likely. Typically, in order to reach a wider market, a manufacturing concern sells subassemblies as well as final products.

We define the demand vector D in such a manner that d_i is the number of units of part i in demand. Of special interest are four functions: simple, edited, reorder, and time-dependent requirements.

For simple requirements,

$$R = T \cdot D,$$

simple requirements

where the result vector R is so defined that r_i is the total number of units of part i that must be purchased (for details) or manufactured (for assemblies) to meet the demand given by D. The simple requirement function gives a total for each part.

edited requirements

If parts are available in stock, the requirements must be "netted" (adjusted against stock); whenever the availability of an assembly is netted against the requirement vector, the requirements for parts at lower levels change. The object of edited requirements is to use as many items as possible from available stock, using the assemblies of lower level number first.

In the following algorithm for the semitriangular N, we define D as the demand vector, A as the availability vector, A^* as the edited stock availability vector, R as the edited requirements vector, L as the number of levels, and m_i denotes the number of parts in level i. For any of these vectors, let $\hat{V}$ be V reordered by levels (to correspond to semitriangular order), but otherwise remain in natural order. Also, let $V^{(k)}$ be V with all entries not in level k replaced by zeros. Let ${}_{(k)}N_{RT}$ be the N_{RT} matrix in which, except for level k, all rows have been replaced by zeros.

Step 1. For initialization, set $i = 1$ and $R = 0$, and obtain
$\hat{F} = \hat{D} - \hat{A}$

Step 2. $\hat{E}^{(i)} = \hat{F}^{(i)} + {}_{(i)}N_{RT} \cdot R.$

Step 3. Letting $e_j^{(i)}$ represent the jth element
$(j = 1, 2, \cdots, m_i)$ in $\hat{E}^{(i)}$, the ith level of $\hat{E}$,

$$\hat{r}_j^{(i)} = e_j^{(i)} \quad \text{and} \quad \hat{a}_j^{*(i)} = 0 \quad \text{for} \quad e_j^{(i)} \geq 0$$

$$\hat{r}_j^{(i)} = 0 \quad \text{and} \quad \hat{a}_j^{*(i)} = -e_j^{(i)} \quad \text{for} \quad e_j^{(i)} < 0.$$

Step 4. If $i < L$, increase i by 1 and continue with step 2.
If $i = L$, the algorithm is completed, giving R and $\hat{A}^*$ as the computed results.

In step 3, we form $\hat{R}$ and $\hat{A}^*$, i.e., the resulting vectors are in level order. In step 2, we need R in natural order. Since the order of elements in $\hat{R}^{(i)}$ is the same as the order in $R^{(i)}$, we simply form R from $R^{(i)}$ by passing over R once and inserting $r_j^{(i)}$ into the proper places of R. R actually expands as it is computed. At the ith stage, R contains only elements in levels 1 to i, and no elements in levels $i + 1$ to L. At each level, we compute m_i elements. Thus

$\sum_{i=1}^{L} m_i = P$ where P denotes the total number of parts.

reorder requirements

The reorder requirements computation function, a refinement of the edited requirements computation, is designed to keep stock availability from falling below the reorder point (wherever possible) and to generate orders when the reorder point is reached. Letting K denote the vector of reorder points, and Q the vector of reorder quantities, the algorithm for simple edited requirements should be modified as follows: In step 1:

$$\hat{F} = \hat{D} - (\hat{A} - \hat{K}) = \hat{D} - \hat{A} + \hat{K}$$

Replace step 3 with the following ($j = 1, 2, \cdots, m_i$):

$$\hat{r}_j^{(i)} = 0 \quad \text{for} \quad e_j^{(i)} \leq 0$$

$$\hat{r}_j^{(i)} = \hat{q}_j^{(i)} \quad \text{for} \quad 0 < e_j^{(i)} \leq \hat{q}_j^{(i)}$$

$$\hat{r}_j^{(i)} = e_j^{(i)} \quad \text{for} \quad \hat{q}_j^{(i)} < e_j^{(i)}$$

$$\hat{a}_j^{*(i)} = \max (\hat{k}_j^{(i)} - e_j^{(i)}, 0).$$

time-dependent requirements

The time-dependent requirement, a refinement of the simple requirement, considers the lead time of each part. This time, b_i, represents the number of time intervals required for part i from start to availability. Consequently, the demand D is now a matrix whose elements $d_{i\tau}$ refer to the demand for part i at time τ. We define G as a matrix containing elements $g_{i\tau}$ representing the quantity of part i that must be available at time τ. Let R be a requisition matrix whose elements $r_{i\tau}$ represent the number of parts i on which we must start working at time τ. Then it is quite obvious that

$$g_{i\tau} = r_{i,\tau-b_i}$$

and

$$r_{i\tau} = g_{i,\tau+b_i}.$$

The computation required to calculate G or R is quite formidable. However, a simplification can be attained if we assume unique

lead times, i.e., we ignore the time necessary to attach each individual part to the subassembly.[2] Let B denote a vector of lead times, where b_i is the time required to obtain part i from its immediate predecessors. Considering only the total time required to produce that subassembly,

$$g_{i\tau} = d_{i\tau} + \sum_{j=1}^{p} n_{ij} r_{i\tau},$$

so that

$$g_{i\tau} = d_{i\tau} + \sum_{j=1}^{p} n_{ij} g_{j,\tau+b_j},$$

which may be written as

$$G^{\tau} = D^{\tau} + N \cdot G^{\tau+B},$$

where $G^{\tau+B}$ is a symbolic representation of $g_{j\tau+b_j}$ for all j. Similarly,

$$r_{i\tau} = g_{i,\tau+b_i} = d_{i,\tau+b_i} + \sum_{j=1}^{p} n_{ij} r_{j,\tau+b_i}.$$

This may be written as

$$R^{\tau} = D^{\tau+B'} + N \cdot R^{\tau+B'},$$

where b_j' is defined as b_i when the ith row of N is used.

In calculating $g_{i\tau}$ or $r_{i\tau}$ for details and subassemblies, we must consider not only $d_{i\tau}$ or $d_{i,\tau+b_i}$ but also the future requirements for a part as determined by the future demand for all higher-level assemblies containing the part. A more accurate description of the steps needed to obtain the time dependent requirement is now presented. Included is a method for determining the length of future time that must be considered.

Let $\bar{U}$ be a Boolean selection vector with 1's in the position of parts whose requirement we wish to determine. Using the Boolean version of the algorithm for semitriangular $\bar{N}$ described before, calculate $\bar{V} = \bar{U} \cdot \bar{T}$. This step can be omitted if we are interested in all parts, in which case $\bar{V}$ consists of all 1's. For all parts indicated by $\bar{V}$, compute h_j , which refers to the planning horizon for each part j. (For details, $h_j = b_j$.) Then, continuing level by level, set

$$h_j = b_j + \max_{n_{ij} \neq 0} \{h_i\}.$$

Where applicable, a simpler method is to specify h_j as input. Again, for all parts indicated by $\bar{V}$, set

$$r_{i\tau} = d_{i,\tau+b_i}.$$

This can be done by shifting the rows of the D matrix to the left. For final products, the answer consists of the shifted D. For all other parts we must continue, in order of their levels, as follows: for each non-zero n_{ij} , shift the jth row of the R matrix b_i units to the left, multiply it by n_{ij} , and add this to the ith row of the R matrix.

Figure 3 Structural example including lead times

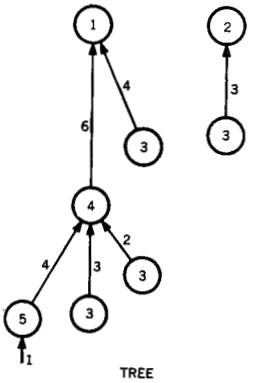


Table 5 Demand matrix D

	τ										
i	1	2	3	4	5	6	7	8	9	10	11
1	2	2	2	3	3	3	2	2	2	2	2
2	1	1	2	2	2	1	1	1	1	0	0
3	2	2	2	2	2	2	2	1	1	1	1
4	0	0	0	0	1	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0	0	0	0

Consider the example in Figure 3 for which the demand matrix is shown in Table 5. Then

$$N = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 3 & 1 & 1 & 2 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

To compute r_{31} , we generate $\bar{U} = (0, 0, 1, 0, 0)$ and then compute $\bar{V} = \bar{U} \cdot \bar{T} = (1, 1, 1, 1, 0)$ (Boolean multiplication). Thus, we can ignore part 5.

We now compute h_1 , h_2 , h_3 , h_4 . For details,

$h_3 = b_3 = 0$. For subassemblies, compute h_4 :

$$h_4 = b_4 + \max\{h_3\} = 4 + 0 = 4.$$

For final products, we compute h_1 and h_2 :

$$h_1 = b_1 + \max\{h_3 , h_4\} = 6 + \max\{0, 4\} = 10$$

$$h_2 = b_2 + \max\{h_3\} = 3 + 0 = 3.$$

Thus, we have the values for b_i and h_i indicated in Table 6. We are interested only in the part of D shown in Table 7. When shifting by b_i , we receive the result shown in Table 8.

For the next level, we compute part 4, for which $n_{41} = 1$, as in Table 9. We shift the first row left $b_4 = 4$ places, and add to the fourth row, leading to the result of Table 10. Next, we compute part 3, for which $n_{31} = 1$ and $n_{32} = 2$. We shift rows 1, 2, 4 left by $b_3 = 0$, multiply by n_{31} , and add to row 3, receiving the result of Table 11. Thus, $r_{31} = 12$.

Simple requirements could be computed, in this method, by making $b_i = h_i = 0$. Thus, the simple requirement is a special case of the time-dependent requirement. The edited and reorder requirements can also be generalized as time-dependent computations.

Concluding remarks

The discussion shows how to obtain various quantities from the next-assembly matrix N. Minimal, although appreciable, use is made of bulk storage. The N matrix is stored in two forms. At some loss of speed, memory space can be approximately halved if only one of the matrices is used.

In an alternative method for solving bill-of-materials problems, the total-requirement matrix T is also saved. By this method, the retrieval and requirement computations are greatly speeded up and simplified. The general algorithm is not used. However, bulk storage requirements would be considerably increased, and engineering changes more awkwardly accomplished, because ΔT is needed in addition to ΔN. A possible way of using ΔT is shown by Gleiberman.[3]

Table 6 Lead times and planning horizons

i	1	2	3	4
b_i	6	3	0	4
h_i	10	3	0	4

Table 7 Relevant portions of the demand matrix

	τ										
i	1	2	3	4	5	6	7	8	9	10	11
1							2	2	2	2	2
2			2								
3	2										
4				1							

Table 8 Shifted portions of the demand matrix

	τ				
i	1	2	3	4	5
1	2	2	2	2	2
2	2				
3	2				
4	1				

Table 9

	τ
i	1
1	2
4	1
new 4	3

Table 10

	τ
i	1
1	2
2	2
3	2
4	3

Table 11

n_{3i}	i	$\tau = 1$	
1	1	2	
1	2	2	
—	3	2	
2	4	6	(3×2)
new	3	12	

Random access bulk storage, provided by magnetic disks, is well suited to this application. However, tapes are also feasible for certain application environments, provided the semitriangular matrices are stored in their level order to allow one-pass computations.

ACKNOWLEDGMENT

Parts of this report are based on unpublished work by L. Gleiberman. The author is also greatly indebted to the contributions of B. Ocampo and D. Berger, and to insights obtained from unpublished work by E. Homer.

CITED REFERENCES AND FOOTNOTE

1. A. Vazsonyi, "The use of mathematics in production and inventory control," *Management Science* **1**, No. 1 (Oct. 1954).
2. The unique lead-time restriction can be removed if one is willing to add some dummy parts to the N matrix.
3. L. Gleiberman, "The engineering change of the total requirement matrix for a bill of materials," *Management Science* **10**, No. 3, 488–493 (April 1964).