THE SCIENCE AND ART OF FORMULATING LINEAR PROGRAMS

by

Pai-chun Ma*
Frederic H. Murphy+
and
Edward A. Stohr*

June 1987

* Graduate School of Business Administration
New York University
90 Trinity Place
New York, N.Y. 10006

+ School of Business
Temple University
Philadelphia, PA

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

Working Paper Series

CRIS #132
GBA #86-82

This paper will be published under the title "Computer-Assisted Formulation of Linear Programs" in the IMA Journal of Mathematics in Management, September, 1987.

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

Table of Contents

List of Figures

Abstract

This paper describes the philosophy underlying the development of an intelligent system to assist in the formulation of large linear programs. The LPFORM system allows users to state their problem using a graphical rather than an algebraic representation. A major objective of the system is to automate the bookkeeping involved in the development of large systems. It has expertise related to the structure of many of the common forms of linear programs (e.g. transportation, product-mix and blending problems) and of how these prototypes may be combined into more complex systems. Our approach involves characterizing the common forms of LP problems according to whether they are transformations in place, time or form. We show how LPFORM uses knowledge about the structure and meaning of linear programs to construct a correct tableau. Using the symbolic capabilities of artificial intelligence languages, we can manipulate and analyze some properties of the LP prior to actually generating a matrix.

1. Introduction

Every book on linear programming has at least one chapter on formulating linear programs (LPs), that illustrates various applications. Some books such as Gass, [9] or Schrage, [19], spend a substantial amount of time going over different formulations in detail. Typically, a few examples are presented followed by a collection of problems that gives the student practice on a range of formulations. Pedagogically, this is the case study method; the rationale is that the examples will approximate the actual problems that the student will eventually have to formulate. The more problems a student formulates, the more likely this will hold true. Yet, a more formal approach could both accelerate the comprehension of different models and be translated into an intelligent system to aid in the process of formulating LPs.

There are several reasons for building an intelligent system to assist in model building. First, the practical use of LPs has been limited to professionals who are knowledgeable in matrix generation software. An intelligent system could make the technique more accessible to managers and also help experts by eliminating much of the error-prone specification and clerical work that usually plagues the construction and maintenance of large LP models. It could also provide automated checking and model documentation and record the impact of modeling decisions on the final result, allowing a much more sophisticated form of sensitivity analysis. Finally, it should be possible to teach linear programming using models of a reasonable size without having to cover matrix generation languages.

An early structured approach to the manual formulation of LPs is given in Dantzig, [6]. Matrix generation systems such as OMNI [11] take an activity or column-oriented view of the problem and provide data manipulation and computational aids for generating the initial tableau input for IBM's LP solver, MPSX [12]. Modeling language systems, such as GAMS [16] and XML [8], take a constraint or row-oriented view accepting an algebraic specification of the LP problem as a set of linear inequalities and translating this into the format required by the solver. Fourer, [8] compares the matrix generator and modeling language approaches. Lucas et al, [13], have developed a system called CAMPS, which employs a new modeling language. Creegan, [4] and [5], has developed a system called PAM that draws on some standard structures, simplifying the process of using Dataform in matrix generation. Finally, Brown and Shapiro, [2] have developed LOGS, a system for formulating linear and integer programs. All of these approaches are language-oriented and require some expertise both in the language system itself and in the art of formulating LPs.

Our research is part of a project to construct a system, 'LPFORM', for formulating linear programs. The interface to LPFORM will employ graphic 'icons' to change the representation of the problem to one that we feel will be more natural to non-expert users and easier for expert users. A prototype version of LPFORM has been being developed in the PROLOG language (Clocksin and Mellish, [3]). This translates statements in a graphical specification language (Ma, et al, [15]) to an algebraic form. The latter is passed to an LP Generator (Stohr, [20]) which, in turn, generates data for input to IBM's MPSX mathematical programming system, [12]. See Murphy and Stohr, [17] for an overview of the LPFORM system and Ma, et al, [15] for a description of the user interface.

In most current systems the symbolic statement, if it exists, is constructed by hand and is usually out of date. As mentioned above, LPFORM takes miscellaneous inputs from the modeler and produces and maintains the algebraic statement of the LP. In contrast, GAMS, (Meeraus, [16]) uses the algebraic statement as a starting point and produces the input file for the LP solver. The advantages of the symbolic statement are that it is useful in the initial conceptual phase of modeling, it is the most compact and universally understood form of documentation and so is best for learning an existing model, it is independent of the data, and finally, one can do a significant portion of the debugging and model analysis on this statement without having to wait for the tableau to be generated. Analysis of the algebraic statement can be used in conjunction with systems (such as Greenberg's ANALYZE, [10]) that investigate the model structure after the LP tableau has been formed. The disadvantage of the symbolic statement is that it can be understood only by LP experts. In LPFORM, naive users and experts will be able to use the graphic form of input; experts will be able to check their models using the generated symbolic statement.

In this paper we show how LPFORM uses knowledge about the structure and meaning of linear programs to simplify the process of building a model and to check the correctness of the formulation. Section 2 gives our view of the formulation process and outlines the techniques used by LPFORM to help users build models. Section 3 describes how some simple syntactic properties can be used to perform a significant portion of the model building. Section 4 shows how additional, semantic information can be used to augment the reasoning in Section 3 and to ensure the correctness of the formulation.

2. A Conceptual View of the Formulation Process

To build an expert system one must formalize the procedures used by experts in the field. Although we, as operations researchers, view the process of formulating linear programs as an art, the fact that so many models have similar characteristics provides opportunities for making the process of model building more systematic. For a human expert the formulation process involves problem recognition, problem classification, symbolic definition, data collection and model statement generation according to the rules of some matrix generator. LPFORM assists in all of these phases except for the first.

When an expert begins to work on a problem, various facts and requirements come to mind in a fairly random fashion. These might be written on a scratchpad and reworked several times before a complete formulation is developed. LPFORM attempts to provide an electronic scratchpad with a graphical interface and memory aids as described more fully in Ma et al, [15]. The most visible feature of the interface is that it allows users to define networks of arcs connecting 'blocks'. Blocks contain subsets of the activities of the problem that are separated from other activities in either space or time. The networks can be decomposed in a 'top-down' fashion into successive levels of detail with lower levels inheriting the properties of higher levels. At the lowest level in the hierarchy, blocks represent exogenous demands and supplies, factories, or machines and the arcs connecting blocks represent flows of commodities or of time. Inventories, resources and commodities are specified on the screen by placing icons on the relevant blocks or arcs.

Figure 1 shows the graph that would be constructed to define a problem in which widgets, *j*, are produced at factories, *f*, and shipped to markets, *m*. The modeler places 'block icons' for the factories and markets on the computer screen, links the two blocks to define the transportation activity, and then places an 'activity icon' in the Factories block. As this process takes place, LPFORM prompts for the names of the blocks, decision variables (X and T) and the inputs and outputs of the production activity. The icon for the latter, is based on in the activity analysis approach of Dantzig, [6], where activities are black boxes that transform inputs into outputs.

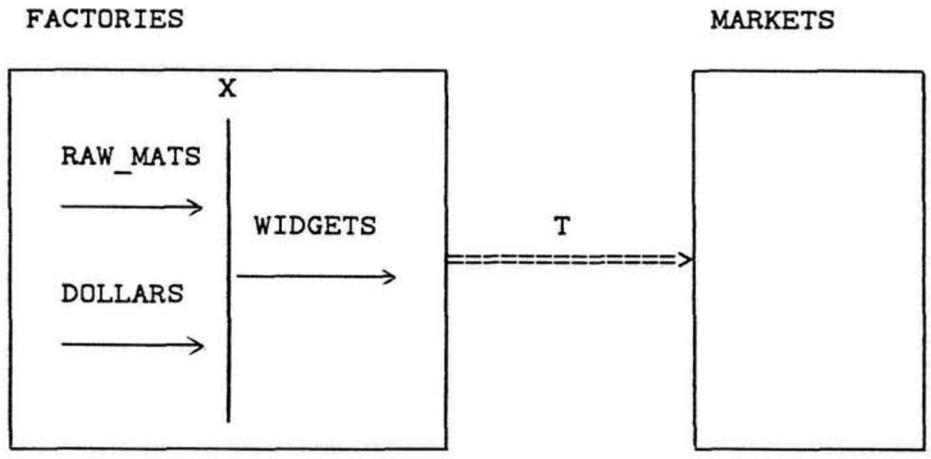


Figure 1: Graphical Definition of a Problem

Given the problem specification shown in Figure 1, LPFORM will produce an algebraic statement equivalent to:

$$\text{Min:} \quad \sum_{j \in Widgets} \sum_{f \in Factories} c_{j,f} X_{j,f} + \sum_{j \in Widgets} \sum_{f \in Factories} \sum_{m \in Markets} w_{j,f,m} T_{j,f,m}$$

$$\text{St:} \quad \sum_{j \in Widgets} a_{i,j,f} X_{j,f} \leq b_{i,f}, \qquad \forall\, i \in RawMats,\, f \in Factories$$

$$-X_{j,f} + \sum_{m \in Markets} T_{j,f,m} = 0, \qquad \forall\, j \in Widgets,\, f \in Factories$$

$$\sum_{f \in Factories} T_{j,f,m} \geq d_{j,m}, \qquad \forall\, j \in Widgets,\, m \in Markets$$

The symbolic statement then goes through a data linking phase in which the elements of the index sets as well as the numeric values of data coefficients are specified.

A second major feature has been to automate a 'bottom-up' approach in which models are specified in terms of previously defined sub-problems. These can be standard formulations such as the product-mix or transportation models; alternatively, users can construct a model using the system and then store it for later use within larger problems. Thus, the above problem could have been specified by retrieving product-mix and transportation templates and 'mapping' them on to the larger problem by renaming index sets and coefficients.

From the perspective of LPFORM, formulation means using various problem-related 'clues' given by the user to generate an algebraic problem statement. The clues can take various forms, and fall into two main classes depending on whether they relate to the structure or to the data of the problem.

The real world relationships, constraints and objectives that define the structure of the LP can be specified in many different ways including:

1. Hierarchically defining network structures in successive levels of detail then linking the resulting 'blocks' into a larger system. This is a 'top-down' approach.

2. Associating real world objects with the structures defined in 1. Thus commodities can be assigned to arcs and resources and inventories

3. Specifying that a portion of the representation is to be 'replicated'. e.g. a factory submodel might be repeated for different regions.

4. Specifying that a previously defined problem (either 'standard' or user-defined) is a sub-problem. This is a 'bottom-up' approach.

5. Specifying a set of activities by describing their inputs and outputs (see Figure 1). This defines a set of columns.

6. Stating a constraint set directly in algebraic form using a summation notation. This defines a set of rows.

Methods 1, 2 and 3 depend for their effectiveness on the computer graphics interface. Eventually, we hope that method 6 will not be needed. It is included to allow users to refine the algebraic representation constructed by LPFORM.

Information relating to the data of the problem - the index sets and activity, cost and right-hand-side coefficients - can also be specified in a number of ways:

1. Interactively input from the terminal during problem definition. This is useful for very small problems. The index sets for coefficients are often implicitly defined during the session in any case.

2. Specified as existing in external tables (the table names and definitions are provided by the user or can be read in from an external file).

3. Specified as queries on database relations; the result of the query is transformed into a table.

4. Defined through computation from other known values. The tables and database relations carry information concerning the unique identifiers for numerical values (which translates to knowledge concerning the dimensionality of data coefficient indices) and, optionally, knowledge concerning the units in which the data coefficients are expressed.

Like a human expert, LPFORM is able to use these different forms of input and infer the algebraic structure of the resulting problem. A design objective was to provide this diversity while allowing the information to be specified in any desired order. Unlike a human expert, LPFORM is unable to recognize what information is pertinent in the first place. However, given a starting basis of facts, it can recognize gaps in the user specification and will request additional information to complete the problem statement. There are many different ways of arriving at a correct statement of a given problem and the inferences can be made starting either from data-related or structure-related statements as defined above. For example, if we know the direction of the optimization and have three compatible tables (with dimensions m, n and $m \times n$ respectively) that we know contain all the data for the problem, then we can infer that we have a standard form of the LP and build the correct algebraic formulation. Alternatively, the same standard LP problem can be pieced together from first principles - by specifying the decision variables, constraints and so on.

The reasoning power of the system is an amalgam of different kinds of knowledge:

1. Syntactic form of an LP - e.g. it must contain both an objective function and constraints and indices and summations have to follow precise patterns.
2. Data-related - e.g. coefficients need to be linked to values, the units in which the data is stated must be consistent and the indices of the coefficients must correspond to those of the constraints and variables.
3. Network structures - e.g. inputs and outputs should balance; standard problem templates can be invoked.
4. Standard constraint types - e.g. material balances, upper and lower bounds, generalized upper bounds and blending.
5. Standard problem types - e.g. the common forms of network, product-mix and blending problems.
6. Economic - e.g. the distinctions between materials and capital stocks or the treatment of different dimensions of space or time.

The above types of knowledge fall into two categories: syntactic (type 1 and semantic (types 2 through 6).

The design strategy for the current version of LPFORM was to concentrate on the syntactic properties of the algebraic and tableau representations of LP problems. Later we intend to extend the system to allow users to add application-specific semantic information. The advantages of the syntactic approach are that it applies to all LP problems, it is easier to understand and to program, it facilitates consistency checking and finally, it provides a good base for later insertion of semantic knowledge. On the other hand, many ambiguities cannot be resolved without semantic knowledge. A semantic approach is being employed in (Binbasioglu, [1]). This may lead to 'deeper' intelligence but in a narrower domain of application.

Some semantic knowledge has already been included in LPFORM, but it is general rather than specific. The system knows the distinctions between inventories and resources and between uses and sources of materials. On the other hand, it does not know that oil is a commodity and that electric lathes consume electricity. Such knowledge either has to be implicit in the structure of the data or provided by the user. Of course, it is not always clear what is syntactic and what is semantic when one examines LPFORM closely. For our purpose, we define syntactic as whatever can be done by the simple pattern-matching algorithm described in the next section. When additional information, such as index role descriptors and constraint and activity types, has to be built into the internal structure of the system we are employing semantic knowledge.

3. Using Syntax and Structure

Figure 2 shows the structure of an LP problem within LPFORM.

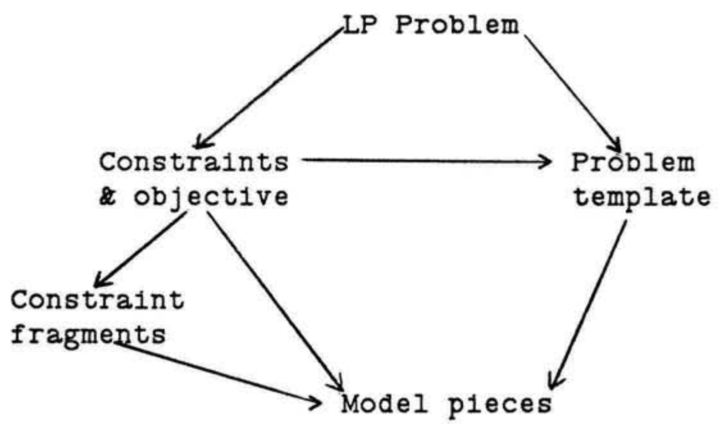


Figure 2: LP Problem Structure

The arrows indicate one-to-many relationships. A model 'piece' is an algebraic term with summation and index sets or a right-hand-side coefficient including the inequality relation. A constraint fragment consists of one or more model pieces. For example, the following constraint fragment has two pieces:

$$\sum_{j \in J} a_{i,j} X_j \leq b_i, \quad \forall i \in I$$

There are standard templates for the product-mix and blending problems and so on. The user can also save a problem definition as a template, thereby building a personal library.

The user interaction and resulting inference process indicate which model pieces, constraint fragments or problem templates are to be included in the problem. The components are then combined as if they were pieces in a jig-saw puzzle (see Murphy et al [18]). The remaining sections of the paper outline the process of building the final problem statement from the component parts shown in Figure 2.

LPFORM analyses inputs from the user and selects the model pieces, constraint fragments and model templates that will be the building blocks for the problem (see Ma, [14], Ma et al, [15] and Murphy, et al, [18]), for a more detailed explanation). Sometimes the pieces are redundant; sometimes there are too few to build a complete tableau. In the former case, LPFORM ignores the redundancies; in the latter case it interacts with the user to obtain more information. This section of the paper briefly explains the 'Puzzler' mechanism that assembles the algebraic form of the LP tableau and checks its consistency. There are three different but related forms of reasoning in the Puzzler: index analysis, simple assignment and dimensional analysis. Discussion of dimensional analysis is delayed until the next section since this employs semantic information.

Index Analysis

We define an index set for a data coefficient or variable as the collection of indices (or subscripts) that identify it in the algebraic statement. Each individual index identifies a 'dimension' of the data or variable and takes on values within some well-defined domain. The set of indices on a coefficient or a variable can be partitioned into two subsets: those summed within a single constraint and those that distinguish the constraints from each other. The indices on the right-hand-side (RHS) include the ones not summed on the left-hand-side (LHS). Note that the indices on the variables and coefficients that are not summed do not necessarily indicate all of the indices on the RHS coefficients since, for example, the limit of a summation on the left can be an index appearing on the right. However, if all summations are over all elements in the domains of the indices summed, the sets are equal. To illustrate these relationships, consider the following example:

$$\sum_{k=1}^{t} \sum_{j \in J} a_{i,j,k} X_{j,k} \leq b_{i,t}, \quad \forall i \in I, t = 1,2, \ldots ,T$$

For a given constraint and variable with coefficients in that constraint, let:

$V =$ {indices on the variable} = {j,k}
$C =$ {indices on the coefficient of the variable} = {i,j,k}
$S =$ {indices that are summed over their whole domain} = {j}
$P =$ {indices that are summed over a subset of their domain} = {k}
$Q =$ {indices of subsets over which partial sums are taken} = {t}
$R =$ {indices that distinguish the individual constraints} = {i,t}

where the sets on the right correspond to the example. Then in a properly specified LP:

$$(V \cup R) \supseteq C, \quad V \supseteq (S \cup P), \quad \textit{and} \quad R = ((V \cup C) - S - P) \cup Q.$$

These relationships can be used to help infer the complete problem from its component parts, to automatically adjust the system when a part of the model is replicated, and to check the correctness of the final formulation. The examples given later in this section illustrate the principles involved.

Simple Assignment

Each model piece is associated with row and column labels and a set of physical units (e.g. dollars per unit of raw material). The row label for a LHS piece is the list of names of indices in the index set for the RHS. The column label for a LHS piece is the variable name itself concatenated with the names of all the indices associated with the variable. If the row and column labels for a piece are known it can be assigned to its correct location in the tableau - solving part of the puzzle. Note that the column labels in the tableau are unique. The row labels are also unique except when there are upper and lower bounds; in this case we need semantic information to distinguish between them and to allow the assignment process to proceed correctly.

Example 1: Building a Model from Pieces

Suppose there are activities $X_{j,p}$ that produce products p in set P using technologies j in set J. The products p are then shipped to warehouses k in set K and the shipping is represented by activities $T_{p,k}$. For product p the amount produced is $\sum_{j\in J} X_{j,p}$, and the amount shipped is $\sum_{k\in K} T_{p,k}$. These two model pieces have row label, p, and would be placed in the same material balance constraint by the reasoning process:

$$\sum_{j\in J} X_{j,p} - \sum_{k\in K} T_{p,k} = 0, \quad \forall p \in P.$$

The minus sign comes from semantic information on whether the variable represents an input or output. This is stored with each piece as it is generated. The RHS value of 0 is provided by default.

Example 2: Using Templates and Constraint Fragments

Suppose the user has defined a single block and specified that it contains a process selection model, which is one of the predefined templates. The following template could be recalled from the library, matched with the current data and executed:

$$\begin{array}{ll} \text{Minimize:} & \sum_{j\in J}\sum_{k\in K} c_{j,k}X_{j,k} \\ \text{Subject to:} & \sum_{j\in J}\sum_{k\in K} a_{i,j,k}X_{j,k} \le b_i, \quad \forall i \in I \\ & \sum_{k\in K} X_{j,k} \ge d_j, \quad\quad\quad\;\; \forall j \in J \end{array}$$

Suppose the user now places an inventory icon in the block and in response to the prompts indicates that final goods (rather than input materials) are the inventoried items. LPFORM adds a time index to every existing data coefficient and variable, selects the inventory constraint fragment template:

$$+ I_{M,t-1} - I_{M,t}$$

where I and M are dummy variables, assigns it a row label of finished goods, and adds it to the existing model using simple assignment and index analysis to get the following:

$$\begin{aligned}
\text{Minimize:} \quad & \sum_{j \in J} \sum_{k \in K} \sum_{t \in T} c_{j,k,t} X_{j,k,t} + \sum_{j \in J} \sum_{t \in T} h_{j,t} I_{j,t} \\
\text{Subject to:} \quad & \sum_{j \in J} \sum_{k \in K} a_{i,j,k,t} X_{j,k,t} \le b_{i,t}, \quad \forall i \in I, t \in T \\
& \sum_{k \in K} X_{j,k,t} + I_{j,t-1} - I_{j,t} = d_{j,t}, \quad \forall j \in J, t \in T
\end{aligned}$$

The need for a holding cost term in the objective is part of the information stored with the inventory constraint fragment but could be inferred in any case.

Example 3: Replication

LPFORM allows a large LP to be constructed by first building and testing small models and then replicating them over sets such as place or time. This process is automated using index analysis. Suppose the user starts building a transportation model by specifying one supply and one demand node as shown in Figure 3(a).

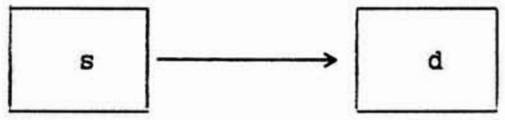
This results in the following LP with one variable:

$$\begin{aligned}
\text{minimize:} \quad & cX \\
\text{subject to:} \quad & X \le s, \quad X \ge d, \quad X \ge 0
\end{aligned}$$

Suppose the user replicates the demand node over some set, J, giving the graph in Figure 3(b). LPFORM puts an index on d and then on X so that the single demand constraint becomes the collection of constraints:

$$X_j \ge d_j, \quad \forall j \in J$$

(a) A Transportation Model with One Source and One Destination



(b) Transportation Model with One Source and Multiple Destinations

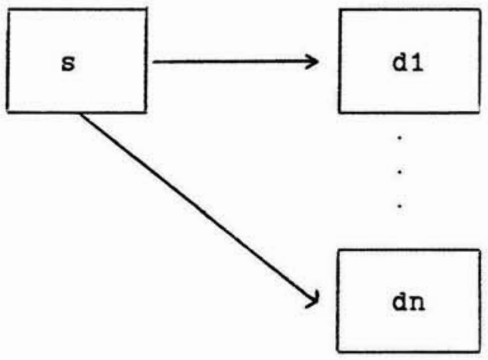


Figure 3: Replication of Demand Points

which satisfies the index rule. However, the supply constraint now violates the index rule so that a summation has to be inserted:

$$\sum_{j \in J} X_j \leq s.$$

Replicating on supply and summing on demand as required by the index rule results in the transportation model.

4. Establishing Meaning in the Knowledge Base

Many aspects of LP model formulation allow us to add meaning to the pattern matching algorithm described in the last section. Examples are dimensional analysis, network analysis, the concepts of supply and demand and of materials and resources, the collections of template models and knowledge concerning the type of transformation. LPFORM uses this information, along with the Puzzler, to construct large LP models from smaller template models or constraint fragments.

Dimensional Analysis

If the units in which the data are expressed are provided by the user, or included in the header information of the data tables, the Puzzler will check to see that they are compatible through the constraint sets intersected by an activity and within each constraint set. This requires a variant of the dimensional analysis used in physics and engineering [7], since summation over all the elements in a domain reduces the index set (dimensionality) of a term in an LP (see the discussion in Example 1) while this does not take place with models in conventional dimensional analysis.

Network Analysis

The first step performed by the LPFORM parser is to analyze the layered hierarchy of network structures input by the user. The leaf nodes are determined and the network structure connecting the leaf nodes is analyzed to determine if it is valid (no isolated nodes or dangling arcs). Thus, using this approach, the network structure is partially validated by the system at the beginning of the formulation process rather than being discovered later as in Greenberg, [10]. If none of the blocks (nodes) have activities, then the problem is a pure network problem. Further analysis determines which of a number of prestored network problem templates should be invoked (transportation, transhipment or general network). Finally, semantic information is gathered on the roles of the leaf nodes in the hierarchy (supply, transhipment or demand points).

Supply and Demand

In complex production systems, the demand for inputs in one production stage requires a supply of outputs from another. The stage that requires the inputs is limited, in part, by the outputs of the previous stage. Noting that supply must be greater than or equal to demand, one can unify the different types of constraints into the following single constraint:

$$-\ sv + uv \leq sc - uc$$

where

$sv =$ the sum of the supply variables (scaled by appropriate constants)

$uv =$ the sum of the use variables (also scaled by the appropriate constants)

$sc =$ the constant measuring fixed supply

$uc =$ the constant measuring fixed demand

A resource limit constraint becomes $uv \leq sc$ and the standard material balance becomes $-\ sv + uv \leq 0$. Note that if one assumes the coefficients in the sums are positive, not all combinations are meaningful and feasible. For example, $-\ sv \leq sc$ is not meaningful since it will never bind and $uv \leq -\ uc$ is never feasible with a positive uc. This structure allows one to check partially the reasonableness of the solution and establishes the sign conventions in linking constraint fragments.

Materials and Resources

In the theory of production, capital, labor and materials are used to produce goods. When a time dimension is added, each of these inputs should be treated differently: we consume materials but only use the services of capital and labor. One accounts for the quantity of a material in the same constraint where it is supplied and used. On the other hand, with capital and labor resources one accounts for the amount of a resource separately from its uses. For example, a typical equation with inventories takes the form:

$$I_{t-1} - I_t + P_t = D_t$$

where I_t is the inventory at the end of period t, P_t is the production in period t and D_t is the demand. Whereas with labor the typical constraints are:

$$w_{t-1} - w_t + h_t - f_t = 0$$

$$a_t P_t - w_t \leq 0$$

where w_t is the number of workers in period t, h_t is the number hired at the beginning of the period, f_t is the number fired, and P_t is production in period t. LPFORM selects the appropriate constraint fragments when a material or a resource is being incorporated in an LP.

Prestored Templates

The prestored constraint fragment and standard problem templates play an important role in LPFORM. The constraint fragments (some as small as individual pieces) are the basic building blocks. The current inventory of constraint fragments includes the material balance, supply, demand, inventory and resource constraints that have been used in earlier examples. Others will be added as the need arises. However, it is obviously impossible to anticipate all the details of every constraint that could occur in practice. The strategy is to provide a few general constraint types. The system must then be able to adapt these to particular problems by adding indices (as in the replication example) or by dropping indices when the problem is less general in scope than the prestored fragment. LPFORM currently has a limited ability both to add and drop indices. The next section outlines a more general scheme.

The inventory of prestored problem templates allows one to very quickly build much larger models. Problem templates are stored as collections of pieces. They are self-documenting in the sense that each component has both a name and an explanation. The ability to add problem templates allows the system to be customized to the needs of individual users.

Transformations

Linear programs represent physical processes that transform inputs into outputs. There are three basic transformations: transformations in place, in time and in form. The transportation problem is a transformation in place, the manpower planning problem is a transformation in time and the product mix model is a transformation in form, e.g. raw materials are physically transformed into a final product. Transformations involve the notion of a flow. There can be a flow of inputs into a product, a flow of items from one location to another, or a flow of inventories through time.

In production systems, an object is described by three dimensions: what it is, where it is and when it exists in that place. This three dimensional state can be altered by transformations in form, place and time. A pure transformation in form converts a collection of 'whats' into another collection of 'whats' while leaving place and time identical. Analogous statements can be made about place and time transformations.

Figure 4, shows how transformation activities are associated with indices having different roles.

A pure transformation in form has indices indicating, respectively, 'from what' (inputs), 'to what' (outputs), and 'where', 'when' and 'how' the transformation is to be performed. Transformations in place are distinguished by having indices on what, when and how, plus from where to where. Transformations in time have indices on what, where and how and from when to when. The indices need not be one dimensional in the usual sense. For example, an item which is the i_{th} component of the j_{th} product may be indicated by the pair *(i,j)* which becomes the index for form. Also, from-to pairs are often collapsed into one index: arcs can be identified by a numbering scheme rather than by from-to location pairs and product-mix activities are identified by the output product alone. Finally, as long as the names on the index sets are carried along, the order of the indices is unimportant.

```
                ACTION
--|-----------|--------------|--
PLACE         FORM           TIME

From where    Where          Where
To where                     What
              From what
              To what        From when
What                         To when
When          When
How           How            How
```

Figure 4: Index Roles for Simple Transformations

Compound activities such as simultaneous transformations of place and time are possible (e.g. if it takes one period to ship the product so that what is produced at time t is available at time $t+1$). However, if one has pure transformations only, the double indexing denoting the 'from-to' of what, where or when allows one to recognize the type of transformation associated with an activity. Conversely, if the type of an activity is known, the above scheme can be used to help match symbolic names with data and to generate prompts to the user if the system can not resolve the issue by itself.

5. Conclusion

This paper has provided some background on our approach to building a system to aid in the formulation of linear programs. Although many of the observations we have made seem fairly obvious they must be stated so that more detailed, program-level rules and procedures can be developed. We do not expect to be able to build a system to automate every aspect of every model formulation. Nor will the system ever be able to formulate all linear programs without the aid of an intelligent user. However, the knowledge base can be expanded, allowing the user to add templates specific to his or her situation. Even with the current prototype we are able to provide a high level of automation for some common situations and heuristic, book-keeping forms of support in less routine situations.

Acknowlegement

This paper has benefited greatly from the suggestions of one of the referees.

References

1. Binbasioglu, M. *Knowledge Based Modelling Support for Linear Programming.* Ph.D. Th., New York University, 1986.

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

3. Clocksin, W.F., and Mellish, C.S.. *Programming in Prolog.* Springer-Verlag, New York, N.Y., 1981.

4. Creegan, J. B., Jr. Dataform: A Model Management System. , Ketron, Inc., Arlington, Virginia, November, 1985.

5. Creegan, J. B., Jr. *PAM: A Practitioner's Approach to Modeling, Volume I - Primer.* Ketron, Inc., Arlington, Virginia, 1985.

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

7. De Jong, F. J.,and Wilhelm Quade. *Dimensional Analysis for Economists.* North-Holland, Amsterdam, 1967.

8. Fourer, R. "Modeling Languages versus Matrix Generators for Linear Programming". *ACM Transactions on Mathematical Software 8,* 2 (June 1983).

9. Gass, Saul I.. *Linear Programming: Methods and Applications.* McGraw-Hill, New York, 1969.

10. Greenberg, Harvey J. "A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models". *ACM Transactions on Mathematical Software 9,* 1 (March 1983), 18-56.

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

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

13. Lucas, C., G. Mitra and K. Darby-Dowman. Modeling of Mathematical Programs: An Analysis of Strategy and an Outline Description of a Computer Assisted System. TR/09/83, Brunel University, Uxbridge, England, 1983.

14. Ma, P. *An Intelligent Approach to Formulating Linear Programs.* Ph.D. Th., New York University, 1986. Ph.D. Dissertation Proposal.

15. Ma, P., F. H. Murphy and E. A. Stohr. Design of a Graphics Interface for Linear Programming. Working Paper No. 136, Center for Research in Information Systems, Graduate School of Business Administration, New York University, New York, 1986.

16. Meeraus, A. *General Algebraic Modeling System (GAMS): User's Guide, Version 1.0.* Development Research Center, World Bank, 1984.

17. Murphy, F. H. and E. A. Stohr. "An Intelligent System for Formulating Linear Programs". *Decision Support Systems 2*, 1 (Jan-Feb 1986).

18. Murphy, F. H., E. A. Stohr and P. Ma. Composition Rules for Building Linear Programming Models from Component Models. Working Paper , New York University, New York, 1987.

19. Schrage, Linus. *Linear, Integer, and Quadratic Programming with LINDO.* The Scientific Press, Palo Alto, 1984.

20. Stohr, E. A. A Mathematical Programming Generator System in APL. Working Paper 96, New York University, New York, 1985.