AN INTELLIGENT SYSTEM FOR FORMULATING LINEAR PROGRAMS

Frederic H. Murphy
School of Business
Temple University
Philadelphia, Pennslyvania

and

Edward A. Stohr
Graduate School of Business Administration
New York University
New York, New York

August 1985

Center for Research on Information Systems
Computer Applications and Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #95

GBA #85-40

Accepted for publication by **Decision Support Systems**.

This work was carried out as part of a jointly-defined research study on expert systems with the IBM Corporation.

ABSTRACT

The research and system development work described in this paper is aimed at overcoming some of the problems associated with the development of large, complex linear programming problems. The most overwhelming problem is that of size. It is not uncommon for large planning and policy analysis problems to have tens of thousands of constraints and activities. Matrix generator systems have been designed to help in this process. However, the amount of manual labor involved is still very great and the formulation process is subject to errors which are difficult to detect. We provide an overview of a system which uses artificial intelligence and database techniques to help a knowledgeable user formulate large linear programs. The system automates many of the tedious processes associated with large-scale modeling and provides a top-down development environment with a number of different forms of problem representation.

AN INTELLIGENT SYSTEM FOR FORMULATING LINEAR PROGRAMS

1. INTRODUCTION

This paper describes a partially implemented system for formulating linear programs (LPs) using artificial intelligence techniques. Given a mathematical program of the form:

(1) maximize cx

subject to:

$$Ax \leqslant b$$

$$x \geqslant 0,$$

the goal of the formulation process is to provide numerical values for c, A, b. Currently, formulating linear programs is treated as an art where one formulates many examples illustrating the possibilities that may occur in practice. After sufficient experience with the examples, one starts to recognize patterns that apply to new problems. Beyond the abstract formulation step where one decides the class of models within which the new one fits, there are the grueling steps of constructing the model, developing labeling schemes for the rows and columns and organizing the data that determine the coefficients.

In this paper we describe the nature of the formulation process and the techniques for formulating a linear program using an intelligent system. The LP Formulator employs artificial intelligence (AI) techniques to simplify the problem formulation process. It is designed to handle a broad class of linear programming problems but will be particularly useful in large

scale systems where there are many submodels. Initially, the system will be most suitable for expert users; eventually we hope that it will become intelligent enough to help managers or students with a minimal exposure to linear programming techniques.

As far as we know no system has yet been developed that employs expert system techniques to aid in the formulation of linear programs. However, there have been many data generators for specific problem types - for example, see Gershon [6]. In an alternative approach, Slate & Spielberg, [14], provide a general purpose PL/1 front-end to IBM's mathematical programming package, MPSX, [12]. The LOGS system, (Brown, et al, [3]) provides a powerful modeling language for specifying logistics planning problems. Our research complements the work done by Greenberg in computer-aided analysis, [8], [9]. He develops tools and mathematical techniques to help understand LPs after they have been formulated. Our approach is to specify and use the structural properties of the problem during the formulation process. A closely related expert systems project which concentrates on formulating LPs for production planning is reported in Binbasioglu and Jarke, [2].

Linear programs range in size from small paradigm problems to massive production/distibution problems running into hundreds of thousands of activities and rows. The larger problems contain small models replicated in dimension and/or linked through some sort of network structure. The art of formulating large linear

programs involves distinguishing the small paradigm problems and characterizing the linkages.

An intelligent system for formulating linear programs should perform the following functions:

- ease the process of formulation
- divide large systems into comprehensible subunits
- simplify the activity and constraint labeling process
- provide logic checks on the formulation
- generate input for a matrix generator or feed directly into a solver.
- facilitate debugging and generation of alternative models

More than anything else we hope that our system will help users overcome the problems arising from the complexity of real world applications of linear programming.

Figure 1 shows the components of the prototype software system that we are building.

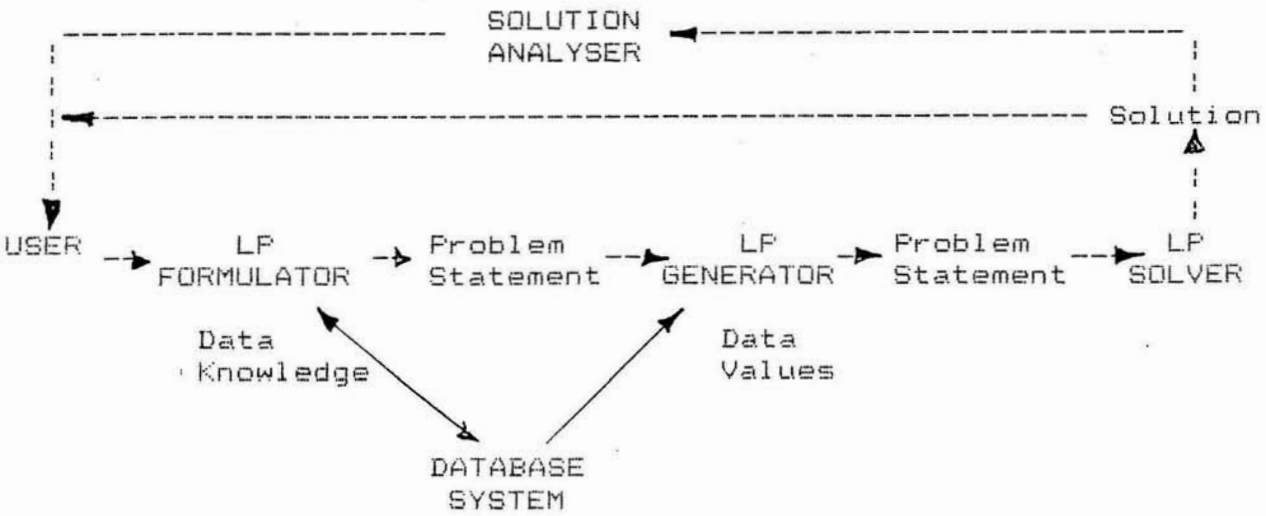


FIGURE 1
Software Components

This is a loosely coupled system that takes advantage of the existence of four powerful existing systems:

- An LP Generator, [15] (this is similar in function to the popular OMNI system, [10]).
- IBM's MPSX system for solving linear and integer mathematical programs, [12].
- IBM's SQL database management system (DBMS),[1].
- A tableau solution analyser (ANALYZE, [8]).

The five software systems in Figure 1 communicate by passing files. It is easy to imagine more integrated designs and other opportunities for the employment of AI techniques - particularly for model validation and sensitivity analysis.

The LP Formulator is being developed in PROLOG on an IBM 4341 computer. PROLOG is a logic-programming language that has been found useful in a number of 'expert system' projects ([4]). The knowledge in the LP Formulator consists of a number of rules relevant to the formulation of LP problems. This knowledge is not specific to any given application. Specific application knowledge and data values for the coefficients in the LP tableau are stored in the DBMS. To simplify the exposition however we assume that the LP problem structure is explicitly defined in an interaction with the user of the system rather than being stored in the DBMS.

Section 2 describes the philosophy underlying our approach. An example in the domain of energy modeling is introduced. Section 3 describes the problem representations available to users and maintained internally by the system. Section 4 illustrates the logic of the rule-based system in formulating a

small transportation problem. Finally, Section 5 presents a summary and outlines future work.

2. PHILOSOPHY UNDERLYING THE LP FORMULATOR

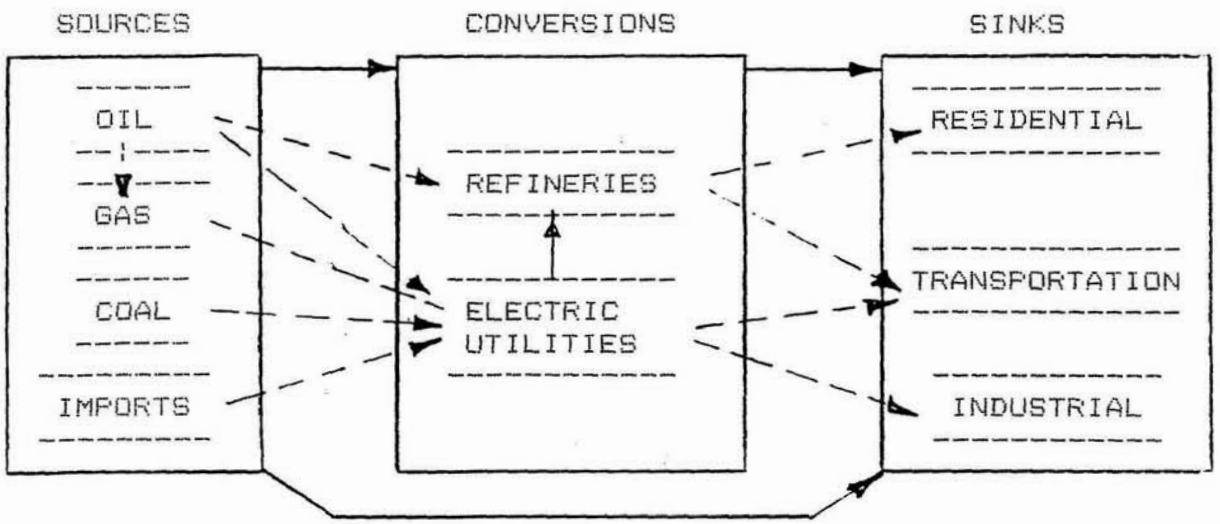
Large linear programs typically have 90 percent or more zeroes in the A matrix and the bulk of the nonzeroes are 1's. This results from the dominance of network substructures. One can take advantage of the networks to visualize the problem and focus attention on the component submodels. For example, the standard representation of the PIES energy policy model, [13], is a diagram that parallels the flows in the energy system. The intelligent system helps categorize the types of linkages and the submodels that are connected.

Activities in a linear program represent transformations in either form, time or space. Transformations in form are the most complex. These occur, for example, in product mix problems where resources such as labor, capital equipment and materials are transformed by the activities into products. Inventory holding activities are examples of transformations in time while transportation activities are examples of transformations in space. Apart from the transportation model and a few manpower planning and scheduling problems, the paradigm models are typically transformations in form.

As a first step in formulating a linear program we isolate (as 'blocks') the submodels that are form transformations and connect them with arcs that represent the existence of some kind of linkage in place or time. This is the first of four goals

that must be reached to identify the values of c, A and b. The second through fourth are to specify the transportation submodels, the interperiod linkages and the physical transformations.

Figure 2 shows the first two levels of refinement of the PIES model.



First level
Second level (not all connections are shown)

FIGURE 2

The first level simply asserts that the model contains sources, conversions and sinks. Sources connect to conversions and to sinks; conversions connect to sinks. Already, the form of the resulting LP tableau has been constrained as we will show momentarily. The second level specifies the kinds of sources, conversions and sinks together with their interconnections. In PIES there are submodels that represent the supply of various types of energy; others that transform one or more forms of

energy into another and finally submodels that represent the consumption of energy. The arcs in Figure 2 represent the physical flows of energy forms. Note that at this point we have a systems diagram but not a transshipment model or a netform as defined by Glover et al, [7].

An interactive session with the LP Formulator might begin as follows (computer prompts are followed by a question-mark):

Create-blocks? sources,sinks,conversions

Indicate links for sources? sinks,conversions
Indicate links for sinks? none
Indicate links for conversions? sinks

At this point the system could display the first level graphic in Figure 2. In terms of the underlying linear programming model we have the information shown in (2) below.

(2)
Minimize

$$\sum_{\substack{i \in SO \\ u \in PSO}} c_{ui}x_{ui} + \sum_{\substack{i \in SO \\ j \in CO \\ q \in OSO}} c_{qij}t_{qij} + \sum_{\substack{j \in CO \\ v \in PCO}} c_{vj}x_{vj} + \sum_{\substack{i \in SO \\ k \in SI \\ q \in OSO}} c_{qik}t_{qik} + \sum_{\substack{j \in CO \\ k \in SI \\ s \in OCO}} c_{sjk}t_{sjk} + \sum_{\substack{k \in SI \\ w \in PSI}} c_{wk}x_{wk}$$

subject to:

$$\sum_{u \in PSO} a_{pui}x_{ui} \leqslant b_{pi}$$

$$\sum_{u \in PSO} a_{qui}x_{ui} - \sum_{j \in CO} t_{qij} - \sum_{k \in SI} t_{qik} \geqslant -b_{qi}$$

$$-\sum_{i \in SO} a_{qij}t_{qij} + \sum_{v \in PCO} a_{qjv}x_{jv} \leqslant b_{qj}$$

$$\sum_{v \in PCO} a_{rjv}x_{jv} \leqslant b_{rj}$$

$$\sum_{v \in PCO} a_{sjv}x_{jv} - \sum_{k \in SI} t_{sjk} \geqslant -b_{sj}$$

$$-\sum_{i \in SO} a_{qik} t_{qik} \qquad +\sum_{w \in PSI} a_{qkw} x_{kw} \leqslant b_{qk}$$

$$-\sum_{j \in CO} a_{sjk} t_{sjk} \qquad +\sum_{w \in PSI} a_{skw} x_{kw} \leqslant b_{sk}$$

$$\sum_{w \in PSI} a_{zkw} x_{kw} \leqslant b_{zk}$$

where:

SO = set of sources
PSO = set of production activities at the sources
OSO = set of outputs from the sources
CO = set of conversions
PCO = set of production activities at the conversions
OCO = set of outputs from the conversions
SI = set of sinks
PSI = set of production activities
p = index of production constraints for sources
q = index of outputs from sources
r = index of production constraints for conversions
s = index of outputs from conversions
z = index of production constraints for sinks

Although very little has been specified in the model, the linear programming representation is already complex. The complexity comes from having to distinguish among constraints for inputs and outputs and possible transformations internal to the blocks. The contrast in complexity between the problem representations in Figure 2 and the algebraic statement (2) illustrates the potential of the LP Formulator system. It will allow users to state their problems in a natural, graphic style and to concentrate their attention selectively on small sub-problems. The algebraic manipulation and book-keeping details will be performed by the system.

The PROLOG implementation will be described in a separate paper. It is sufficient here to describe the general strategy. The system initially knows that the LP has the form (1). As more knowledge of the problem structure is gained through interaction with the user the

formulation becomes more detailed. Thus the program successively attempts to refine problem statements but it gives the user the freedom to move to another goal before the current goal is met and to return to it at a more convenient time. The system achieves its final goal when the statement is detailed enough to allow all coefficient values to be retrieved from the DBMS and when the problem statement satisfies a number of other completeness and integrity checks.

The idea is to work from the general to the particular. The LP formulation at each stage is the most general one that is consistent with the information obtained so far. Thus each 'block' in the visual representation (Figure 2) is recursively associated with an LP of the form (1). Similarly each 'arc' is associated with a set of rows and columns that algebraically link the associated blocks. Each arc is also potentially associated with a bundle of different flows and may be capacitated or not. Properties of a higher level in the representation are inherited by lower levels.

3. PROBLEM REPRESENTATIONS

One of the advantages of an automated approach is that a number of different problem representations can be generated and displayed to the user during the problem formulation process. This is illustrated in the top half of Figure 3.

The first requirement is that the user should be able to recall and modify the problem definition in a form close to that used for original entry. This is the function of the 'Block Language' which is used to express the operations that can be performed on blocks in the graphics view of problem definition. As the work proceeds the user may

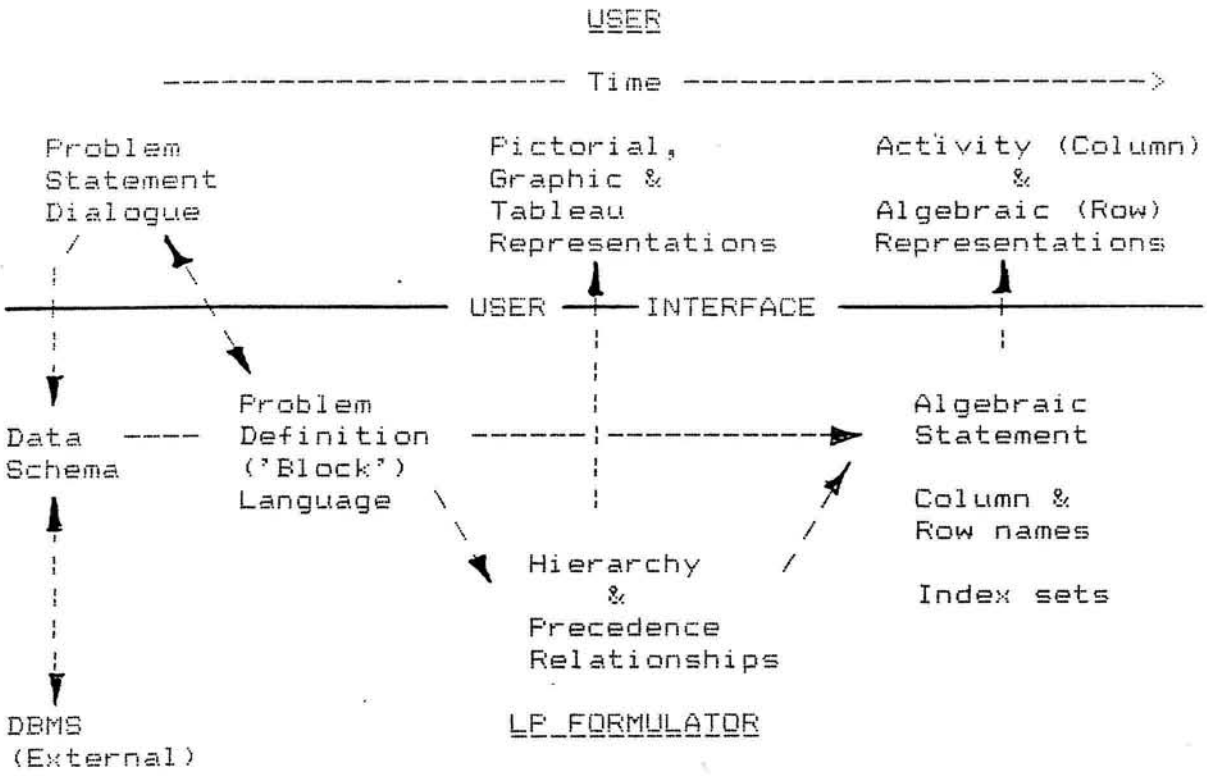


FIGURE 3
User Interface and Internal Problem Representations

wish to view a graphic showing the hierarchical and network structure of the problem. Alternatively a 'picture' of the tableau in a highly summarized form may be desired, [8]. Finally, the user may wish to inspect the algebraic statement of the problem (a format similar to (3) below) or to look at the coefficients in individual columns. The latter, 'activity analysis' approach, is advocated by Dantzig, [5]. It is useful because activities typically have fewer coefficients than rows (usually no more than 3 or 4).

Internally, the system maintains the data structures shown in the lower half of Figure 3. A rule-base internal to the system allows it to translate between these internal and external representations.

The data schema is described below. It forms an important part of the application knowledge base and greatly simplifies the task of the Formulator. The hierarchical and precedence relationships between problem components are maintained continuously. These allow a top-down approach to the problem definition and are the basis for structural analyses aimed at avoiding unboundedness and infeasiblities. The internal algebraic statement is equivalent to the normal 'sigma' notation of LP textbooks (see [15]). This is easily translated into the MPS format required by the LP Generator. Along with the equation forms of the LP, the system automatically generates and maintains the names of:

- column groups and individual columns (variables)
- constraint groups and individual rows
- index sets for the summations in objective and constraint rows

Actually, users share the responsibility for the names. The system supplies 2- or 3-letter abbreviations of the names input by users. These are generated according to certain fixed rules and checked against a continuously maintained data dictionary to avoid name clashes.

4. DETAILED DISCUSSION FOR A SIMPLE PROBLEM

In this section we illustrate the use of the above framework and rules in the definition of a classical transportation problem:

(3) Minimize $\sum_i \sum_j C_{ij}X_{ij}$

Subject to:

$$\sum_j X_{ij} \leqslant S_i$$

$$\sum_i X_{ij} \geqslant D_j$$

Figure 4 below gives the internal representation of the data schema and problem definition after interaction with the user.

Data_Schema

a. TRANS-COSTS(Vendor, Warehouse, C, $ per unit)
b. SUPPLY(Vendor, S, units)
c. DEMAND(Warehouse, D, units)

Problem_Definition_Statements

a. CREATE-BLOCKS(Trans-problem, [Vendors, Warehouses])
b. LINK-BLOCKS(ALL, [Vendors, Warehouses], X)
c. CREATE-BLOCKS(Vendors, Vendor = [v1..v3]])
d. CREATE-BLOCKS(Warehouses, Warehouse = [w1,w2])
e. MINIMIZE(Trans-costs)

FIGURE_4_
Stored Definition of a Transportation Problem

A data schema entry follows a relational format as follows:

Table-name(key-identifier, data-name, units-meta-data)

The table-name and key-identifier are names that actually occur in the 'real' external database system. The former identifies the external file (or relational table). The latter is a group of database fields that uniquely identify the data value in the data-name field. Thus in Figure 4 Trans-costs is the name of the file containing the transportation cost data for the problem. This file has three fields. The Vendor and Warehouse

fields uniquely identify a record in the file and the 'C' field contains the unit cost of transportation from each vendor to each warehouse. The name given in the data-name field is used in the algebraic problem formulation and may be an abbreviation or synonym for the corresponding field name in the external relation. The last field is used to check that the data for the problem is expressed in compatible units. This field does not occur in the real world file (since it would normally have the same value for every record). It must be supplied by the user or found automatically by the system from a data dictionary query.

Although this is a very restricted format for the data it is satisfactory for the initial prototype in which we are mainly concerned with developing the rules for formulating LPs. Later work will be directed towards generalizing the database interface.

We now explain the block-language statements in Figure 4 and illustrate the rules that are used to develop the LP problem statement.

The CREATE-BLOCKS operation refines the definition of blocks by creating sub-blocks at a new level of detail in the problem. The first argument specifies the block to be refined; the second provides the list of sub-blocks. In Figure 4 the first statement gives the problem the name 'Trans-problem' and specifies that Vendors and Warehouses blocks exist at the first level.

The LINK-BLOCKS operation creates directed arcs between blocks. The first argument gives the blocks that are to be connected; the second is a list of from-to pairs that specify the

arcs; the third provides a collective variable name for the linking activities. In Figure 4 the second statement specifies an arc from Vendors to Warehouses. At this stage the definition of level 1 is complete but no match can be made in the data schema for 'Vendors' and 'Warehouses'. The system therefore continues its interaction with the user.

The next two CREATE-BLOCKS statements create a second level of detail. Property inheritance from the prior LINK-BLOCKS operation ensures that each specified vendor is linked to each specified warehouse. Other block-language statements can be used to add or delete arcs at this level if necessary.

A MINIMIZE (or MAXIMIZE) statement specifies the objective function coefficients by providing a list of names of relations where the data can be found together with the algebraic sign to be used e.g. MINIMIZE(Unitprofits - Transcosts - Fixedcosts).

The five statements in Figure 4 could be specified in any order. Together with the data schema they allow the rule-based system to complete the problem formulation. Some typical rules are shown in Figure 5.

First, we explain the interaction with the data schema. 'Vendor' and 'Warehouse' are recognized as field names thus satisfying one of the problem completeness criteria. Statements c. and d. actually specify a database selection by specifying which data records of the underlying real database are to be retrieved later by the LP Generator system. Arbitrarilly complex selections can be envisaged; specifying the field name by itself

Rule No.	General Description
1.	A block is of type 'exogenous supply' if it has no activities defined for it and has only outgoing arcs.
2.	If a block is of type 'exogenous supply': a. Add a row to the tableau for each output commodity; the row name consists of the block name concatenated with the output name. b. Add a 'supply limit' constraint set of the form: $\sum_j ?_{ij} \leq ?_i$ where the ? symbols are yet to be identified. The index set for i is known from 2a.; the index set for j is determined when the variable is identified.
3.	For a LINK-BLOCKS statement: a. Add a column to the tableau for each flow; the column name is a concatenation of the from- and to-block names and the name of the commodity. b. Use the variable name and index sets in the argument to identify the transportation activities associated with the columns of step a. c. If no gains or losses are associated with the activities their tableau coefficients in the associated demand constraints are equal to ± 1.

FIGURE 5
Illustration of Rules to Generate Algebraic Problem Statement

is equivalent to specifying all the data records in the relation. From statement b. and rule 3b., the system infers that the 'X' variables are doubly subscripted by the elements of Vendors and Warehouses. Using c. and d. it can be seen that these form the key of Trans-costs. The system can therefore associate the values in the 'C' field with variable X in the objective function of (3).

The system next uses Rule 1 in Figure 5 to infer that v1 through v3 are source blocks. Rule 2 is selected next to generate the general form of the first inequality constraint in (3). It can then infer from the data schema that the required values of the RHS coefficients are given by the 'S' column in the SUPPLY relation and that X is the relevant variable name. Similar steps are performed with similar rules for the warehouse blocks.

The final steps performed by the LP Formulator are as follows:

(a) Access the metadata concerning the units in which the data are stored and perform a dimensional analysis to determine if the data are conformable and to calculate conversion factors if necessary.

(b) Access and analyse the actual data values to see if there are scaling problems that will impair the accuracy of the results and to calculate corrective factors if necessary (not planned in current implementation).

(c) Output the algebraic problem statement (equivalent to (3)) for execution by the tableau generator together with the relevant data retrieval commands.

The preceding description shows how the various components of the knowledge base can be used cooperatively to formulate the LP. Obviously this was a very simple problem. Before leaving this section we will therefore briefly describe some other block language statements:

DEF-INPUTS (DEF-OUTPUTS)

Used to define the inputs (outputs) to a block in a multi-commodity network.

LINK-INPUTS-OUTPUTS

In a multi-commodity network this command links like inputs to like outputs. e.g. natural gas as an output of a block will be automatically linked to all blocks for which it is an input.

REPLICATE-BLOCK

Blocks (or groups of blocks) can be replicated in either space or time. For example, if the inputs, outputs and internal structure of all oil-producing regions are the same, it is easier just to define the structure of one region and then use this command to generate similar structures for all the others. Essentially this just adds a region index to all production and transportation activities associated with the block. Note that the tableau data values may still differ from region to region.

DEF-ACTIVITIES

Used to define the list of activities of various types associated with a block. This is the most important step in defining blocks representing form transformations as in blending and product-mix problems. Each activity is associated with an index set, a variable name and its inputs and outputs.

Note that the two "LINK" commands create transportation activities while the DEF-ACTIVITIES command creates more general kinds of activities.

5. CONCLUSION

This paper has given a brief overview of the LP Formulator system. However many details have been ommitted. A future paper will describe the Block Language in more detail - particularly with regard to the replication of problems in space and time and the definition of various kinds of form transformations. Other papers will describe the PROLOG implementation and the role of

the database in the knowledge representation.

The prototype currently being constructed represents a first attempt only. Many questions will remain to be investigated even after this has been built and tested. The first set of questions concerns the design of the user interface (beyond the Block Language which is primarily for internal system purposes). What is the best mixture of graphic-, menu- and command- driven styles of input? Will different users have very different requirements? Can the system support non-expert as well as expert users?

The second set of questions involves the role of the database system. Can this be generalized so that it can determine data needs and automatically access data in the corporate database? Can complex data transformations be included as part of the retrieval process?

A third area of investigation will involve extending the system to other problem types. This will include a better understanding of the time dimension to allow us to handle scheduling problems. We will also need to investigate how to formulate integer and perhaps non-linear programming problems.

Finally, much work remains to be done in order to build intelligence into other aspects of the problem solving process: model testing and validation, automatic generation of alternative scenarios and aids for analyzing the results of models.

References

1. Astrahan, M. M., and Chamberlin, D. D., 'Implementation of a Structured English Query Language', Communications of the ACM, Vol. 18, No. 10, October, 1985, pp. 580-588.

2. Binbasioglu, M., and M. Jarke, 'Domain-Specific Dss Tools for Knowledge-Based Model Building', Working Paper #97, Center for Research in Information Systems, New York University, 1985.

3. Brown, R. W., W. D. Northup and J. F. Shapiro, 'Logs: A Modeling and Optimization System for Business Planning', Computer Methods to Assist Decision Making, North Holland, New York (to appear).

4. Clocksin, W. F. and C. S. Mellish, Programming in Logic, Springer-Verlag, New York, 1981.

5. Dantzig, George B., Linear Programming and Extensions, Princeton University Press, Princeton, N. J., 1963.

6. Gershon, Mark E., 'L.P.-BLEND: Documentation and User Manual', Memo, Temple University, 1985.

7. Glover, Fred, John Hultz and Darwin Klingman, 'Improved Computer-Based Planning Tools, Part 1', Interfaces, Vol. 8, No. 4, 1978, pp. 16-25.

8. Greenberg, Harvey J., 'A Tutorial on Computer-Assisted Analysis', in Advanced Techniques in the Practice of Operations Research, Greenberg, Harvey J., Murphy, Frederic H. and Shaw, Susan H. (eds), New York: North-Holland, 1982, pp. 212-250.

9. Greenberg, Harvey J., 'A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models', ACM Transactions on Mathematical Software, Vol 9, No. 1, March 1983, pp. 18-56.

10. Haverly Systems Inc., OMNI Linear Programming System: User Manual and Operating Manual, Denville, N.J., 1977.

11. Hogan, William W., James L. Sweeney and Michael H. Wagner, 'Energy Policy Modeling in the National Energy Outlook', in Energy Policy: TIMS Studies in the Management Sciences, Vol. 10, J. S. Aronsfoky, A. G. Rao and M. F. Shakun (eds), 1978.

12. IBM Mathematical Programming Language Extended/370 (MPSX/370), Program Reference Manual, SH19-1095, IBM Corporation, Paris, France, 1975.

13. Murphy, F. H., R. Sanders, S. Shaw and R. Thrasher, 'Modeling Natural Gas Regulatory Proposals: Using the Project Independence Evaluation System', Operations Research, vol 29, no 6, Nov-Dec 1981, pp876-902.

14. Slate, L. and K. Spielberg, 'The Extended Control Language of MPSX/370 and Possible Applications', IBM Systems Journal, Vol. 17, No. 1, 1978, pp.64-81.

15. Stohr, Edward A., 'A Mathematical Programming Generator System in APL', Working Paper 96, Center for Research in Information Systems, Graduate School of Business Administration, New York University, 1985.