

UIMP: User Interface for Mathematical Programming

E. F. D. ELLISON

UNICOM Consultants, UK Ltd.

and

GAUTAM MITRA

Brunel University, England, and UNICOM Consultants, UK Ltd.

UIMP is a matrix-generator report-writer system designed to aid the realization (generation) of mathematical programming models and also the analysis-reporting of the solutions of such models. The data-structure facility of the system allows the underlying structure of a user model to be captured and helps to define such models. This data-structure feature is not only a powerful modeling aid, it also finds use in the analysis of solutions and report generation. The experience of using the system, its shortcomings, and possible extensions are also discussed.

Categories and Subject Descriptors D.3.2 [Programming Languages]: Language Classifications—*applicative languages*, G.1.6 [Numerical Analysis]: Optimization—*linear programming*

General Terms None

Additional Key Words and Phrases Matrix generator, report writer, solution analysis

1. INTRODUCTION

A number of matrix-generator report-writer (MGRW) systems are in use in industry, and for any organization that uses mathematical programming (MP) as a serious modeling tool such systems are of considerable value. Without exception such systems are used in conjunction with proven MP software: [11] contains a brief survey of such systems.

To understand the requirements of an MGRW system in which matrix-generator (MG) programs and report-writer (RW) programs are written it is necessary to identify the tasks performed by an MG or a RW. An MG or a RW may, of course, be written in a high-level computer language such as FORTRAN, ALGOL, PL1, or in an MGRW language. A typical modeling system using an MG, an OPTIMIZER, and a RW as shown in Figure 1 may work in the following way. The PROBLEM DATA is presented in the form of sets of tabulated information. These are read and processed by an MG program that produces a card image INPUT FILE. This usually contains,

Authors' addresses: E. F. D. Ellison, UNICOM Consultants, UK Ltd., G. Mitra, Department of Statistics and Operations Research, Brunel University, Uxbridge, Middlesex UB8 3PH, England.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1982 ACM 0098-3500/82/0900-0229 \$00.75

ACM Transactions on Mathematical Software, Vol. 8, No. 3, September 1982, Pages 229-255

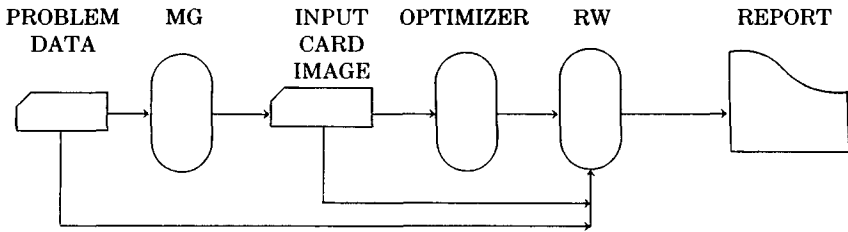


Fig 1 Matrix generation, optimization, and report writing: a flow diagram.

- (a) the logical (i.e., slack and artificial) variables names or row names that are coded by suitable text expressions;
- (b) the structural variables names or column names that are similarly coded;
- (c) the coefficients of the problem matrix;
- (d) the RHS, BOUNDS, and RANGES information;
- (e) some information concerning the starting basis.

A RW is used primarily to extract only the pertinent information from the solution obtained by the OPTIMIZER, and it presents this information in a suitably tabulated format. The RW usually consults the PROBLEM DATA held as tables, and the BCD INPUT; it may also carry out some arithmetical operations on these solution values. An MGRW system therefore incorporates at least the following features:

- (i) input of PROBLEM DATA in tabular form;
- (ii) construction of row and column names by name expressions;
- (iii) using constants or arithmetic expressions to specify the matrix, RHS, BOUND, etc., coefficients by suitable row or column generator clause (procedure);
- (iv) accessing the solution file to obtain solution values, reduced costs, ranges, etc.;
- (v) format and print tabular information.

These features are, of course, common to all of the known conventional MGRW systems. The UIMP system provides, in addition, a great emphasis on the data-structuring aspect of MP modeling. To illustrate this the following observations may be made. One important aspect of real-life MP models is that the variables as well as constraints of these models usually appear in groups. Such groupings may be typically found in models involving multi time periods, multiproducts, or with decentralized facilities, etc. This type of structure of a model taken from real life is naturally reflected in the input data structure and the structure of the constraint matrix. In the UIMP language a data-structure facility is provided explicitly whereby such models, in spite of their apparent complexity, may be constructed in a straightforward fashion. In Section 2 of this paper an overview of the language is presented and some syntactic components are defined. In Section 3 a small but realistic problem is presented. The problem is illustrative of a structure commonly found in practice; the tactics adopted to formulate this model explain the principle of use of the language. In Section 4 some ramifications of the data-structure facility are presented. In Section 5 the illustrative problem is fully coded in UIMP language and the various steps are annotated.

The experience of using the system, its shortcomings, and a possible alternative approach to matrix generation and reporting are discussed in Section 6. Appendix A contains a summary of UIMP commands, and the translator, executor, and the information flow through the system are explained in Appendix B.

A number of matrix-generator report-writer systems have been developed and are in use. The most prominent of these are MAGEN [2], MGRW (IBM) [4], GAMMA.3 [10], DATAFORM [7], and MGG/RWG [9].

However, what mainly distinguishes the UIMP system is the way its data-structure facility plays a central role in the model definition tactics. In developing a simple (linear) optimization model one first introduces the decision variables and then their constraint relationships. In developing a structured model (that most real-life models are structured is discussed earlier in this section) relevant subscripts and sets of indices for these subscripts need to be introduced in the first step. The decision variables as well as matrix (technology) coefficients are then introduced in terms of these subscripts. In UIMP the structures are used in a way comparable to the use of the subscripts and the sets. From an information processing point of view these structures may be looked upon as trees or networks (Knuth [8]). These structures are first introduced to program a particular model and its output report-analysis, etc. The input tables, the decision variables (the columns), the logical variables (the rows), and the output tables are all defined subsequently in terms of these structures. This sequence of model construction obliges the modeler to first think out the structure and then define the model relationships. This, if one analyzes the task of model construction, is a logical and successful approach. The other modeling languages do not necessarily oblige the users to follow such a sequence; however, a successful modeler naturally discovers this for himself.

UIMP has been developed as a machine-independent system. The system has been implemented using American National Standard FORTRAN IV in as much as this is possible. In addition, a number of software tools (preprocessing of the source to generate FORTRAN for a target machine) and software building blocks (character handler, file handler, etc.) have been used. The techniques used are very similar to those described by Kernighan and Plauger [6]. The system is thus implemented [12, 13] to run on CDC CYBER 170 series computers with 60-bit word size and also to run on ICL 2900 series computers with 32-bit word size. The latter system [5] is called MGRW 2900: it is otherwise identical with the UIMP system.

2 AN OVERVIEW OF THE LANGUAGE AND THE BASIC SYNTACTIC COMPONENTS

The UIMP system is essentially a high-level language compiler and executor; the language is of course limited in its generality. The language provides facilities for easy definition and manipulation of tabular data. In addition to numeric or text information one may also define structure made out of structure elements. The structure elements can be used in name expressions to construct names for the ROWS, COLUMNS, RHS, BOUNDS, etc., for an LP model. As a part of the language a few generator clauses are provided that are used to construct the model by "rows" or by "columns." These clauses require (cf. syntax later) that name expressions be provided for model row and column names and arithmetical

expressions be used to specify the model coefficients. The language possesses a block structure; that is, a set of UIMP statements may be bracketed by BEGIN and END commands. The language also incorporates table-manipulative and table-print facilities that find use in RW programs.

The basic syntax of the language is described using the American National Standard COBOL syntax notation [1]. This brief description is supplemented by explanatory notes where appropriate. A detailed introduction to the system may be found in [12], and the full reference specification is supplied in either [13] or [5].

2.1 Preliminary Syntax

The following metalinguistic notations are introduced to define the UIMP language.

- (i) “:=” is the metalanguage connective meaning “is syntactically defined as.”
- (ii) The optional components of syntax are placed in square brackets [...] while braces {...} indicate an obligatory component of the syntax.
- (iii) Alternatives may appear within braces or within square brackets and are written above each other.
- (iv) In case of repeated occurrence of a notation variable, these variables may be numbered, for example, *identifier-1*, *identifier-2*. The numbering in this instance is simply for the convenience of explaining the semantics.
- (v) Indefinitely repeated occurrence of notation variables is indicated by a string of three dots in square brackets, that is, [...] or [, ...]. In the latter instance the comma as a separator must appear.

2.2 Simple Declarator Clauses

The integer, real, text, and structure variables are declared by the clause

LET VARIABLE {*identifier list*} BE TYPE $\left. \begin{array}{l} \text{STRUCTURE} \\ \text{INTEGER} \\ \text{REAL} \\ \text{TEXT} \end{array} \right\}$

where *identifier list* is defined as

$$\textit{identifier list} ::= \textit{identifier} [, \dots],$$

and an *identifier* is defined as a sequence of up to eight characters (alpha or numeric), and always starting with an alpha character.

```
LET VARIABLE I1 BE TYPE INTEGER
LET VARIABLES Q, S1, S2, S3 BE TYPE STRUCTURE
```

are examples of this declarator clause.

The structure declaration takes the form

```
LET STRUCTURE {str-identifier} BE {str-identifier list}
```

Examples of this are

```
LET STRUCTURE Q1 BE SA, SB, SC
LET STRUCTURE SA BE P1, P2, P3
```

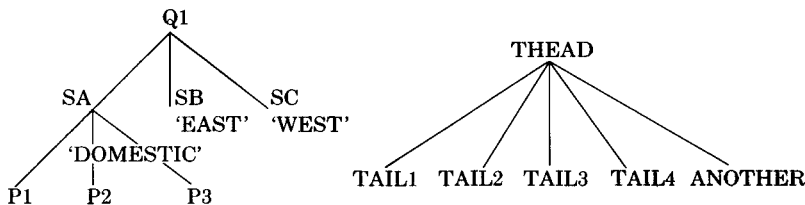


Fig 2 Structure relationships, two examples

```
LET STRUCTURE THEAD BE
@ LIST I . = 1 STEP 1 UNTIL 4 / "TAIL*", ANOTHER
```

These declarations lead to the two structures shown in Figure 2. In these two structures the (structure) identifiers Q1, SA, THEAD represent the set of structure elements that immediately follow. Depending on the context, therefore, SA may be referred to and used as a structure identifier, implying there is a substructure made up of *structure elements*. It may also be referred to as an element that implies there is a superstructure, in this case Q1, to which it belongs.

The following properties are associated with *structure elements* and *structure identifiers*.

- (i) A *structure identifier*, referring to the head of the structure containing a set of *structure elements*, has a cardinality that is the number of elements in the immediate substructure. This is expressed in the form N: *structure identifier*. Thus N:THEAD has the value 5.
- (ii) A structure element has an ordinality within its super structure expressed as I: *structure element* IN *structure identifier*. Thus I:SB IN Q1 has the value 2.
- (iii) The character string making up the name of the structure element is obtained by bracketing the structure element.
- (iv) One may associate with a *structure element* or a *structure identifier* text strings of any length. These may be referred to using the prefix T:. The following text assignment statements illustrate its use:

```
T: SA := 'DOMESTIC'
T: SB := 'EAST'
T: SC := 'WEST'
```

These can be used as text strings in the text expressions and the print statements, etc.

2.3 Declaration of Tables

The following declarator clause is used to declare tables:

```
LET TABLE identifier list BE {
    structure DOWN
    structure ACROSS
    structure DOWN BY structure ACROSS
}

TYPE {
    INTEGER
    REAL
    TEXT
}
```

EXAMPLE	Q1			SB	SC
	SA	P2	P3		
	P1				
THEAD	TAIL1				
	TAIL2				
	TAIL3				
	TAIL4				
ANOTHER					

Fig 3 Down and across structures defining table margins.

Example

LET TABLE EXAMPLE BE THEAD DOWN BY Q1 ACROSS TYPE REAL

This leads to the table called EXAMPLE shown in Figure 3. The table definition can be made implicit at the time of data entry (cf. Section 4).

2.4 Structure Element Context and Enumeration

To identify a particular cell in a table, such as the one shown in Figure 3, it is necessary to devise a scheme for referencing the components of the row and column structuring. For example, column 3 is identified by "P3 IN SA IN Q1": this illustrates the form "element context." In this, each element is quoted in the context of its immediate superior in the hierarchy. An element may have different superiors in the whole structure, but a context specifies a particular one.

This is syntactically defined as

$$element\ context ::= \left\{ \begin{array}{l} named\ element \\ structure\ variable \end{array} \right\} [IN \dots]$$

context is an important attribute of the structure variables. Assignments of this may be made by statements such as Q := P3 IN SA IN Q1, where Q is a structure variable as defined earlier, and P3, SA, Q1 are named elements of a structure. Q may now be used as a reference to the column in the table of Figure 3. Then the statement EXAMPLE(TAIL 2 IN THEAD, Q) ... refers to the table location row 2 and column 3.

Enumerative forms are generally used to specify repetition. In UIMP the general form is called *element context enumeration* and is defined as

$$element\ context\ enumeration ::= \left\{ \begin{array}{l} structure\ variable\ IN\ element\ context \\ structure\ variable\ [IN\ element\ context]\ structure\ variable \\ structure\ variable\ [IN\ element\ context]\ element\ context\ enumeration \end{array} \right\} .$$

Element context enumerations are used in FOR and SUM enumerative clauses which take the form

FOR *element context enumeration* [*parallel for clause*][*when clause*]
 SUM *element context enumeration* [*parallel for clause*][*when clause*]
 where *parallel for clause* takes the form
 AND FOR *element context enumeration*

and *when clause* takes the form
 WHEN *conditional expression*

FOR therefore enumerates the repetition of a statement. A group of statements may also be repeated by using DO, and the BEGIN . . . END brackets, as shown below:

```
FOR ... DO
    BEGIN
        statement-1
        statement-2
        :
        :
    END
```

Similar constructs involving SUM enumerate the summation of a subexpression within a linear form that is defined in Section 2.5. In each case the enumeration consists of allowing one or more structure variables to carry through all the assignments (for all possible element contexts).

2.5 Linear and Column Form

The row generator and column generator statements defined in Section 2.6 use the linear form and the column form that are defined in this section

$$linear\ form ::= \left[\left\{ \begin{array}{c} PLUS \\ MINUS \end{array} \right\} \right] linear\ term \left[\left\{ \begin{array}{c} PLUS \\ MINUS \end{array} \right\} \dots \right],$$

where linear term is defined as

$$linear\ term ::= \left\{ \begin{array}{l} simple\ linear\ term \\ compound\ linear\ term \end{array} \right\}.$$

The simple linear term and the compound linear term are, in turn, defined as

$$simple\ linear\ term ::= \left\{ \begin{array}{l} arithmetic\ expression\ TIMES\ model\ variable\ reference \\ model\ variable\ reference\ TIMES\ arithmetic\ expression \\ model\ variable\ reference \end{array} \right\}$$

$$compound\ linear\ term ::= \begin{array}{l} enumerative \\ sum\ clause \end{array} \left\{ \begin{array}{l} linear\ term \\ left\ square\ bracket\ linear\ form\ right\ square\ bracket \end{array} \right\},$$

where *enumerative sum clause* is described in Section 2.4; *left square bracket* and *right square bracket* represent the two characters “[” and “],” respectively.

The *column form* is syntactically very similar to the *linear form* and is defined as

$$column\ form ::= column\ term [ALSO \dots],$$

where *column term* is defined as

$$column\ term ::= \left\{ \begin{array}{l} simple\ column\ term \\ compound\ column\ term \end{array} \right\}.$$

The *simple column term* and the *compound column term* are, in turn, defined as

$$\begin{aligned} \text{simple column term} & ::= \left\{ \begin{array}{l} \text{arithmetic expression ON model constraint reference} \\ \text{model constraint reference} \end{array} \right\} \\ \text{compound column term} & ::= \text{enumerative for clause} \left\{ \begin{array}{l} \text{column term} \\ \text{left square bracket} \quad \text{column term} \quad \text{right square bracket} \end{array} \right\}, \end{aligned}$$

where *enumerative for clause* is defined in Section 2.4.

2.6 Generating the Model

A group of variables or constraints of an LP model is declared by a declarator clause of the form

```
LET CLASS { MODVAR } identifier list BE structure-1
          { MODCON }
[BY structure-2] [typespec] [solution attribute declaration]
```

In the declaration the *typespec* defines the model variable or constraint types. For variables

$$\text{typespec} ::= \text{TYPE} \left\{ \begin{array}{l} \text{PL} \\ \text{MI} \\ \text{FX} \\ \text{FR} \end{array} \right\},$$

where

PL—nonnegative variables, the default choice
 MI—nonpositive, variables
 FX—fixed, variables
 FR—free, variables

specify the type of model variable.

For constraints

$$\text{typespec} ::= \text{TYPE} \left\{ \begin{array}{l} \text{LE} \\ \text{GE} \\ \text{EQ} \\ \text{FR} \end{array} \right\},$$

where

LE—less than or equal to, the default choice
 GE—greater than or equal to
 EQ—equal to
 FR—no restrictions that is free

specify the type of model constraint.

The *solution attribute declaration* takes the form

$$\text{solution attribute declaration} ::= \text{ATTRIBUTE } \left\{ \begin{array}{l} X \\ DJ \\ ST \end{array} \right\},$$

where

- X—solution value
- DJ—reduced cost coefficient
- ST—basis status

specify the attribute that is of interest.

A model variable or constraint, when declared, always possesses an implicit attribute CODE that is used as MPS column or row name.

The actual row or column relationships are specified by a row generator statement:

$$\begin{aligned} \text{row generator statement} ::= & \text{ROW model constraint reference [typespec]} \\ & [\text{IS linear form}] [\text{rhs clause}[\dots]] \\ & [\text{range clause}[\dots]] \\ & [\text{pivot clause}[\dots]], \end{aligned}$$

where the last three clauses take the form

$$\text{rhs clause} ::= \text{RHS name expression VALUE arithmetic expression}$$

$$\text{range clause} ::= \text{RANGE name expression VALUE arithmetic expression}$$

$$\text{pivot clause} ::= \text{VARIABLE model variable reference PIVOTS OUT}$$

$$\text{ROW } \left[\text{AT } \left\{ \begin{array}{l} \text{LB} \\ \text{UB} \end{array} \right\} \right]$$

or a column generator statement:

$$\text{column generator statement} ::= \text{COLUMN model variable reference}$$

$$\left[\text{TYPE } \left\{ \begin{array}{l} \text{PL} \\ \text{MI} \\ \text{FX} \\ \text{FR} \end{array} \right\} \right]$$

$$\begin{aligned} & [\text{HAS column form}] \\ & [\text{bound value clause}] \\ & [\text{set bound clause}] \end{aligned}$$

where the last two clauses take the form

bound value clause

$$::= \text{BOUND name expression } \left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{UP} \\ \text{LO} \\ \text{FX} \end{array} \right\} \text{VALUE arithmetic expression} \\ \text{MI} \\ \text{FR} \end{array} \right\}$$

$$\text{set bound clause} ::= \text{IS SET TO } \left\{ \begin{array}{l} \text{UB} \\ \text{LB} \end{array} \right\}$$

The pivot clause of the row generator statement together with the set bound clause of the column generator statement allows the modeler to specify a “starting point” or a “basis.” No other MGRW system, as far as the authors are aware, provides this facility.

2.7 Table Print Clause

The table print statement allows the user to specify format layouts for the margins as well as the body of the table. The statement takes the form

$$\text{table print statement} ::= \text{PRINT TABLE } \left\{ \begin{array}{l} \text{table name} \\ \text{code attribute table} \\ \text{solution attribute table} \end{array} \right\}$$

where the formats mentioned are supplied elsewhere in the program body.

3. PRINCIPLE OF USE AND MODELING TACTICS ILLUSTRATED BY AN EXAMPLE

In this section an illustrative problem is first described and its linear programming formulation in mathematical notation is provided. A user formulation is then prepared. This is, of course, in a format that may be directly used for solution by an optimizer.

3.1 An Illustrative Example

A company manufactures three products P1, P2, P3 (NUTS, BOLTS, WASHERS) and has at its disposal three machines M1, M2, M3. The company can undertake normal and overtime production and needs to plan for two time periods, say WINTER and SUMMER. Any product left after the second time period has very little resale value. The necessary information concerning the operation of the company is set out in Tables I-III.

It is necessary to find an LP formulation that maximizes the profit of the company's operation over the two periods. The management also wishes to have a report of the optimum plan in a layout shown in Table IV.

3.2 A Mathematical Formulation

Let the four indices i, j, k, l be defined as

$i = 1, 2$ the index for the two time periods, Summer and Winter;
 $j = 1, 2$ the index for the two modes of production, Normal, Overtime;
 $k = 1, 2, 3$ the index for the three product types, P1, P2, P3;
 $l = 1, 2, 3$ the index for the three machines, M1, M2, M3.

The following information relating to the problem are available in the table TABH,

t_{ijkl} number of hours required to produce one unit of the product type k on the machine l , in the time period i , using Normal or Overtime production j ;
 a_{ijl} machine availability in hours for the machine l in period i and mode j .

Table I. Table of Machine-Hours (TABH)

	Summer Period (H1)								Winter Period (H2)							
	Normal working hours (N)			Overtime (O)			Total hours available (AV)		Normal working hours (N)			Overtime (O)			Total hours available (AV)	
	P1	P2	P3	P1	P2	P3	Normal working hours	Over-time	P1	P2	P3	P1	P2	P3	Normal working hours	Over-time
Machine 1 (M1)	4	5	6	3	4	5	100	80	5	6	7	4	5	5	110	90
Machine 2 (M2)	7	6	6	6	5	5	100	90	8	7	7	7	6	6	110	100
Machine 3 (M3)	3	—	—	2	—	—	40	30	4	—	—	3	—	—	50	40

Note P1 = nuts, P2 = bolts; P3 = washers.

Table II. Table of Production Costs (TABC)

	Summer Period							Winter Period					
	Normal working hours			Overtime				Normal working hours			Overtime		
	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3	
Machine 1	2	3	4	3	4	5	3	4	5	4	5	6	
Machine 2	4	3	2	5	4	3	5	4	3	6	5	4	
Machine 3	1	—	—	2	—	—	2	—	—	3	—	—	

Note P1 = nuts, P2 = bolts, P3 = washers.

Table III. Table of Additional Data (TABD)

	Summer period			Winter period		
	Nuts	Bolts	Washers	Nuts	Bolts	Washers
Sale price	10	10	9	11	11	10
Minimum demand	25	30	30	30	25	25
Storage data						
Capacity	20	20	—			
Cost	1	1	1			
Resale value				2	2	1

In the table TABD,

- p_{ik} selling price,
- d_{ik} demand,
- s_k storage cost for the product type k in one time period;
- h_k the corresponding storage capacity;
- r_k the final resale value at the end.

Table IV. Production Schedule for Two Periods Set Out Against Demand

	Summer period			Winter period		
	Nuts	Bolts	Washers	Nuts	Bolts	Washers
Machine 1	xx.x	xx x	xx x	xx.x	xx x	xx.x
Machine 2	xx x	xx x	xx x	xx.x	xx.x	xx.x
Machine 3	xx.x	xx x	xx x	xx x	xx x	xx x
Demand	xx x	xx.x	xx.x	xx.x	xx x	xx x
Total	xx.x	xx x	xx x	xx x	xx x	xx.x

Note All production in 1000 lbs weight of iron.

In the table TABC,

c_{ijkl} the production cost in the category i, j, k, l .

The variables of the problem follow. Let,

x_{ijkl} denote the quantity to be produced in the category i, j, k, l ;

y_{ik} denote the quantity of product k stored in the period i ;

z_{ik} denote the quantity of product k sold in the period i .

The profit function of the problem may be expressed as

$$\text{Profit} = \sum_{i=1}^2 \sum_{j=1}^2 \sum_{k=1}^3 \sum_{l=1}^3 (p_{ik} - c_{ijkl})x_{ijkl} - \sum_{k=1}^3 s_k y_{1k} + \sum_{k=1}^3 (r_k - p_{2k})y_{2k}.$$

In an optimal plan Profit must be maximized subject to the constraints,

(i) machine availability,

$$\sum_{k=1}^3 t_{ijkl} \cdot x_{ijkl} \leq a_{ijl}, \quad \text{for all } i, j, l;$$

(ii) stock balance in the two periods,

$$\sum_{j=1}^2 \sum_{l=1}^3 x_{1jkl} - y_{1k} - z_{1k} = 0, \quad \text{for period 1, and all } k,$$

and

$$\sum_{j=1}^2 \sum_{l=1}^3 x_{2jkl} + y_{1k} - y_{2k} - z_{2k} = 0, \quad \text{for period 2, and all } k;$$

(iii) minimum demand to be satisfied

$$z_{ik} \geq d_{ik}, \quad \text{for all } i, \text{ and } k;$$

(iv) upper bound on storage,

$$y_{1k} \leq h_k, \quad \text{for all } k;$$

(v) nonnegativity of the variables,

$$y_{ik} \geq 0, \quad \text{for all } i, k, \quad \text{and} \quad x_{ijkl} \geq 0, \quad \text{for all } i, j, k, l.$$

```

NAME      STB
XL T1NP1M1 T1P1ST
XL T1NP2M1 T1P2ST
XL T1NP3M1 T1P3ST
XL T2NP1M1 T2P1ST
XL T2NP2M1 T2P2ST
XL T2NP3M1 T2P3ST
LL T1P1D
LL T1P2D
LL T1P3D
LL T2P1D
LL T2P2D
LL T2P3D
ENDATA

```

Fig 4 A starting basis specified in MPS format

3 3 Formulation of the Problem for an LP User

To formulate the model in a form that can be processed by an OPTIMIZER it is necessary to construct names for the rows and columns of the model, specify the values of the matrix, cost, rhs coefficients, and prepare these in a card image format. In the subsequent discussion input format of MPSX [3] is taken as the standard. It is noted there are three groups of variables in the model. These are x , y , z , and let these groups be called PRODN, STORE, DEMAND.

Naming the variables:

PRODN set variables; let these be called T1NP1M1 for x_{111} , T1OP1M1 for x_{121} , etc.;

STORE set variables; let these be called T1P1STR for y_{11} , etc.;

DEMAND set variables; let these be called T1P1D for z_{11} , etc.

It follows from the table entries that not all $x_{i,j,k}$, hence the corresponding $T_i \dots M_1$ variables, are defined.

Naming the constraints:

PROFIT the name of the objective row,

T1M1AN availability constraint (cf. (i) of machine M1 in time period T1 under Normal shift, etc.),

T1P1ST stock balance equation (cf. (ii) in time period T1 for the product P1, etc.).

Let the RHS column be called RHS, and the BOUNDS be called LIM. The corresponding LP user formulation of the model is derived and set out in Table V; also note that a partial starting basis STB can be suggested as shown in Figure 4.

4. SOME RAMIFICATIONS OF THE DATA-STRUCTURE FACILITY, DATA DEFINITION, AND DATA ENTRY

It follows from a cursory look at the tables TABH, TABD, TABC used in the last section that some structuring is already invoked in the rows (DOWN) and columns (ACROSS) of these tables. For instance, in the table TABH a set TH is made up of H1, H2, the production information in period 1 and period 2. In each time period the operation can be of the two types, Normal (N) or Overtime (O); one is also concerned with the availability (AV) of the machines in these two time periods.

Finally, each of the Normal (N) and Overtime (O) operations is again broken down by product type, P1, P2, P3. It would be relatively easy to formulate the model if the following sets could be defined, namely,

$$\begin{aligned} \text{TH} &= \{\text{H1}, \text{H2}\}, & \text{H1} &= \{\text{N}, \text{O}, \text{AV}\}, & \text{H2} &= \{\text{N}, \text{O}, \text{AV}\} \\ \text{N} &= \{\text{P1}, \text{P2}, \text{P3}\}, & \text{O} &= \{\text{P1}, \text{P2}, \text{P3}\} \dots, \text{etc.}, \end{aligned}$$

The structure facility of UIMP achieves exactly this: it allows such elements as TH, H1, N, P1, etc., to be defined and used in a meaningful way. In the opinion of the authors if the input tables and the data are properly structured, then this already goes a long way toward the compact yet comprehensive definition of the model.

To generate the model already formulated in Section 3, it is necessary to write an input (program) module in UIMP. The module is called MGTABS and is set out immediately below. Various statements in the module, as identified by their sequence numbers, are annotated.

```
00010  MODULE MGTABS
00020  LET STRUCTURE TH BE H1, H2; LET STRUCTURE H1, H2 BE N, O, AV
00030  LET STRUCTURE N, O BE P1, P2, P3; LET STRUCTURE AV BE AN, AO
00040  LET STRUCTURE MACH BE M1, M2, M3
00050  LET STRUCTURE TD BE T1, T2; LET STRUCTURE T1, T2 BE P1, P2, P3
00060  LET STRUCTURE DROW BE PRICE, DEM, STR
00070  LET STRUCTURE STR BE CAP, COST, RESL
00080  LET STRUCTURE TD BE C1, C2; LET STRUCTURE C1, C2 BE N, O
00090  LET TABLE TABH BE MACH DOWN BY TH ACROSS TYPE INTEGER
00100  LET TABLE TABD BE DROW DOWN BY TD ACROSS TYPE INTEGER
00110  LET TABLE TABC BE MACH DOWN BY TC ACROSS TYPE INTEGER
00120
00130  READ TABLE TABH
00140  READ TABLE TABD
00150  READ TABLE TABC
00160  ENDMODULE
```

The data for the module are expected to take the form

```
4, 5, 6, 3, 4, 5, 100, 80, 5, 6, 7, 4, 5, 6, 110, 90
7, 6, 6, 6, 5, 5, 100, 90, 8, 7, 7, 7, 6, 6, 110, 100
3, , , 2, , , 40, 30, 4, , , 3, , , 50, 40
END TABLE
10, 10, 9, 11, 11, 10
25, 30, 30, 30, 25, 25
20, 20
1, 1, 1
, , , 2, 2, 1
END TABLE
2, 3, 4, 3, 4, 5, 3, 4, 5, 4, 5, 6
4, 3, 2, 5, 4, 3, 5, 4, 3, 6, 5, 4
1, , , 2, , , 2, , , 3
END TABLE
```

Annotation of the module MGTABS follows:

20-40 These statements declare the structure required to represent the columns of the table TABH; conceptually this has a tree structure, Figure 5.

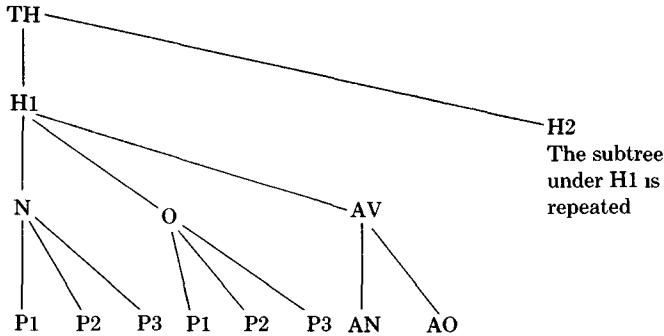
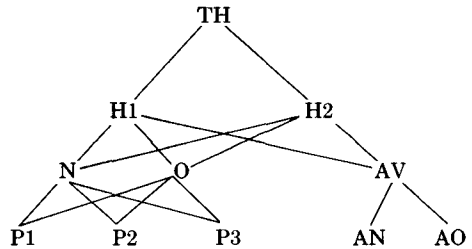


Fig. 5. Across structure TH for the table TABH.

Fig 6 Compact representation of the structure TH



However, because of the repetition of the subtree of H2, the network shown in Figure 6 is a more convenient representation of this structure.

All the structures used in the program are set out in the next section in this network form; these diagrams are useful for understanding the structure facility and its use in the program.

- 40-70 Declare all the structures necessary for the tables, TABH, TABC, TABD.
- 90-110 Declares the structure of the data tables and also declares a data input mode in which the data tables can be filled in.
- 130-150 The data tables are input in free format.
- 160 Signals the end of the module MGTABS.

It is natural to define implicitly the structures of the rows and the columns of the tables at the time of preparing the input data tables. Therefore a display format specification of the structure by using the input data tables is also possible in the UIMP system. The program module FORMTABS below illustrates this alternative way of carrying out the task of MGTABS module.

```

00010 MODULE FORMTABS
00020
00030 TABLE TABH TYPE REAL
00040 ACROSS TH
00050 (1) H1 , H2
00060 (2) N , O , AV ; N , O ; AV
00070 (3) P1,P2,P3;P1, P2,P3;AN,AO;P1,P2,P3;P1, P2,P3;AN,AO
    
```

```

00080 DOWN MACH
00090         (1) M1 = 4, 5, 6, 3, 4, 5,100, 80, 5, 6, 7, 4, 5, 6,110, 90
00100         (1) M2 = 7, 6, 6, 6, 5, 6,100, 90, 8, 7, 7, 7, 6, 6,110,100
00110         (1) M3 = 3, , , 2, , , 40, 30, 4, , , 3, , , 50, 40
00120 ENDTABLE
00130
00140 TABLE TABC TYPE REAL
00150 ACROSS TD
00160 (1)                C1      ,      C2
00170 (2)                N      , 0 ; N      , O
00180 (3)                P1,P2,P3;P1,P2,P3;P1,P2,P3;P1,P2,P3
00190 DOWN MACH
00200         (1) M1 = 2, 3, 4, 3, 4, 5, 3, 4, 5, 4, 5, 6
00210         (1) M2 = 4, 3, 2, 5, 4, 3, 5, 4, 3, 6, 5, 4
00220         (1) M3 = 1, , , 2, , , 2, , , 3, ,
00230 ENDTABLE
00240
00250 TABLE TABD TYPE REAL
00260 ACROSS TD
00270 (1)                T1      , T2
00280 (2)                P1,P2,P3;P1,P2,P3
00290 DOWN DROW
00300         (1) PRICE          = 10, 10, 9, 11, 11, 10
00310         (1) DEM           = 25, 30, 30, 30, 25, 25
00320         (1) STR (2) CAP   = 20, 20, , , ,
00330         (2) COST         = 1, 1, 1, , ,
00340         (2) RESL         = , , , 2, 2, 1
00350 ENDTABLE
00360
00370 ENDMODULE

```

5. THE ILLUSTRATIVE EXAMPLE FULLY CODED AND ANNOTATED

To complete the description of the MG and RW programs written in UIMP it is necessary to describe the four separate program modules that are used for this task. These modules are

MGTABS
 MGEN
 RWTABS
 REPORT

The module MGTABS was annotated in Section 4. In this section the three remaining modules are set out and their steps are annotated.

```

00380 MODULE MGEN
00390
00400 GET TABLE TABH, TABD, TABC FROM FORMTABS
00410
00420 LOAD TABLE TABH FROM FORMTABS
00430 LOAD TABLE TABD FROM FORMTABS
00440 LOAD TABLE TABC FROM FORMTABS
00450
00460 LET STRUCTURE TA BE A1, A2
00470 LET STRUCTURE A1, A2 BE AN, AO

```

```

00480 LET VARIABLE KP, JW, IT, LM, JA, IA, JC, IC BE TYPE STRUCTURE
00490 LET CLASS MODVAR PROD BE MACH BY TC
00500 LET CLASS MODVAR DEMAND BE TD
00510 LET CLASS MODVAR STORE BE TD
00520 LET CLASS MODCON AVAIL BE MACH BY TA
00530 LET CLASS MODCON BAL BE TD TYPE EQ
00540 LET MODCON OBJ BE TYPE FR
00550
00560 FOR LM IN MACH DO
00570 @ FOR IC IN TC
00580 @ ANDFOR IT IN TD
00590 @ ANDFOR ITA IN TA DO
00600 @ BEGIN FOR KP IN JW IN IC DO
00610 @ CODE:PROD(LM, KP) := (IT) & (JW) & (KP) & (LM)
00620 FOR JW IN IA DO
00630 CODE:AVAIL(LM, JW) := (IT) & (LM) & (JW)
00640 END
00650
00660 FOR KP IN IT IN TD DO
00670 @ BEGIN CODE:DEMAND(KP) := (IT) & (KP) & 'D'
00680 CODE:STORE(KP) := (IT) & (KP) & 'STR'
00690 CODE:BAL(KP) := (IT) & (KP) & 'ST'
00700 END
00710
00720 CODE:OBJ := 'PROFT'
00730
00740 @ ROW OBJ
00750 @ IS SUM KP IN JW IN IT IN TC
00760 @ SUM LM IN MACH
00770 @ (TABD(PRICE, KP IN TD(I:IT)) - TABC(LM, KP)) TIMES
00780 @ PROD(LM, KP)
00790 @ MINUS SUM KP IN TI IN TD
00800 @ TABD(COST IN STR, KP) TIMES STORE(KP)
00810 @ PLUS SUM KP IN T2 IN TD
00820 @ (TABD(RESL IN STR, KP) - TABD(PRICE, KP)) TIMES
00825 STORE(KP)
00830
00840 FOR JA IN IA IN TA
00850 @ ANDFOR JW IN IT IN TH
00860 @ ANDFOR JC IN IC IN TC DO
00870 @ BEGIN FOR LM IN MACH DO
00880 @ ROW AVAIL(LM, JA)
00890 @ IS SUM KP IN JW
00900 @ TABH(LM, KP) TIMES PROD(LM, KP IN JD)
00910 @ RHS 'RHS' VALUE TABH(LM, JA IN AV IN IT)
00920 END
00930
00940 FOR KP IN T1 IN TD DO
00950 @ ROW BAL(KP)
00960 @ IS SUM LM IN MACH
00970 @ SUM JW IN C1 IN TC
00980 @ PROD(LM, KP IN JW)
00990 @ MINUS STORE(KP)
01000 @ MINUS DEMAND(KP)
01010 @ VARIABLE PROD(M1, KP IN N IN C1) PIVOTS OUT ROW
01020

```

```

01030 FOR KP IN T2 IN TD DO
01040 @ ROW BAL(KP)
01050 @ IS SUM LM IN MACH
01060 @ SUM JW IN C2 IN TC
01070 @ PROD(LM, KP IN JW)
01080 @ MINUS STORE(KP)
01090 @ MINUS DEMAND(KP)
01100 @ PLUS STORE(KP IN T1)
01110 @ VARIABLE PROD(M1, KP IN N IN C2) PIVOTS OUT ROW
01120
01130 FOR KP IN T1 IN TD DO
01140 @ COLUMN STORE(KP)
01150 @ BOUND 'LIM' UP VALUE TABD(CAP IN STR, KP)
01160
01170 FOR KP IN IT IN TD DO
01180 @ COLUMN STORE(KP)
01190 @ BOUND 'LIM' LO VALUE TABD(DEM, KP)
01200
01210 WRITE MODEL 'EXMAT'
01220 WRITE BASIS 'EXBAS'
01230
01240 SAVE VARIABLE DEMAND
01250 SAVE VARIABLE STORE
01260 SAVE CONSTRAINT AVAIL
01270 SAVE CONSTRAINT BAL
01280 SAVE CONSTRAINT OBJ
01290
01300 ENDMODULE

```

Module MGGEN generates the complete model and provides code names for all the variables and constraints used. The action of this module may be explained as follows:

- 400 Extracts from the UIMP data base tables TABH, TABD, and TABC introduced by the module FORMTABS.
- 420-440 Loads the tables created by FORMTABS.
- 460-540 Declarations of the model variables and constraints.
- 560-640 Assign the code names for the production variables x_{ijkl} (T1M1AN, etc.).
- 660-700 Assign the code names for demand and storage variables z_{ik} (T1P1D, etc.), y_{ik} (T1P1STR, etc.), and the stock balance constraint (T1P1ST, etc.).
- 720 Assigns code name to the objective row OBJ.
- 740-820 Generates the objective row PROFIT, which is equivalent to

$$\sum_{i=1}^2 \sum_{j=1}^2 \sum_{k=1}^3 \sum_{l=1}^3 (p_{ik} - c_{ijkl})x_{ijkl} - \sum_{k=1}^3 s_k y_{1k} + \sum_{k=1}^3 (r_k - p_{2k}) y_{2k}$$

- 840-920 Generates the machine availability row, AVAIL, which is equivalent to

$$\sum_{k=1}^3 t_{ijkl} \cdot x_{ijkl} \leq a_{il} \quad \text{for all } i, j, l.$$

940-1010 Generates the first stock balance constraint:

$$x_{1jhl} - y_{1k} - z_{1k} = 0 \quad \text{for } i = 1, \quad k = 1, 2, 3$$

and also sets up the partial starting basis. Note the use of the default convention of TIMES 1.

1030-1110 Generates the second stock balance constraint:

$$\sum_{j=1}^2 \sum_{l=1}^3 x_{2,jhl} + y_{1k} - z_{2k} = 0 \quad \text{for } i = 2, \quad k = 1, 2, 3$$

and similarly sets up the partial starting basis.

1130-1190 Sets up the upper and lower bounds on storage and production:

$$y_{1k} \leq h_k \quad \text{for all } k$$

$$z_{ik} \geq d_{ik} \quad \text{for all } i, k.$$

1210 Writes the complete model in MPS format.

1220

1240-1280 Saves various items on the UIMP database.

01310 MODULE RWTABS

01320

01330 GET STRUCTURE TD, MACH FROM MGEN

01340

01350 LET STRUCTURE RTROW BE MACH, DEM, TOT

01360 LET TABLE RTAB BE RTROW DOWN BY TD ACROSS TYPE REAL

01370

01380 T:T1 := 'SUMMER - PERIOD'

01390 T:T2 := 'WINTER -- PERIOD'

01400 T:P1 := 'NUTS'

01410 T P2 := 'BOLTS'

01420 T:P3 := 'WASHERS'

01430 T:DEM := 'DEMAND'

01440 T:TOT = 'TOTAL'

01450 T:M1 := 'MACHINE 1'

01460 T M2 := 'MACHINE 2'

01470 T:M3 := 'MACHINE 3'

01480

01490 SAVE TEXT TD

01500 SAVE TEXT RTROW

01510

01520 ENDMODULE

Module RWTABS creates the table RTAB necessary to reproduce the solution information as required by the report module. It also sets up the next commentary, using text attributes, so that they may be used in printing the report table.

01530 MODULE REPORT

01540

01550 GET ALL FROM MGEN

01560 GET ALL FROM RWTABS

01570

01580 LOAD TABLE TABD FROM FORMTABS

01590 LOAD TEXT TD FROM RWTABS

01600 LOAD TEXT RTROW FROM RWTABS

01610

01620 LET CSE BE TEXT

```

01630
01640 LET PROD HAVE ATTRIBUTE X
01650
01660 READ SOLUTION CASE CSE
01670
01680 PAGE FEED
01690 LINE FEED(3)
01700 PRINTTF('PRODUCTION SCHEDULE FOR TWO PERIODS', 14, 35)
01710 PRINT LINE
01720 PRINTTF('===== ===== === =====', 14, 35)
01730 PRINT LINE
01740 PRINTTF('SET OUT AGAINST DEMANDS', 23, 23)
01750 PRINT LINE
01760 PRINTTF('==== ===== =====', 23, 23)
01770 PRINT LINE
01780
01790 LET VARIABLE KP, IT, LM, IC BE TYPE STRUCTURE
01800
01810 FOR KP IN IT IN TD DO
01820 @BEGIN RTAB(TOT, KP) := 0.0
01830 RTAB(DEM, KP) := TABD(DEM, KP)
01840 END
01850
01860 FOR LM IN MACH DO
01870 @ FOR IT IN TD
01880 @ ANDFOR IC IN TC DO
01890 @ FOR KP IN IT DO
01900 @ BEGIN RTAB(LM, KP) := X.PROD(LM, KP IN N IN IC)
01910 @ + X.PROD(LM, KP IN O IN IC)
01920 RTAB(TOT, KP) := RTAB(TOT, KP) + RTAB(LM, KP)
01930 END
01940
01950 PRINT TABLE RTAB DOWN TEXT F1 ACROSS TEXT F2 BODY F3
01960
01970 LET F1 BE DOWN FORMAT
01980 @ (0) FIELD(0, 0)
01990 @ (1) FIELD(3, 10) (SKIP(0, 3)),
02000 @ (2) SKIP(0, -10) FIELD(3, 10)
02010 LET F2 BE ACROSS FORMAT
02020 @ (0) FIELD(0, 0),
02030 @ (1) FIELD(3, 20) (SKIP(1, 2)),
02040 @ (2) FIELD(3, 7) (SKIP(0, 1))
02050 LET F3 BE BODY FORMAT DECIMAL(1)
02060
02070 LINE FEED
02080 PRINTTF('NB. ALL PRODUCTION IN 1000 LBS WEIGHT OF IRON', 1, 46)
02090 PRINTLINE
02100
02110 ENDMODULE

```

5.1 This Module

This module prints the actual report.

- 1660 Reads the solution generated by the LP OPTIMIZER.
- 1680-1770 Prints out the report headings.
- 1810-1840 Initializes the report table RTAB by zeroing the total row and copying the demand row from TABD.

- 1860-1930 Fills in the various entries in the report table by entering the combined normal and overtime solution values in the MACH fields, and accumulating the total in the TOT field.
- 1950 Prints the complete table, according to the formats F1, F2, and F3 specified on lines 1970, 2010, and 2050.
- 1970-2000 Specifies the down format F1.
- 2010-2040 Specifies the across format F2.
- 2050 Specifies the body format F3.

5.2 Printed Report

The following commentary briefly explains the operation of the formatted table print facility, used in the Report Module on line 1950. For a complete description of the use of table formats the reader is referred to [13, Chap. 9].

In order to describe the effects of UIMP formats on printing tables, a notional cursor, representing a character position on the line printer page, is used. Printing actually takes place within a rectangular area, known as a FIELD, the position of the field being indicated by the cursor. The size of field associated with each printable entity is specified in the format definition.

Consider the DOWN FORMAT, F1 (line 1970)

- (i) A printing field is defined by FIELD (m, n) where m represents the number of rows and n the number of columns. A null field may be defined by FIELD (0, 0).
- (ii) Level 1. A 3×10 field is used to print the text attributes of MACH, DEM, and TOT. Note T:MACH is output as a blank field.
- (iii) Level 2. The blank field of T:MACH is backspaced over by repositioning the cursor using SKIP (0,-10). Thus the blank field is overprinted with T:M1, that is, MACHINE 1.

The ACROSS and BODY formats may be similarly interpreted; a complete description of these formats may be found in [13].

5.3 An Extension of the Model

In order to illustrate the advantages of using an MGRW system, the following extensions of the problem are now considered. It is decided to investigate the maximization of revenue (rather than profit) at specified levels of profit. It is also intended to consider a few more products, say a total of ten.

5.3.1 *Revenue* The expression for revenue is

$$\sum_i \sum_j \sum_k \sum_l p_{ik} x_{ij,kl} + \sum_k (r_k - p_{2k}) y_{2k}.$$

Thus a revenue objective row is simply added to the model by the statements

```
03010 LET MODCON REVENUE BE TYPE FR
03020 CODE REVENUE = 'REVENUE'
03030 ROW REVENUE
03040 @IS SUM KP IN JW IN IT IN TC
03050 @ SUM LP IN MACH
03060 @ TABD (PRICE, KP IN TD (I:IT)) TIMES PROD (LM, KP)
03070 @PLUS SUM KP IN T2 IN TD
03080 @ (TABD(RESL IN STR, KP) - TABD(PRICE, KP)) TIMES STORE (KP)
```

5.3.2 *Profit*. To investigate different profit levels two new real variables are first introduced:

```
LET VARIABLE RPLEVEL, RPDELTA BE TYPE REAL
```

They should also be initialized to some desired values. The old objective row is then changed to an equality relation by the statement

```
ROW OBJ IS TYPE EQ RHS 'RHS' VALUE RPLEVEL
@'NEXTRHS' VALUE RPDELTA
```

In the optimizer, using a postoptimal parametric procedure, the $\text{RHS} + \text{THETA} \cdot \text{NEXTRHS}$ vector is investigated for the full range of values of the scalar multiplier THETA.

5.3.3 *Products*. The ten different products may be introduced by changing P1, P2, P3 in the structure declarations (line 00030 and line 00050 MGTABS) to P1, P2, . . . , P10 or

```
LIST I1 = 1 STEP 1 UNTIL 10/'P**'
```

In this case, the input table data need to be appropriately extended. However, the report modules do not need to be altered. This is because automatic pagination is carried out by the output PRINT TABLE statements of the tabulation subsystem, thus taking care of the increased table size.

6. CONCLUDING REMARKS

The UIMP translator is made up of around 9000 lines of FORTRAN source code. The FORTRAN coding for the executor library and for the database save and load facilities add up to another 7000 lines. For the CDC version the translator, after overlaying, runs within 56,000 words of memory. For arithmetic and logic statements, etc., one UIMP line of code after translation expands to anything between 3–8 lines of FORTRAN. For input/output commands, such as READ TABLE, PRINT TABLE, READ MODEL, WRITE MODEL, etc., only one line of FORTRAN subroutine call statement is generated to call the appropriate executor library subroutine. By means of a CDC Cyber 172 system the given matrix generator example was translated and, also, FORTRAN compiled in 6 CPU seconds. It was then executed in 4 CPU seconds, which includes linking and loading time or the generator program. For models up to 2500 nonzeros the generator computing cost and the report analysis computing cost with usual printing of tables, etc., are higher than the corresponding optimization computing cost of the CDC APEX system. For larger models this trend is progressively reversed.

The authors wish to make the following comments concerning the strengths and weaknesses of the system. In UIMP a novel and natural structure feature has been introduced to aid the construction of the LP models and the reports. The system is perhaps unique in that a modeler can use PIVOT and BOUND commands to specify a starting BASIS. In line with most other MGRW systems, it treats empty (undefined) data values implicitly. This means arithmetic with undefined data values leads to undefined data values and it ultimately leads to the deletion of a row or column of the matrix, as appropriate [13]. The system allows specification of the models either by rows or by columns. In the model communication commands there is a REJECT facility based on a logical test.

An important aspect of modeling strategy for the real-life problems is to define the sets (dimensions) that structure the model. By and large, the model coefficients and decision variables are defined in terms of these. In UIMP systems the tables, the model variables, and the model constraints are restricted to be at most two-dimensional. One argument put forward in favor of this restriction was that all data capable of visual input and output are presented as scalars or as one- or two-dimensional tables. Any higher dimensional table (array) is always adopted for presentation in some convenient two-dimensional form. In the UIMP system a multidimensional array is equivalently presented as a two-dimensional array by introducing multidepth structure and using element context enumeration. On hindsight this is the largest drawback of the system as this has led to complexity and lack of clarity, which could have been avoided. In a well-designed language a source program for matrix generation should also constitute a clear documentation of the model. In constructing models where the input data requires more than two-dimensional presentation, as in the example, UIMP system does not stand up to this test of clarity of documentation. The authors have found that an application programmer who is usually familiar with one of the established high-level procedural languages often shows resistance toward learning the syntax and semantics of yet another language. For modelers who fall into this category one alternative could be to implement the MGRW system

- (i) as a set of specialist procedures,
- (ii) or as a preprocessor with macrofacilities,
- (iii) or as a combination of (i) and (ii),

embedded in a host language such as FORTRAN or SIMULA.

Most LP models are constructed through the usual steps of

- (a) defining the sets—dimensions;
- (b) defining the coefficients, model variables, and model constraints in terms of these dimensions;
- (c) specifying the linear relationships (ROW or COLUMN statements) that connect those items mentioned in (b).

It is perhaps possible to devise a prompting mechanism and to realize an interactive system that can guide the modeler to construct the MG source program. A modeling system is most useful at the time of developing a model or a report-analysis program. At that stage such an interactive development aid may be of considerable value to the modeler.

APPENDIX A. A SELECTION OF UIMP COMMANDS

Data Input

READ TABLE	Read tables
READ LINE	Read line by line
READIF	Fixed format read (i.e., column numbers specified)
READRF	
READTF	
READIV	Variable format read (i.e., data separated by commas).
READRV	
READTV	

Report Output

PRINT TABLE	Print entire table
PRINTIF } PRINTRF } PRINTTF }	Create reports line by line
PRINT LINE	Print one line
LINE FEED	Skip one line
PAGE FEED	Skip one page

Model and Basis Generation

COLUMN	Generate by column
ROW	Generate by row
RHS	Generate right-hand side
RANGE	Generate ranges
BOUND	Generate bounds
PIVOTS	Generate starting basis

Control

GOTO	Unconditional jump
PERFORM	Transfer control
NEXT	Return control
STOP	Terminate execution
BEGIN ... END	Compound statement
IF ... THEN ...	Conditional statement
FOP } SUM }	Loop control
AND FOR	Parallel loop control

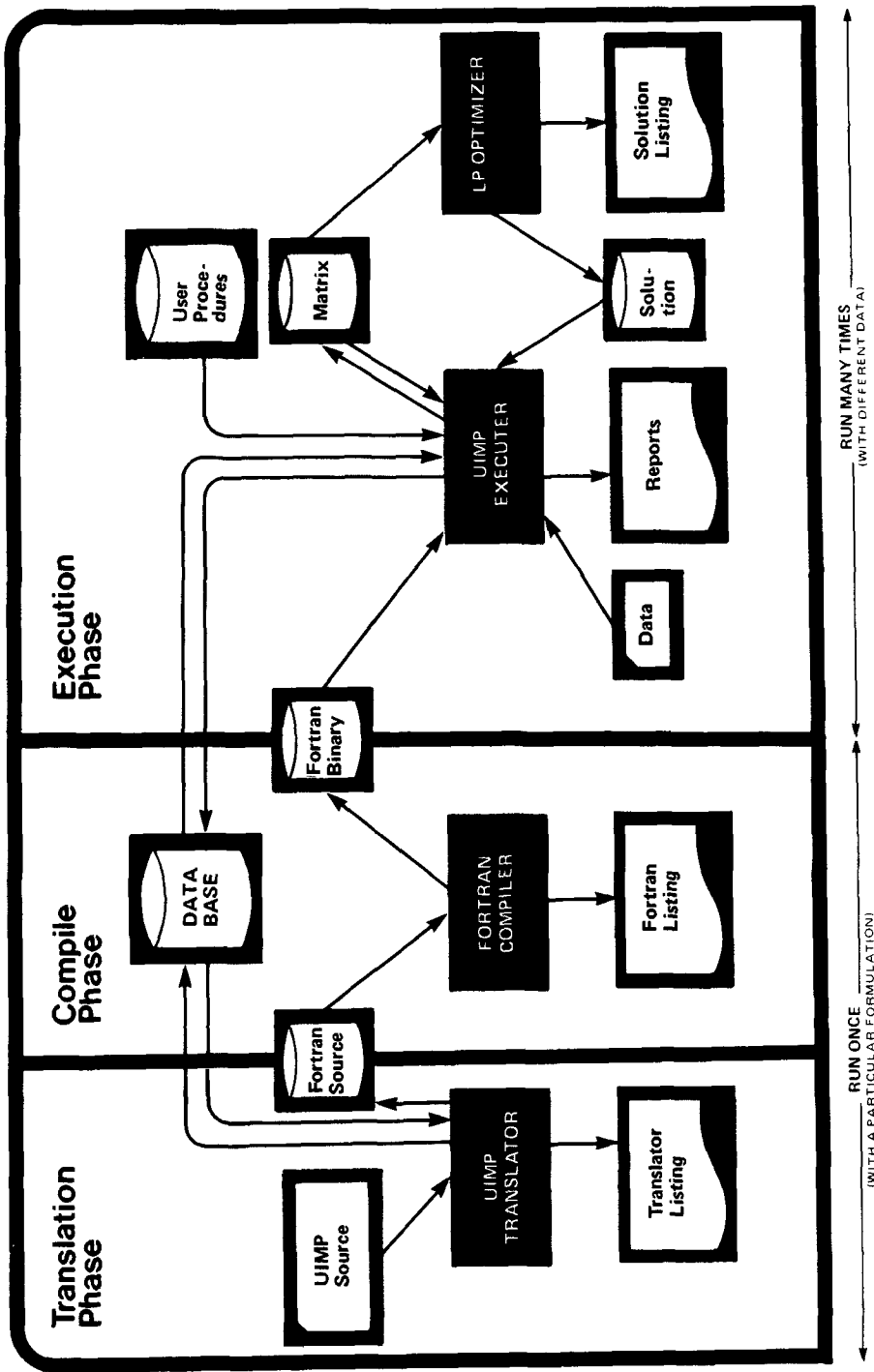
Database Communication

GET STRUCTURE	
GET TABLE	
GET MODVAR	Repeat declarations made in another module (at transate time)
GET MODCON	
GET ALL	
SAVE TEXT } SAVE TABLE } SAVE ALL }	Save data (at execution time)
LOAD TEXT } LOAD TABLE } LOAD ALL }	Load data (at execution time)

Model Communication

READ MODEL	Read matrix
READ SOLUTION	Read solution
REJECT	Ignore current row/column
WRITE BASIS	Write starting basis
WRITE MODEL	Write matrix

APPENDIX B. THE UIMP SYSTEM



REFERENCES

1. AMERICAN NATIONAL STANDARDS INSTITUTE *American National Standard Programming Language COBOL* ANSI X 3 23, New York, 1974
2. HAVERLY SYSTEMS *MAGEN Reference Manual* 1977
3. IBM CORPORATION *Mathematical Programming System Extended (MPSX)*. Program 5734 XM4, 1971
4. IBM CORPORATION *MGRW Program Reference Manual* Program SH19-5014, 1977.
5. INTERNATIONAL COMPUTERS LTD *Matrix Generator and Report Writer: MGRW 2900*. 1979.
6. KERNIGHAN, B W , AND PLAUGER, P J *Software Tools* Addison-Wesley, Reading, Mass., 1976.
7. KETRON INC *MPS III DATAFORM User Manual* 1975.
8. KNUTH, D E *The Art of Computer Programming*, vol 1, 2nd ed. Addison-Wesley, Reading, Mass , 1973
9. SCICON COMPUTER SERVICES *MGG User Guide, RWG User Guide*. 1975.
10. SPERRY UNIVAC COMPUTER SYSTEMS *GAMMA 3 4 Programmer Reference*. No. UP-8199, Rev. 1, 1977
11. UNICOM CONSULTANTS LTD *A Brief Survey of the Currently Available Matrix Generator Report Writer System* 1974
12. UNICOM CONSULTANTS LTD *User Manual for UIMP*. 1977
13. UNICOM CONSULTANTS LTD *Reference Manual for UIMP*. 1977

Received March 1982, revised May 1982, accepted May 1982