NOT FOR QUOTATION
WITHOUT PERMISSION
OF THE AUTHOR

On the problem of designing matrix generators

V.Yu. Lebedev

September 1982
CP-82-53

Collaborative Papers report work which has not been performed solely at the International Institute for Applied Systems Analysis and which has received only limited review. Views or opinions expressed herein do not necessarily represent those of the Institute, its National Member Organizations, or other organizations supporting the work.

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

On the problem of designing matrix generators

V. Yu. Lebedev

Computing Center of the USSR Academy of Sciences, Moscow

1. INTRODUCTION

One of the main difficulties which arises when using a general purpose linear programming (LP) package is the need to prepare a special code to transform initially structured model data into a format appropriate for subsequent use by an LP solution system (for example, at IIASA we require an MPS-type input file for use with the MINOS LP system). These codes are usually in general purpose programming languages, are often rather complex, and are generally quite difficult to implement. There is a lack of advanced software capable of simplifying the preparation of information in the form required by the LP package, and this is a problem especially for those who use linear models in the solution of a wide variety of applied problems. In such cases it is highly desirable to have a flexible and convenient interface between user and LP software.

Here we describe a tool which saves time for the analyst by making it possible to organize investigations with linear models in an interactive manner.

There are many programming systems which have been designed to overcome the interface problem described above (see, for example [1] and [2] for a general discussion). Here we consider one more system of this type, which has been designed and implemented at the Computing Center of the Academy of Sciences of the USSR. This system is based on different principles to most of the others; and we shall discuss these principles here as they are not difficult to implement and lead to quite convenient universal systems.

2. ON APPROACHES TO THE DESIGN OF MATRIX GENERATORS

There are various ways of designing matrix generating software which provides the user with a convenient general purpose means of describing the initial data for LP models while the transformation of this description into the correct input format for some solution package is performed automatically. One approach is to start from the block structures of matrices in LP problems and develop advanced techniques for the description and combination of these blocks [3]. Another possibility is to use linear algebraic vector notation or even the conventional scalar notation of mathematical formulas as a basis for the input language of some matrix generator. In each case the generator would produce the same result - some record similar to an MPS file. However, the ways in which this record is created may be quite different.

The most immediate way is to generate the final file by translating the initial description of the problem in an interpretive manner. This is a traditional method of organizing the generation process and it has two essential disadvantages. Firstly, the higher the level of the input language, the greater the difficulties in implementation, so that the development of a convenient general purpose generator proves to be very complicated. Secondly, interpretation usually leads to a reduction in efficiency and therefore interpreting generators often work slowly. For these reasons we suggest another approach to the design of matrix generators.

Suppose that for a given LP problem we have a routine capable of calculating the values of the constraining functions for given values of the variables and another routine to calculate the (analytical) derivatives of these functions. These routines may be implemented in some high-level general purpose programming language and their structure may be closely connected to that of the problem considered. This means, in particular, that the variables may be subdivided into numerical arrays of different dimensions. The results obtained with the second routine are therefore placed in certain positions within these arrays. We can now design a multistep procedure for generating a compact (without zeroes) record of the problem, carrying out the following calculations at each step:

1) Set the value of all variables to zero.

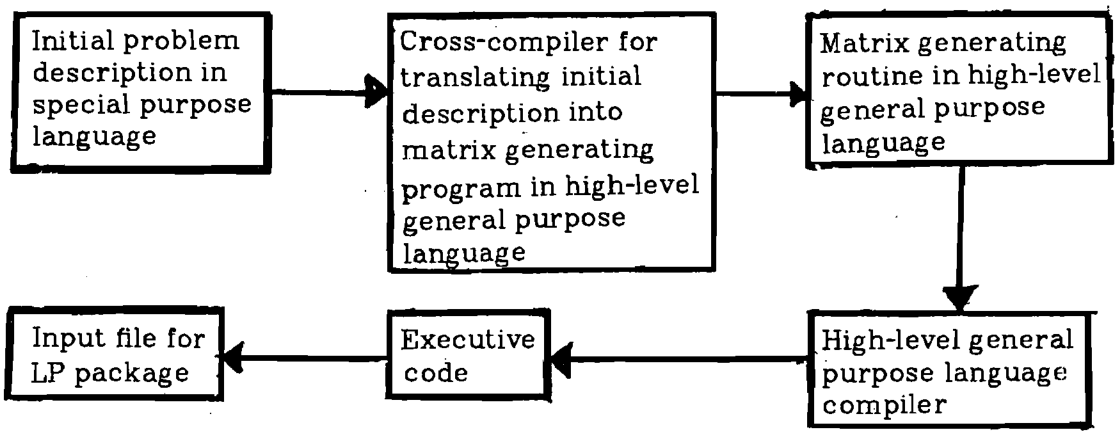
2) Calculate the value of the next constraining function (these functions are examined in turn). As a result, the value of the right-hand side of this constraint will be obtained. Put the value obtained and the index of the constraint under consideration into some intermediate file.

3) Calculate the derivative of the next constraint; by successively looking through each array of variables find all nonzero coefficients of this constraint and put them (identified by the indices of the constraint and the variable) into an intermediate file.

The intermediate file created by this procedure may easily be transformed into an MPS file or something of a similar kind. The structure of the routine performing this transformation has nothing to do with the structure of the LP problem at hand and there is therefore no need to redesign it for each new LP problem that arises.

This method for designing special matrix generating routines for individual LP problems is quite efficient and relatively easy to automate. To be more precise, it is less difficult to translate the description of the LP problem in some special high-level language into routines for calculating constraints and their derivatives and to design an automatic compilation of the matrix generating procedure than to devise a method for the immediate transformation of the same description into an input file suitable for use by an LP package.

The savings in time and effort associated with this technology stem from the fact that when translating the initial description of the problem into the high-level general purpose programming language of the matrix generating routine, it is not necessary to solve the complex programming problems that inevitably arise if it is attempted to translate this description directly into a format acceptable to an LP package. In fact we are simply taking advantage of the solution of this problem already found by those programmers who implemented the general purpose language used as an intermediate object language. Thus the whole cycle of matrix generation may be described as follows:


Initial problem description in special purpose language
Cross-compiler for translating initial description into matrix generating program in high-level general purpose language
Matrix generating routine in high-level general purpose language
High-level general purpose language compiler
Executive code
Input file for LP package

3. MAGISTR SYSTEM

The approach to the design of matrix generating systems outlined above has been implemented on a BESM-6 computer at the Computing Center of the USSR Academy of Sciences. The resulting programming system is called MAGISTR. ALGOL-60 has been chosen as the basic general purpose language for this system, i.e., ALGOL-60 is used as an object language by the cross-compiler that translates the initial description of the LP problem into matrix generating routines, and these routines are therefore ALGOL-60 codes.

The cross-compiler itself has been implemented in ALGOL-60 with some negligible assembly language impurities which are necessary to perform some simple string processing. Conventional scalar notation of mathematical formulas has been chosen as the basis for the language used to describe LP problems.

There are two reasons for this choice. Firstly, this is the most flexible type of language for describing mathematical problems. Secondly, this language is very close to some of the more widely used general-purpose calculation-oriented programming languages such as FORTRAN and ALGOL.

The description of an LP problem in the input language of the MAGISTR system is composed of three paragraphs.

The first paragraph contains a description of the parameters and variables of the problem. All of these are data structures of ALGOL-60 type and therefore this paragraph looks like a chain of ALGOL declarations.

The next paragraph includes a description of the constraints and objective function. This paragraph is actually quite similar to the conventional algebraic record of a problem in scalar notation. To define the constraints of a given problem we use sentences (in input language) which are exact analogues of algebraically correct formulas defining linear combinations of variables. These can be written, for example, using the concept of summation over an index ($\sum_{i=1}^{n}$ and so on). It is then only necessary to change the summation notation by simple input language constructions (of which there is an exhaustive set) and to rewrite the initial index expressions in ALGOL format.

The third paragraph consists of instructions that must be performed before the main generating procedure can be initiated. These may include preliminary data transformations, for example, 'read' operators. To arrange any cyclical operation or summation over an index it is possible to use the same special input language constructions as in the previous paragraph; instructions in ALGOL-60 may also be given here.

Once we have routines for calculating constraints and their derivatives, it is easy to compile a program for an approximate solution of the problem using any penalty function technique; this compilation can also be done automatically. It would be unreasonable to neglect such an opportunity for automation and indeed compilation in the MAGISTR system is now fully automatic.

Thus, with the aid of MAGISTR a user may obtain a compact record of the input data in a format similar to that of an MPS file and then solve the problem with the aid of a general purpose LP package; he could also obtain an approximate solution of his problem using some specific penalty function algorithm. In either case he provides the system with the same structural description of his problem.

We have now been using MAGISTR for about three years, during which time many widely different large-scale LP problems have been solved. These include dynamic multisectoral balance optimization problems, static problems in the optimal allocation of agricultural production, problems of water supply, and so on. Our experience of the system leads us to conclude that the principles behind the design of MAGISTR are sufficiently valid to give good results in practice.

The system's input language can be learned by users quickly and without any significant difficulty, and at the same time provides them with a wide spectrum of convenient features for brief and natural problem descriptions. It is worth mentioning here that MAGISTR performs a syntax analysis of all specific input language constructions. All other constructions are checked by the ALGOL compiler and there is no difficulty in connecting an error detected in this way with the text of the initial description of the problem. Consequently there are no serious difficulties in debugging texts in MAGISTR input language.

The matrix generating technology implemented in the MAGISTR system proved to be quite efficient: the generating process (including the compilation from MAGISTR input language into ALGOL-60 and from ALGOL-60 into executive code) rarely took more than 10 percent of the time necessary to find an optimal solution.

The implementation of the MAGISTR system has taken about one man-year of effort. We are now considering developing a new version of this system, in which FORTRAN will be the intermediate object language and a new cross-compiler will be implemented in PASCAL.

4. ILLUSTRATIVE EXAMPLE

In order to give the reader an opportunity to appreciate more fully the possibilities of the language in the MAGISTR system, we shall consider an example. We shall compare the initial algebraic description of a given LP problem with the equivalent description in MAGISTR language. Let the problem be Leontief's dynamic optimization model. The mathematical formulation of this problem is as follows:

$$\sum_{t=0}^{T-1} P(t)w(t) = \max$$

$$x_i(t) - \sum_{j=1}^{N}(a_{ij}x_j(t) + \sum_{k=0}^{L} b_{ij}(k)u_j(t-k)) - w(t)c_i = 0;$$

$$y_i(t) - x_i(t) \geq 0;$$

$$y_i(t+1) - y_i(t) - u_i(t) = 0;$$

$$i = 1,2,...,N;\ t = 0,1,...,T-1;$$

$$y_i(0) - y0_i = 0;\ i = 1,2,...,N,$$

where

$$x_i(t),\ u_i(t),\ i = 1,2,...,N;\ t = 0,1,...,T-1;$$

$$y_i(t),\ i = 1,2,...,N;\ t = 0,1,...,T;\ w(t),\ t = 0,1,...,T-1,$$

are the variables and the other symbols represent the parameters of our problem. We assume that N=10, T=10, L=5. The description of this problem in MAGISTR input language is as follows

```
*
* This is the declaration of the problem
*
1=N(10),T(10),L(5);            * declaration of integer scalar
                             * parameters with simultaneous
                             * initialization
4=a[1:N,1:N],c[1:N],           * declaration of real vector
    b[0:K,1:N,1:N],p[0:T-1]; * parameters
1=x[0:T-1,1:N],y[0:T,1:N],       * declaration of  variables
    w[0:T-1],u[0:T-1,1:N];
3=BAL,CAP,GR[0:T-1,1:N];        * declaration of array of constraints
4=t,i,j,k;.                   * declaration of indices
*
* This is the description of the constraints
*
2='t:=0 T-1''i:=1 N'(
    BAL[t,i]:=x[t,i]-'j:=1 N'(
        a[i,j]*x[t,j]+'k:=0 L'b[k,i,j]*u[t-k,j]
        )
    -w[t]*c[i]=;
    CAP[t,i]:=y[t,i]-x[t,i];
    GR[t,i]:=y[t+1,i]-y[t,i]-u[t,i]=;
    );
 'I:=1 N'IN[i]:=y[0,i]-y0[i]=;.
*
* This is the description of the objective function
*
='t:=0 T-1'p[t]*w[t];.
*
* These are the instructions which should be executed
* before the main generating procedure starts
*
1=read(a,b,c,p);.
```

ACKNOWLEDGEMENT

The author wishes to thank Dr. E. Nurminski for his valuable criticism and help in preparing this paper.

References

1. J. Bisschop and A. Meeraus, "Toward Successful Modelling Applications in a Strategic Planning Environment," in *Large-Scale Linear Programming: Proceedings of a IIASA Workshop, 2-6 June 1980*, ed. G.B. Dantzig, M.A.H. Dempster and M. Kallio,International Institute for Applied Systems Analysis, Laxenburg, Austria (1981).
2. W. Orchard-Hays, "Problems of Symbology and Recent Experience," in *Large-Scale Linear Programming: Proceedings of a IIASA Workshop, 2-6 June 1980*, ed. G.B. Dantzig, M.A.H. Dempster and M. Kallio,International Institute for Applied Systems Analysis, Laxenburg, Austria (1981).
3. W. Orchard-Hays , "The Scope of Mathematical Programming Systems," pp. 27-102 in *Design and Implementation of Optimization Software*, ed. H.J. Greenberg,Sijthoff and Noordhoff (1978).