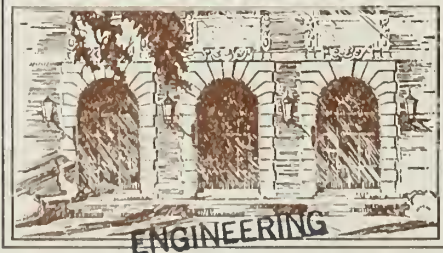


LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

510.84

IL63c

no. 1-10



AUG 5 1976

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN


ENGINEERING

CONFERENCE ROOM

JUL 07 REC'D

AUG 04 1983

JUL 15 REC'D



Digitized by the Internet Archive
in 2012 with funding from
University of Illinois Urbana-Champaign

<http://archive.org/details/matrixgenerators00marc>

CONFERENCE ROOM

ENGINEERING LIBRARY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC DOCUMENT NO. 4

A MATRIX-GENERATOR SYSTEM FOR
LINEAR PROGRAMMING PROBLEMS

by

Ian W. Marceau

and

Thomas W. Mason

June 30, 1971

A MATRIX-GENERATOR SYSTEM FOR
LINEAR PROGRAMMING PROBLEMS

by

Ian W. Marceau
and
Thomas W. Mason

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

June 30, 1971

ABSTRACT

A matrix-generator language for use in structuring and inputting linear programming problem matrices is described. The generator is based on the definition of the geometric sub-arrays which occur within a sparse linear programming matrix. An exhaustive list of sub-array structures is given, and their use within the described generator language is demonstrated by the presentation of a real-world agricultural planning problem. Estimates of the cost-reducing potential of this generator system are given.

TABLE OF CONTENTS

	Page
INTRODUCTION	1
COST EFFECTIVENESS OF A MATRIX GENERATOR	2
MATRIX GENERATOR LANGUAGE	6
Examples of Data Structures	8
Use of the Data Structures	11
EXAMPLE MODEL	14
The Tableau	14
The Matrix Generator Statements	20
Figure 1	21
Figure 2	23
Figure 3	24
Figure 4	24
Figure 5	25
CONCLUSION	27
REFERENCES	28

INTRODUCTION

The major costs of linear programming are incurred in three areas: model formulation and input to the computer, data collection and collation, and interpretation of solutions. The costs incurred in the first and third of these areas can be significantly reduced by the use of matrix-generation and associated report-generation procedures. The use of these generators goes hand-in-hand with generalized modeling techniques. With these techniques one can formulate a set of generic model structures which can be applied to a wide range of specific problems.

The conventional approach to linear programming is to formulate a model for each problem, write out the list of coefficients in SHARE standard format (column I.D., row I.D., value), keypunch each coefficient, and read the data from the resultant cards. For any but very simple problems, this is an extremely expensive and error-prone process.

In cases where given model structures are used repeatedly, as in commercial applications of linear programming, some of the problems of model specification, input and output have been overcome by the use of matrix generators and report generators. These generators are usually designed for the model structures used, and are not sufficiently general to allow major changes in model structure to be made easily. Such model-specific generators, despite their lack of generality, result in significant reductions in the costs of entering models to the computer. For example, a generator developed by Marceau et. al. for application in farm planning reduced the input requirement for the average individual problem model from 5000 cards to 70 cards [1].

The approach to matrix generation described in this paper is one of complete generality. The generator takes advantage of the sub-array structures of a sparse matrix, and allows the user to describe the matrix structure in terms of these arrays. This approach overcomes the problems arising in the use of model-specific generators. An additional advantage is that the user simply needs to define his model structures in generic terms (i.e., by activity type) and then reproduce them by the use of matrix-generator parameters.

These mn coefficients (excluding the right-hand sides) could be defined using the matrix generator with one statement:

DIAGONAL(m) : 1 * n.

With this statement an m by m sub-matrix whose coefficients are 1 on the diagonal and 0 elsewhere is duplicated n times within the total matrix structure.

In general, using a matrix generator yields three types of savings. First, there is the saving in time and money required for keypunching. This saving, of course, is due to the lesser number of cards required for the matrix generator and can be expressed as

$$C_k \left(\frac{N}{2} - N_m \right),$$

where C_k is the cost of keypunching one card (under the assumption that it costs the same to punch a card for the conventional system as for the matrix generator),

N is the number of data items, i.e., twice the number of cards required for the conventional system, and

N_m is the corresponding number of cards for the matrix generator.

Second, there is the saving realized in getting the data into the computer. This is the difference between reading fewer cards and internally generating the values versus reading a larger deck of cards and extracting all data from this deck, and can be expressed as:

$$C_c \left(T \frac{N}{2} - (T + et) N_m \right)$$

where C_c is the cost of computer time,

T is the time to read a card and scan the information,

e is N/N_m ; i.e., the average number of elements generated per matrix generator statement, and

t is the time to internally generate one value.

Noting that $e = N/N_m$ yields a simplified expression of

$$C_c \left[(T - 2t) \frac{N}{2} - TN_m \right].$$

Since T will normally be measured in milliseconds and t will be performed in microseconds, T-t is approximately equal to T. Thus the saving realized in getting the data into the computer is

$$C_r \left(\frac{N}{2} - N_m \right)$$

where C_r is $C_c T$, the cost of reading a card.

The third form of saving is not as easy to quantize. This is the saving in analyst's time in going from an idea of a model to implementation and calculation on the computer. The primary component of this saving is due to the saving in keypunch time. However, the monetary reward is related to the increased efficiency of the analyst due to the enhancement of the feedback of results to the model builder. An additional saving is the reduction in analyst's time necessary to prepare the forms for keypunching.

Considering only the first two types of savings in the above discussion,

$$S = (C_k + C_r) \left(\frac{N}{2} - N_m \right)$$

where S is the total saving due to use of the matrix generator. Replacing the sum of keypunching and reading a card, $C_k + C_r$, by C yields

$$S = C \left(\frac{N}{2} - \frac{N}{e} \right), \quad \text{or}$$

$$S = C \frac{N}{2} \left(\frac{e-2}{e} \right).$$

Note that $C \frac{N}{2}$ equals the total cost of keypunching and entering data by conventional means. Tests of the matrix generator have shown that a value for e of 50 is not unreasonable. Hence, the equation above shows that for this value of e, the saving produced by the matrix generator is 96% of the cost that would be incurred with the conventional system. That is to say, the cost of the matrix generator approach is 4% of the conventional system cost.

The remaining question is whether this saving is significant. Considering a keypunching cost of \$6.00 per 100 cards and a data field width of half a card, one arrives at a value for C_k of \$0.03. An analysis of the cost of reading a card, C_r (under the assumption of using magnetic tape with the B5500) yielded a value of \$0.0003. Since C_k is two orders of magnitude greater than C_r , we shall consider the total cost, C , to be equal to C_k . Thus for a tableau of 400,000 nonzero data items, the cost of inputting data by conventional means is \$12000.00. This cost incurred for a matrix generator with an average efficiency, e , of 50 is 4% of the conventional cost or \$480.00. It should be noted that C was conservatively estimated, and thus the total cost for inputting the data is probably underestimated. In addition, the value of e will tend to increase as the size of the model increases. Hence, the attractiveness of using a matrix generator increases with the size of the model.

MATRIX GENERATOR LANGUAGE

Many classes of linear programming problems can be formulated in generic terms, requiring the definition of only one of each type of activity occurring in the model. For example, in an agricultural model, the structures of the vectors defining the production of multiple crops are identical so the model builder needs to define the structure only once. Additionally, activity and constraint definitions can be made in such a way that most of the nonzero matrix entries are unity, and therefore constant for all problems of the class specified. Individual problem matrices can be created by naming the specific activities required and augmenting the matrix skeleton with technical coefficients, objective function values and constraint levels pertaining to the problem.

A linear programming matrix can be regarded as consisting of a number of sub-arrays created by a small number of geometric structures. In the language described herein, there are twelve elementary structures, six of which have mirror images, giving a total of eighteen structures. A right mirror image is indicated by "(R)" prefacing the name of the elementary structure. The geometric structures and their generator names are listed in Table 1. The term "value" refers to only nonzero coefficients.

The generator statement defining each sub-array has the form:

<structure (dimension): value list * repeat factor> .

Structures are either singly or doubly dimensioned. In the latter case, the first integer in the parentheses specifies the number of rows in the structure, and the second, the number of columns (except in the case of BANDS, where the second refers to the number of diagonals in the structure). The value list is either simple or compound. A simple value list is a single value which is used for all elements in the structure. A compound value list is a series of values (separated by commas) which are used in accordance with the rules of the structure. For example, the DIAGONAL structure accepts a compound value list and will use the first value in the list as the first diagonal element, the second value in the list as the second diagonal element, etc. Whenever a structure accepts a compound value list and the list contains fewer values than are specified by the

Table 1

Geometric Structures and Generator Names

Structure	Generator Name
1. point, or single value	POINT
2. row	ROW
3. column	COLUMN
4. diagonal	DIAGONAL
5. band matrix: values above and below main diagonal	(R) BAND
6. lower band matrix: values only below main diagonal	(R) LOBAND
7. upper band matrix: values only above main diagonal	(R) HIBAND
8. upper triangular matrix	(R) HITRI
9. lower triangular matrix	(R) LOTRI
10. parallelogram	(R) PARM
11. rectangle	RECTANGLE
12. square	SQUARE

dimension of the structure, the last value in the list is used to provide the missing terms. Thus, if a value list of 1,2,3 was supplied to a DIAGONAL (5) structure, the resulting sub-matrix would have diagonal elements of 1,2,3,3,3. Decimals may be written without a prefacing zero (i.e., both .5 and 0.5 yield the same value) and integers may be written without a zero decimal part (i.e., both 1 and 1.0 yield the same value). The value list may be input on cards, as in the examples of the data structure below, or may be contained in a file identified in a structural statement of the form:

<structure (dimension): file name (file location)> :

where the file from which the values are read may be cards, tape, or disk.

The repeat factor is an integer and specifies the total number of contiguous structures to be horizontally strewn. For example, a declaration of LOTRI(2) : 1 * 3 will produce 3 lower triangular matrices.

A characterization of the eighteen different structures by dimension and value types is shown in Table 2. Specific examples for each type follow. Although only integers are used to display sample structures, reals are equally acceptable to the matrix generator code.

Table 2
Characterization of Data Structures

	Values		
	Simple	Compound	
Dimension	Single	POINT, SQUARE, (R)LOTRI, (R)HITRI	ROW, COLUMN, DIAGONAL
	Double	RECTANGLE	(R)BAND, (R)LOBAND, (R)HIBAND (R)PARM

Examples of Data Structures

1. Single dimension, simple value list:

POINT : 1 * 5
11111

SQUARE(3) : 2
222
222
222

LOTRI(4) : 1 * 2
1 1
11 11
111 111
11111111

RLOTRI(4) : 1 * 2
1 1
11 11
111 111
11111111

HITRI(4) : 1 * 2
11111111
111 111
11 11
1 1

RHITRI(4) : 1 * 2
11111111
111 111
11 11
1 1

2. Single dimension, compound value list:

ROW(5) : 1,2,3 * 2
1233312333

COLUMN(5) : 1,2,3 * 2
11
22
33
33
33

DIAGONAL(5) : 1,2,3,4,5 * 3
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5

3. Double dimension, simple value list:

RECTANGLE(3,5) : 1
11111
11111
11111

4. Double dimension, compound value list:

BAND(6,5) : 1,2,3,4,5

321
4321
54321
54321
5432
543

BAND(6,5) : 1,2,3

321
3321
33321
33321
3332
333

RBAND(6,5) : 1,2,3,4,5

123
1234
12345
12345
2345
345

LOBAND(6,5) : 1,2,3,4,5

1
21
321
4321
54321
54321

PARM(6,4) : 1 *2

1 1
11 11
111 111
11111111
11111111
11111111
111 111
11 11
1 1

LOBAND(6,2) : 1 * 2

1 1
11 11
11 11
11 11
11 11
11 11

RLOBAND(6,3) : 1,2,3

1
12
123
123
123
123

HIBAND(6,5) : 1,2,3,4,5

54321
54321
5432
543
54
5

RHIBAND(6,5) : 1,2,3

12333
12333
2333
333
33
3

PARM(4,6) : 2

2
22
222
2222
2222
2222
222
22
2

RPARM(4,6) : 2

2
22
222
2222
2222
2222
222
22
2

Use of the Data Structures

In order to use these structures to create the matrix representation of a linear programming model, it is necessary to specify their locations in the matrix. Experience in developing a prototype matrix generator showed that an explicit (x,y) coordinate system for expressing sub-array locations was very complex to use even when x and y were expressed as variables. The problem was that each sub-array is rigidly fixed to the origin. Thus a change in the size of a preceding data structure could be reflected only through increasingly complex coordinate expressions.

In response to this problem, a language has been developed in which the sub-array coordinates are implied through specifications of the activities and constraints. For example, suppose the following partial L.P. matrix was to be defined in the matrix generator language:

Activities Constraints	Obtain 1 unit in period 1-----p	Use 1 unit in period 1-----p	Save 1 unit in period 1-----p	Total units accumulated	
Initial inventory	1 1				= Monthly salary
Dispose or Accumulate	-1 -1	1 1	1 1		=0
Disposal Requirements		1 1			= Monthly expenses
Addition to savings			C_1 ----- C_p	-1	=0

We begin by defining a block of activities. In this case we will define the activities of obtaining units in various periods. The ones on the diagonal are the coefficients of the variables whose values the solution algorithm will determine. The statement defining this block of vectors is:

```
**OBTAIN UNITS BY PERIOD
```

The two asterisks denote the definition of a block of activities containing PERIOD variables, where PERIOD has yet to be defined. If, for example, we were considering 12 monthly periods in our model, we would define PERIOD as:

```
PERIOD = 12
```

After a block of activities is defined, the blocks of constraints applicable to those activities are defined and data structures are supplied. For this model we have:

```
* INITIAL INVENTORY BY PERIOD
```

```
DIAGONAL (PERIOD) : 1
```

```
RHS(PERIOD) : MONTHLY SALARY (CARDS)
```

```
* DISPOSE/ACCUMULATE BY PERIOD
```

```
DIAGONAL (PERIOD) : -1
```

```
RHS(PERIOD) : 0
```

Note that RHS stands for right-hand side (constraint level) and that the provision exists for extracting nonzero data from card, tape and disk files.

```
** USE UNITS BY PERIOD
```

```
* DISPOSE/ACCUMULATE BY PERIOD
```

```
DIAGONAL(PERIOD) : 1
```

```
* DISPOSAL REQTS BY PERIOD
```

```
DIAGONAL(PERIOD) : 1
```

```
RHS(PERIOD) : MONTHLY EXPENSES(CARDS)
```

```
** SAVE UNITS BY PERIOD
```

```
* DISPOSE/ACCUMULATE BY PERIOD
```

```
DIAGONAL(PERIOD) : 1
```

```
* ADDITION TO SAVINGS
```

```
ROW(PERIOD) : RETURN VECTOR (DISK)
```

```
RHS(PERIOD) : 0
```

```
** TOTAL UNITS
```

* ADDITION TO SAVINGS

POINT : -1

The preceding illustrates the nature of the matrix-generator language and the format of the statements used to create a linear programming matrix. The next section presents a more complex example which will facilitate the reader's understanding of the procedure, and will demonstrate the use of the generator for a "real" problem.

EXAMPLE MODEL

The model used as an example below is a conventional individual-farm production planning model. The model is applicable to crop-livestock farms, the characteristics of which are defined by the activity sets created and by the nonzero data coefficients used in the matrix.

The model considers the production of crops on discrete tracts of land, at various technological levels (fertilizer, row spacing, etc.), and their disposal by sale or as livestock feeds. The disposal activities are expressed on a periodic basis. The livestock alternatives are the feeding of various types of cattle and hogs, where "type" means starting weight, daily rate of gain, feeding period, finished weight, or any other set of variables affecting price and nutrient requirements.

The Tableau

The example model is presented in tableau form, with generic structures shown for each of the sets of activities. The model sections are shown in order from the top, left-hand corner of the matrix. Where a constraint set is common to more than one set of activities, it is identified in the diagram by the previously used name. The same is true for activity sets using more than one constraint set. The complete matrix tableau can be reconstructed by matching-up the overlapping sets of constraints and activities.

The use of generic structures facilitates the creation of the L.P. matrix for an individual problem through the use of the matrix generator language.

The individual sections of the generalized model are defined and shown below.

1. The model first considers crop growing, which requires land and labor, is subject to restrictions on crop acreages, and produces yields. This is shown in Figure 1.

The coefficients l , l' and l'' are periodic labor requirements, per acre, for each of the crops. A given crop grown at a specified level has a discrete set of these coefficients which are identical for each tract.

The coefficients y_j are expected yields of each crop, at each level,

Figure 1
Crop Production Activities

	Grow 1 acre CROP 1 at LEVEL 1 on TRACT 1-T 1 2 3 . . . T	Grow 1 acre CROP 1 at LEVEL 2 on TRACT 1-T 1 2 3 . . . T	Grow 1 acre CROP C at LEVEL 1 on TRACT 1-T 1 2 3 . . . T	Right Hand Side (R.H.S.)
1-T	1 1 2 1 3 1 T . . . 1	1 1 1 T . . . 1	1 1 1 T . . . 1	\leq acres in TRACT 1-T
e ction -C	1 1 1 1 . . . 1 . . . C	1 1 1 . . . 1 . . . C	1 1 1 . . . 1 . . . C	acreage restriction \leq by CROP 1-C
1-P	1 l . . . l 2 l' . . . l' P l'' . . . l''	1 l . . . l 2 l' . . . l' P l'' . . . l''	1 l . . . l 2 l' . . . l' P l'' . . . l''	labor available \leq by PERIOD 1-P
ory -C	1 -y . . . -y . . . c	1 -y . . . -y . . . c	1 -y . . . -y . . . c	$= 0$
ive w	c . . . c . . . c	c . . . c . . . c	c . . . c . . . c	$= z(\text{min.})$

on each tract. All the y_j are therefore discrete values.

The coefficients c_j are the per acre costs of growing the crop. These are constants for a given crop grown at a specified level, but vary between crops and between levels.

2. To convert the per acre crop yields to total production of each crop, thus allowing the consideration of disposal activities, a set of storage activities is created.

Figure 2
Crop Storage Activities

		Store 1 unit of CROP 1-C	R.H.S.
		1 . . . C	
Inventory by CROP 1-C	1 . . . C	1 . . . 1	See Figure 1
Storage Capacity for CROP 1 by PERIOD 1-P	1 2 . . . P	1 1 . . . 1	Capacity ≤ for CROP 1
Storage Capacity for CROP C by PERIOD 1-P	1 2 . . . P	1 1 . . . P	Capacity ≤ for CROP C
Objective Row		0	See Figure 1

The geometric arrays described earlier in this report are all-inclusive, but a new feature of their use is encountered in Figure 2 and later figures. This is the STEP function, which is used in cases where geometric sub-arrays are repeated in a diagonal format. For example, in Figure 2, the storage capacity constraints are columns of dimension (P), but each column starts one row below the end of the column on its left. This structural format is defined as <COLUMNSTEP>. All other basic array statements can have <STEP> appended to them, with the same effect as here. This feature will be encountered in later sections of the model and the associated generator statements.

3. Since periodic sale of the crop products is a feature of the model, crop sale activities are defined.

Figure 3
Crop Sale Activities

		Sell 1 unit of CROP 1 in PERIOD 1-P				Sell 1 unit of CROP C in PERIOD 1-P				R.H.S.				
		1	2	.	.	.	P	1	2	.	.	.	P	
Storage Capacity for CROP 1 by PERIOD 1-P	1 2 . . . P	-1 -1 -1 . . . -1 -1 . . . -1												See Figure 2
Storage Capacity for CROP C by PERIOD 1-P	1 2 . . . P							-1 -1 -1 . . . -1 -1 . . . -1						See Figure 2
Objective Row		-c . . . -c						-c . . . -c						See Figure 1

The coefficients $-c_j$ are per unit sale prices for each crop in each period.

4. The use of the crop products as livestock feeds (in this case, cattle only are specified, hogs having the same structure) requires removal from storage and specification of nutrient contents of each feed.

Figure 4
Crop Use as Feeds Activities

		Use 1 unit of CROP 1 for Cattle feed in PERIODS 1-P 1 2 . . . P	Use 1 unit of CROP C for Cattle feed in PERIODS 1-P 1 2 . . . P	R.H.S.
Storage Capacity for CROP 1 by PERIOD 1-P	1 2 . . P	-1 -1 -1 . . -1 -1 . . . -1		See Figure 2
Storage Capacity for CROP C by PERIOD 1-P	1 2 . . P		-1 -1 -1 . . -1 -1 . . . -1	See Figure 2
NUTRIENT 1 Inventory by PERIOD 1-P	1 2 . . P	n n . . n	n n . . n	\leq 0
NUTRIENT N Inventory by PERIOD 1-P	1 2 . . P	n' n' . . n'	n' n' . . n'	\geq 0
Objective Row		0	0	See Figure 1

The coefficients, n , are the nutrient analyses per unit for each feedstuff. The inequality constraint for nutrient 1 indicates a maximum requirement for this nutrient, while that for nutrient 2 indicates a minimum requirement.

5. The final section of the model comprises the cattle feeding activities. Each cattle type can be started in the feedlot in periods 1 to p , and is fed for a specified period of time, thus generating band matrices in the nutrient and feedlot capacity constraint rows.

Figure 5
Cattle Feeding Activities

		Feed 1 head Type 1 Cattle starting in PERIOD 1-P					Feed 1 head Type 2 Cattle starting in PERIOD 1-P					R.H.S.
		1	2	.	.	P	1	2	.	.	P	
NUTRIENT 1 Inventory by PERIOD 1-P	1	-n					-n					See Figure 4
	2	-n	-n				-n	-n				
			
	.	-n		
	P					-n	-n	-n				
NUTRIENT N Inventory by PERIOD 1-P	1	-n'					-n'					See Figure 4
	2	-n'	-n'				-n'	-n'				
			
	.	-n'		
	P					-n'	-n'	-n'			-n'	
Feedlot Capacity by PERIOD 1-P	1	1					1					Feedlot ≤ Capacity (No. of head)
	2	1	1				1	1				
			
	.	1	.	.	.		1	.	.	.		
	P					1	1	1			1	
Objective Row		-c	.	.	.	-c					-c	See Figure 1

The coefficients, $-n$, are the periodic requirements for each nutrient per head of cattle. For each type of cattle, the vectors of requirements for each nutrient are discrete. Within each cattle type, the column vectors forming the band matrix for a given nutrient are identical; i.e.,

the coefficients of a given diagonal are constants. The number of coefficients in each of the column vectors (thus the number of bands) is determined by the feeding period specified for the cattle type.

The Matrix Generator Statements

This section demonstrates the use of the matrix generator to specify a matrix for an individual problem conforming to the generic structure described above.

The characteristics of the problem to be modeled are as follows:

1. Five crops are considered:

corn)	2 fertilizer levels each;
soybeans		
rice)	1 fertilizer level each.
sorghum		
alfalfa		

2. Twenty tracts (fields) can be used to grow each crop.

3. Acreage restrictions exist for alfalfa, sorghum and rice. There are no restrictions for corn and soybeans.

4. The number of periods to be considered for crop disposal and cattle feeding is 12 (months).

5. The crop disposal alternatives are:

corn)	sale or cattle feed;
alfalfa		
sorghum		
soybeans)	sale only.
rice		

6. The cattle types to be considered are:

Type 1: 450 lb. start; rate of gain 2.0 lbs. per day; feeding period 9 months;

Type 2: 600 lb. start; rate of gain 2.0 lbs. per day; feeding period 7 months.

In the statements which follow, many of the individual data coefficients are not given explicitly, but are identified by their file locations.

First, the list of variables to be used by the generator is defined. This is followed by the actual structural statements, which are identified by the preceding figures to which they refer.

```
C VARIABLE LIST
CROP = 2; CORN, BEANS
CROP1 = 3; RICE, SORGHUM, ALFALFA
LEVEL = 2; FERTILIZER
LEVEL1 = 1; FERTILIZER
TRACT = 20
CL = CROP * LEVEL
LT = LEVEL * TRACT
CL1 = CROP1 * LEVEL1
LT1 = LEVEL1 * TRACT
CCL = CROP + CROP 1
PERIOD = 12
CATTLE = 2; TYPE 1, TYPE 2
GROWPERIOD = 9
GROWPERIOD1 = 7
CPROD = CL * TRACT + CL1 * TRACT
STORE = CCL * PERIOD
```

Figure 1

```
** CROP PRODUCTION BY CPROD
* ACRES BY TRACT

DIAGONAL (TRACT) : 1 * CL
DIAGONAL (TRACT) : 1 * CL1
RHS (TRACT) : FIELDACRES (CARDS); L
```

The first statement defines the matrix columns to be created (CPROD = 140). The single asterisk statement identifies the constraint set to which the structural statements refer. The DIAGONAL statements cause the creation of 7 unitary diagonals of 20 elements each. The RHS statement says that the 20 constraint level values are on a card file called "FIELDACRES," and that these constraint equations are of the less-than-or-equal type.

* RESTRICTION BY CCL

ROWSTEP (LT) : 0 * CROP

ROWSTEP (LT1) : 1 * CROP 1

RHS (CROP 1) : CROPAREA(CARDS); L,E,G

This set of statements defines the acreage restriction constraints. The first ROWSTEP statement is used merely as a row counter by the generator, since there are no restrictions on corn and beans. The specification of a zero value in these two rows indicates that the rows are to be counted but not generated in the matrix. (This procedure can be used with all other structures recognized by the generator.) The second ROWSTEP statement causes the creation of three stepped unitary rows of 20 elements each. The RHS statement gives the source of the restriction levels and identifies the nature of the inequalities, in order, as less-than-or-equal, equal and greater-than-or-equal.

* LABOR BY PERIOD

COLUMN (PERIOD) : CORNLABOR (CARDS) * LT

COLUMN (PERIOD) : BEANLABOR (CARDS) * LT

COLUMN (PERIOD) : RICELABOR (CARDS) * LT1

COLUMN (PERIOD) : SORGLABOR (CARDS) * LT1

COLUMN (PERIOD) : ALFALFALABOR (CARDS) * LT1

RHS (PERIOD) : MANHOURS (CARDS); L

The above statements define the structure of the labor constraints. Each COLUMN statement specifies that a set of 12 values will be found in the named file, and that each column is to be repeated the number of times specified by the repeat factor. In this case, the labor requirements per acre for each level of corn (and soybeans) are the same, so only one set of coefficients is necessary for each crop. However, cases could arise where the labor requirements would differ between levels. In such cases, the crops would be defined as CORN and CORNL, and the labor requirements specified for each. Thus, the definition of variables can be used to create models with quite different characteristics, within a given generic structure. This holds, of course, for applications in many areas other than agriculture.

```

* INVENTORY BY CCL

ROWSTEP (LT) : CORNYIELD (CARDS)
ROWSTEP (LT) : BEANYIELD (CARDS)
ROWSTEP (LT1) : RICEYIELD (CARDS)
ROWSTEP (LT1) : SORGYIELD (CARDS)
ROWSTEP (LT1) : ALFALFAYIELD (CARDS)
RHS(CCL) : 0 ; E

```

The ROWSTEP statements above generate the yield rows, the coefficients being retrieved from a series of card files. The RHS statement specifies that the constraint levels are zero. This statement, although resulting in no generated matrix values, is included because an RHS statement is mandatory for each set of constraints specified, excepting the OBJECTIVE ROW set.

```

* OBJECTIVE ROW

OBJECTIVE (TRACT) : CORNCOST (CARDS)
OBJECTIVE (TRACT) : CORNCOST1 (CARDS)
OBJECTIVE (TRACT) : BEANCOST (CARDS)
OBJECTIVE (TRACT) : BEANCOST1 (CARDS)
OBJECTIVE (TRACT) : RICECOST (CARDS)
OBJECTIVE (TRACT) : SORGCOST (CARDS)
OBJECTIVE (TRACT) : ALFALFACOST (CARDS)

```

The OBJECTIVE ROW coefficients for this section of the model are the per acre growing costs for each crop and level of crop.

Figure 2

```

** CROP STORAGE BY CCL

* INVENTORY BY CCL
DIAGONAL (CCL) : 1

* STORAGE BY STORE

COLUMNSTEP (PERIOD) : 1 * CCL
RHS (PERIOD) : CORNSTORE (CARDS) ; L
RHS (PERIOD) : BEANSTORE (CARDS) ; L
RHS (PERIOD) : RICESTORE (CARDS) ; L
RHS (PERIOD) : SORGSTORE (CARDS) ; L

```

```
* RHS (PERIOD) : ALFALFASTORE (CARDS) ; L
```

It should be noted that the INVENTORY constraint set defined above does not require an RHS statement, this having been defined in the generator statements referring to Figure 1.

```
* OBJECTIVE ROW  
OBJECTIVE (CCL):0
```

Although the objective coefficients for this section of the model are zero, and are therefore not generated, the OBJECTIVE statement is included because it is mandatory for each set of activities defined.

Figure 3

```
** CROP SALES BY STORE  
* STORAGE BY STORE  
LOTRISTEP(PERIOD)*-1,-1,-1,-1,0*CCL  
* OBJECTIVE ROW  
  
OBJECTIVE(PERIOD):CORNPRICE(CARDS)  
OBJECTIVE(PERIOD):BEANPRICE(CARDS)  
OBJECTIVE(PERIOD):RICEPRICE(CARDS)  
OBJECTIVE(PERIOD):SORGPRICE(CARDS)  
OBJECTIVE(PERIOD):0
```

The LOTRISTEP statement above creates four contiguous, stepped lower triangular matrices of value -1 and leaves a zero array of the same dimension in the set of storage constraints defined for alfalfa, since this is not sold. This is necessary because the number of constraints specified in the structural statement must conform to that defined by the single asterisk statement, in which <STORE> is the operative variable.

The zero value in the last statement corresponds to alfalfa, which is not sold.

Figure 4

```
** CROP FEEDING TO CATTLE BY STORE  
* STORAGE BY STORE  
LOTRISTEP(PERIOD): -1,0,0,-1,-1*CCL  
* NUTRIENT 1 BY PERIOD  
DIAGONAL(PERIOD: CROPANALYSIS1(CARDS)*CCL  
RHS (PERIOD):0;L
```

```

* NUTRIENT N BY PERIOD
  DIAGONAL(PERIOD):CROPANALYSISN(CARDS)* 1
  RHS(PERIOD):O;G

* OBJECTIVE ROW
  OBJECTIVE(STORE):O

```

The zero values in the LOTRISTEP statement above are included because neither soybeans nor rice are to be considered as cattle feeds. The remainder of the above is self-explanatory.

Figure 5

```

** CATTLE FEEDING BY CATTLE*PERIOD

* NUTRIENT 1 BY PERIOD
  LOBAND(PERIOD,GROWPERIOD):C1NUTRIENT1(CARDS)
  LOBAND(PERIOD,GROWPERIOD1):C2NUTRIENT1(CARDS)

* NUTRIENT N BY PERIOD
  LOBAND(PERIOD,GROWPERIOD):1
  LOBAND(PERIOD,GROWPERIOD1):1

* FEEDLOT CAPACITY BY PERIOD
  LOBAND(PERIOD,GROWPERIOD):1
  LOBAND(PERIOD,GROWPERIOD1):1
  RHS(PERIOD:FEEDCAPACITY(CARDS);L

* OBJECTIVE ROW
  OBJECTIVE(PERIOD:C1PRICE(CARDS)
  OBJECTIVE(PERIOD:C2PRICE(CARDS)

*** END

```

The above series of statements defines the cattle feeding sections, causing the structuring of a series of band matrices with the number of bands (9 and 7 respectively) conforming to the feeding period (GROWPERIOD) for each cattle type. The data on nutrient requirements are derived from a series of card files as specified.

The output from the matrix generator is in SHARE standard format, which is accepted by most linear programming codes. In the case of the above example, the generator will produce a model of 289 columns, 139 rows and 3,459 nonzero coefficients. The non-unit coefficients must, of course,

be cardpunched, but due to the use of repeat factors, the number of coefficients input is small (508). A further reduction in card input is achieved by the use of the row type descriptors in the <RHS> statements. The net reduction here is 139. Thus, the number of cards required to input this model is reduced from 3598 to 161.

An additional advantage derived from this type of matrix generator is that the user-defined names can be passed to a report-generator which interprets the solutions and outputs them in a report, with format specified by the user, in which the assigned names are utilized. The development of a report generator, which will contribute further to cost reductions in the use of linear programming, is currently being undertaken. This will be the subject of a later paper.

CONCLUSION

As the use of linear programming becomes more widespread in the solution of large resource-allocation problems, it is becoming evident that automation and simplification of matrix preparation and computer input must be accomplished in order to reduce the high costs of using the technique.

We feel that the matrix generator language described in this paper represents a significant advance in the state of the art, and will lead to considerable cost reductions in many areas in which linear programming is applied. This generator has the advantages that it is easy to learn and apply; it allows complete user freedom in the naming of activities and constraints; its use greatly facilitates the introduction of changes in the structure of models; and the concept of the generator itself is independent of the linear programming code, the computer, and the model with which it is to be used.

REFERENCES

1. I.W. Marceau, R.E. Tongate, C.B. Baker and John T. Scott, jr.,
A Computer-Assisted Planning Method for Individual Farms, Special
Publication 20, College of Agriculture, University of Illinois at
Urbana-Champaign (February, 1971), 40 pp.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

REPORT TITLE

Matrix-Generator System for Linear Programming Problems

DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

AUTHOR(S) (First name, middle initial, last name)

John W. Marceau and Thomas W. Mason

REPORT DATE

June 30, 1971

7a. TOTAL NO. OF PAGES

33

7b. NO. OF REFS

1

CONTRACT OR GRANT NO.

DAF 30(602)-4144

PROJECT NO.

ORPA Order 788

8a. ORIGINATOR'S REPORT NUMBER(S)

CAC Document No. 4

8b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

DISTRIBUTION STATEMENT

Copies may be requested from the address given in (1) above.

SUPPLEMENTARY NOTES

None

12. SPONSORING MILITARY ACTIVITY

Rome Air Development Center
Griffiss Air Force Base
Rome, New York 13440

ABSTRACT

A matrix-generator language for use in structuring and inputting linear programming problem matrices is described. The generator is based on the definition of the geometric sub-arrays which occur within a sparse linear programming matrix. An exhaustive list of sub-array structures is given, and their use within the described generator language is demonstrated by the presentation of a real-world agricultural planning problem. Estimates of the cost-reducing potential of this generator system are given.

14. KEY WORDS	LINK A		LINK B		LINK C
	ROLE	WT	ROLE	WT	ROLE
Linear Programming Procedure and Problem Oriented Languages (Miscellaneous)					



UNIVERSITY OF ILLINOIS-URBANA

510.84IL63C C001
CAC DOCUMENT\$URBANA
1-10 1970



3 0112 007263764