AUTOMATED SUPPORT FOR FORMULATING LINEAR PROGRAMS[1]

by

Edward A. Stohr

Information Systems Area
Graduate School of Business Administration
90 Trinity Place
New York University
New York, NY 10006

February 1988

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

Working Paper Series

CRIS #172
GBA #88-4

---

[1]To be published in: *Proc. NATO ASI: Mathematical Models for Decision Support*, Val D'Isere, France, August 1987.

Table of Contents

List of Figures

ABSTRACT

Most research in mathematical programming has been concerned with efficient computational algorithms. However, there is increasing interest in developing automated techniques for supporting the modeling process. This paper describes a new kind of interface for formulating linear programming models and explains the inference process used to translate problem specifications into algebraic formulations. The main idea underlying the design of the interface is to change the specification language to a graphical rather than a mathematical notation. The inference process involves the generation of algebraic terms and their subsequent combination into constraint equations. This relies on the syntactic relationships among indices and a knowledge of the physical entities that they represent. An advantage of the approach is that it facilitates the reuse of model components from previous models. The ideas discussed in this paper have been incorporated in a prototype system.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-88-04

AUTOMATED SUPPORT FOR FORMULATING LINEAR PROGRAMS

1. Introduction

Linear programming (LP) models have been used to support operational and planning decisions in every major industry - including petroleum, steel, forestry, manufacturing, communications and banking. However, applications have been limited by the difficulties in model formulation, management and interpretation (Johnson [1987]). Rapid advances in computing have made it possible for us to solve far larger models than we can understand. To maximize the benefits from the considerable progress that has been made in the development of efficient computational algorithms, we need to reduce the cost of modeling and improve both the quality of models and the decision processes associated with their use.

An important research goal is to help modelers cope with the complexity of large-scale models so that they can devote a greater proportion of their time to understanding the subject to be modeled and to analysing model results. Since successful models can have a long life and can be used by many different people, we also need to develop procedures to automate the production of documentation during the model building phase and to maintain it during subsequent revisions. Finally, we need to conduct behavioral studies to learn how information contained in models can be made more accessible to users so that they can gain deeper insights and make higher quality decisions.

Research aimed at automating model building and management has begun to generate useful results. New approaches to LP modeling and analysis are described in Balci [1985], Bisschop and Meeraus [1982], Breitman and Lucas [1987], Fourer, Gay and Kernighan [1987], Greenberg, Lucas, and Mitra [1986], Nance and Balci [1983] and Welch [1987]. Greenberg [1983] describes a system for analysing the structure of LPs and diagnosing problems. Geoffrion [1987] has developed a comprehensive framework for model building and management called structured modeling; some extensions appear in Dolk [1987] and Lenard [1986]. Binabsioglu and Jarke [1986] describes the use of knowledge-based techniques to help automate the process of formulating linear programming models.

There are six steps in using any modeling technique: recognition of a problem to be solved, identifying the solution technique to be used, formulating the model, linking the model parameters to data values, solving the model and analysing the results. The research described in this paper is concerned with the formulation phase of the modeling process. The focus is on linear programming formulations because

they are an important class of models in and of themselves and because they are rich in structure. Since there has been little work in this area, there are few guidelines as to what functions can and should be automated, and virtually no experience in designing and building an effective man/machine interface. To test our design strategy, we are building a prototype system, LPFORM, which helps automate the formulation process (Ma [1987], Murphy and Stohr [1986]).

This paper describes our approach to designing the man-machine interface and developing the theory needed to support enhanced modeling and analysis. The next section provides an overview of the design principles underlying the system and a brief description of the system itself. Section 3 describes the interface by means of a short example. Section 4 contains an overview of the reasoning process. Finally, Section 5 explains how various kinds of knowledge (about syntax, semantics, physical transformations and previously defined templates) are represented and used to help the formulation process.

2. Overview of System

To improve the modeling process, we must understand how people formulate and build models and how the computer can assist this process. Our goals are to reduce the amount of bookkeeping required formulate a model and to provide enough flexibility to enhance the creative processs of model building. How these goals can be achieved is a research question that can be resolved only empirically. However, we have proposed the following design strategy and are incorporating it in a prototype system. There are three major components:

- Multiple problem representations ranging from a graphical, non-mathematical representation to a more traditional algebraic representation and an annotated data dictionary.
- Support for a number of different problem-solving strategies that reduce the complexity of the formulation process (hierarchical decomposition, reuse of previously defined model components and a non-procedural, piece-meal approach to problem specification).
- Capabilities not available in other software (relational database, consistency checking, model management).

The interface supports several different styles of problem representation. The interplay between these specification modes and their potential for making the user's task less demanding requires further research. Some parts of the model specification may be easier to state in one form and others in another. For example, we suspect that the overall structure of a problem is most easily visualized and stated graphically. This will certainly be true of transportation network structures - up to a certain level of detail. Other requirements may best be stated in terms of previously stored parcels of knowledge in the form of constraint and problem templates e.g. "this problem involves a product-mix and several transportation sub-problems." Many requirements are best stated with an activity orientation - for

example "we have buying, selling and inventorying activities at each of our warehouses" (in LPFORM, three activity icons would be placed in the warehouse block). Still other problem requirements, and in particular, policy constraints, are best stated with a row orientation e.g. "the total tons of product produced in each factory has to be less than certain given limits" or "the unions require that the total overtime hours of each kind of labor be distributed equitably". LPFORM supports all of these different classes of input although, in its present form, row-oriented requirements have to be stated algebraically for the most part. A simple extension will allow more English-like expressions of policy requirements.

A major objective of the system is to reduce the complexity of large-scale modeling by providing a number of automated aids and by supporting different formulation strategies. Hierarchical decomposition is supported by allowing the modeler to depict the problem graphically in layers of increasing detail. Properties of objects at higher levels are automatically inherited by lower level objects. This is a top-down approach to modeling. A bottom-up approach is also supported. This involves building and testing component models and then combining them into larger models. LPFORM automates the saving and reuse of models and parts of models and supports interactive queries about models stored previously in the model bank. A final strategy, is to allow uses to define their model requirements as they come to mind rather than to impose any strict order.

Model builders also need support for a number of laborious tasks that must be performed during the modeling process. First, the LPFORM system provides a relational database for data retrieval and manipulation (Stohr [1986]). Second, since the formulation process itself is being supported, there are many opportunities to check the accuracy of the problem statement as it is being produced rather than after the final tableau has been generated (Murphy et al [1987]). Finally, the model management facilities include an on-line model dictionary and query facility and a method for automatically retrieving and combining model components to build larger models (Ma [1987]).

LPFORM translates from an iconic problem representation to an algebraic statement of the model and finally produces the input needed by solvers such as LINDO (Schrage [1987]) or IBM's MPSX (IBM [1975]). Figure 2-1 shows the structure of the prototype system.

The LPFORM Interface is being built on an IBM PC/AT class machine using a set of graphics tools written in the C programming language (EVA [1987]). A prototype of the LPFORM Analyser has been implemented in PROLOG (Ma [1987]). The two subsystems are loosely coupled. The Interface sends the results of the user specification to the Analyser in the form of statements in the LPSPEC language (Ma et al [1987]). Each LPSPEC statement captures a single action made by the user in the graphics interface. The next two sections of the paper describe these two subsystems in more detail.

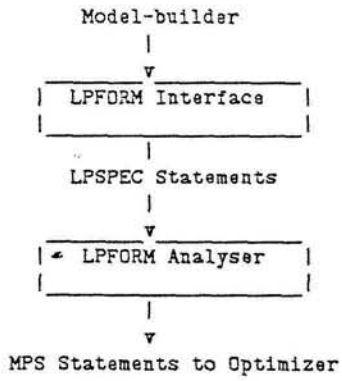

Figure 2-1: The LPFORM System

3. Graphical Representation

The graphical specification language uses blocks, links and templates to aid in model formulation. A model is represented by a collection of interconnected blocks. A block is a collection of zero, one or more linear programming activities that are separated in space or time from other activities. Blocks provide a convenient way for a modeler to decompose the real world. For example, one block may be a template representing the production of raw materials, a second may be a template representing the conversion of raw materials into finished products, and a third may represent consumption of finished goods. The links between blocks represent activities involving space or time transformations. A template is an LP submodel that can be simply a collection of constraints or a complete LP. A number of different standard models - transportation, product-mix, blending and so on - are stored in the model base and can be retrieved and combined into larger models.

Hierarchical layers are supported by allowing blocks to represent collections of blocks (or sub-blocks) plus their linkages. For example, a refinery model may be a single block in an initial LP and subsequently decomposed into blocks representing individual refinery units and links representing pipe connections. Finally, blocks can be replicated in either space of time.

The graphics interface is explained in detail in Ma et al [1987]. Figure 3-1 shows a sample screen.

The screen contains a central work area where graphs can be drawn and three sets of commands. The commands in the top border are used for major operations such as loading and saving the problem statement, accessing the database, moving up and down a hierarchical level in the model, and solving the model to produce the algebraic formulation and then, optionally, the mathematical solution.

The commands in the bottom border, operate on the graphical images in the center of the screen. These

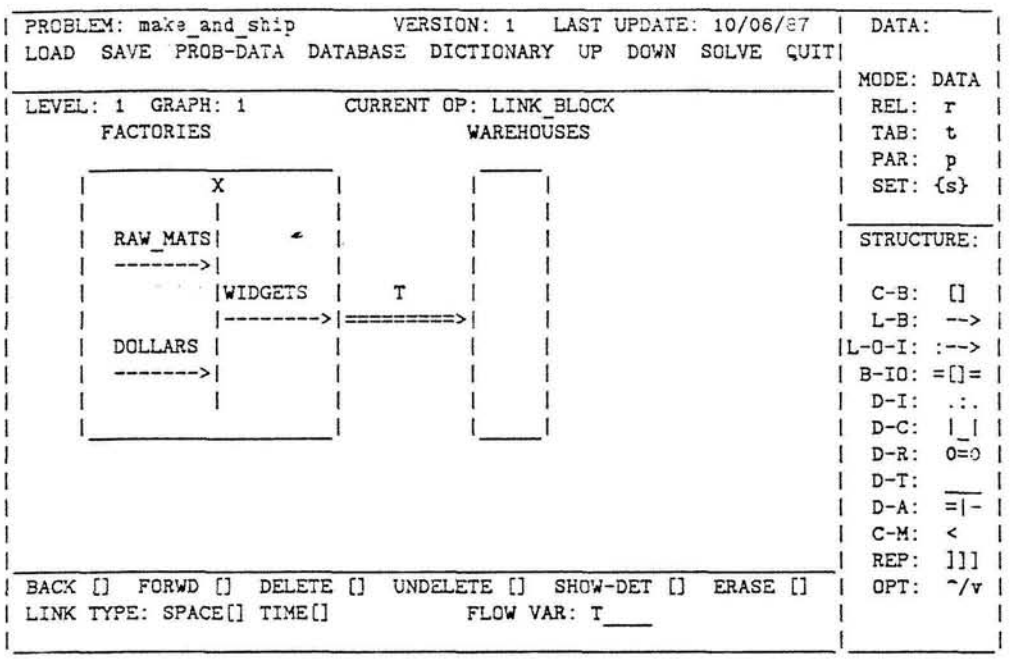

Figure 3-1: Graphic Screen for LPFORM

allow the user to step backwards and forwards through prior model building steps, to delete and restore model components, to show the detail underlying a part of the problem representation, and to erase the screen.

The commands used for modeling are on the right of the screen. The data commands in the upper part of the right-border allow the user to link the symbolic model to the data either interactively during the terminal session or by specifying links to existing data in a database. A relational database query language, which is part of the interface system, is used to perform queries and to manipulate the data.

The commands for defining model structure are in the lower part of the right-hand border and are associated with icons. To place an object on the screen, users point (with the cursor or another device such as a light-pen) to the command and then to a position on the screen. They are then led through a series of questions associated with the command or asked to fill in an electronic form to supply necessary textual information. The model building commands include Create-Block (C-B), Link-Block (L-B), Def-Activity (D-A) and Call-Model (C-M). The first command creates a block that can be filled with an activity or a template, the second links blocks for which the output of one is an input to another, the third defines a linear programming activity in terms of its inputs and outputs and the fourth places a template (submodel) into a block.

Figure 3-1 also shows the graph that would be constructed to define a production and distribution problem, "Make-and-ship", in which widgets, i, are produced at factories, f, and shipped to warehouses, w. The modeler places 'block icons' for the factories and warehouses on the computer screen, links the

two blocks to define the transportation activity, and then places an 'activity icon' in the factories block. As these actions are performed, the user is prompted for the names of the blocks, the decision variables (X and T) and the inputs and outputs of the production activity. Given the problem specification shown in Figure 3-1 the following algebraic statement will be produced:

$$\text{Min:} \quad \sum_{f \in Factories} \sum_{i \in Widgets} cst_{f,i} X_{f,i} + \sum_{f \in Factories} \sum_{w \in Warehouses} \sum_{i \in Widgets} tr_{f,w,i} T_{f,w,i}$$

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

$$-X_{f,i} + \sum_{w \in Warehouses} T_{f,w,i} = 0, \qquad \forall i \in Widgets, f \in Factories$$

$$\sum_{f \in Factories} T_{f,w,i} \geq d_{w,i} \qquad \forall i \in Widgets, w \in Warehouses$$

The remainder of this section explains this process in more detail. LPFORM processes a succession of problem representations: (1) the graphical screen interfaces presented to the user, (2) the LPSPEC language statements, (3) an internal tableau representation, (4) the algebraic statements output to the Matrix Generator, and finally, (5) the MPS form of the problem statement as input to the solver. Figures 3-3 through 3-6, which illustrate representations (2) and (3), contain computer outputs from a test run. The design of the user interface is discussed more fully in Ma et al [1987]. The model is developed using two alternative approaches. The first uses basic concepts of LPFORM to construct the model from 'first principles'. In other words, the components of the model, blocks, activities, links and so on, are described by the user in detail as in the illustration above. The second approach to model formulation involves the use of pre-existing LP model templates that are "mapped" on to the Make-and-ship problem. The formulations resulting from both approaches are the same.

3.1. First Principles Approach

Figures 3-1 through 3-5 show some of the screens generated by a user defining the Make-and-ship problem from first principles. The user decomposes the problem hierarchically into two layers. The first layer consists of an abstraction - there are only two blocks representing factories and warehouses and a single transportation link between them. The second level contains the representation of all the individual factories and warehouses of the real problem. If it is assumed that all factories can ship to all warehouses (or if infeasible links are given infinite costs in the data) there is no need to draw the second, more detailed, level of the problem. Each actual factory will inherit the properties specified for the parent "Factories" block, each actual warehouse will inherit the properties of the "Warehouses" block and the system will implicitly assume a completely connected graph. If these conditions are not true, the connected graph is automatically generated and the user can delete edges that do not apply. Alternatively, if the second level graph is complex, as will be the case in many real world applications, the user can specify an external table that specifies the arcs.

The first step is to define the highest level of blocks (for Factories and Warehouses) using the *CREATE-BLOCKS (C-B)* command and to link them using the *LINK-BLOCKS (L-B)* command. Figure 3-1 shows the screen at the end of this step.

Next, the user specifies the production activities in the Factories blocks using the *DEF-ACTIVITIES (D-A)* command. The screen in Figure 3-2 is obtained by pointing to the D-A command on the right of the screen and then to the Factories block. D-A is modeled after the activity modeling approach in Dantzig [1963], in which an activity set is defined by its inputs, outputs, activity coefficients, and objective coefficients (profits or costs). After the detailed specification of the activity has been completed, the network representation is restored, but the specified block is highlighted to indicate that it is not a simple demand, supply or transshipment point.

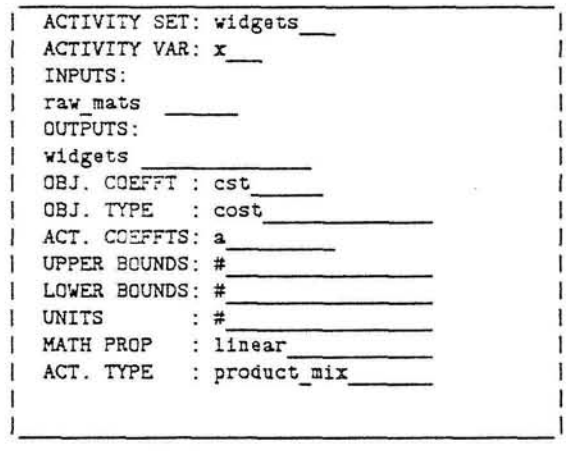

Figure 3-2: Define Production Activity

In the next step, the user specifies the members of the sets represented by Factories, Warehouses, Widgets and Rawmats using the *SET* command. The members of the set can be entered interactively (e.g. "W1, W2, W3."), or by a relational database query (e.g. "SELECT Warehouse_name FROM Warehouse WHERE Region = USA."). Here, Warehouses is an external relational database table containing Warehouse_name and Region fields (amongst others).

The final step in defining the problem is to specify the direction of optimization using the *OPT* command in the lower right corner of the screen. After this interaction, the user saves the problem (using the SAVE command in the top of the screen) and then selects the SOLVE command to generate the LPSPEC statements.

Each of the commands in the graphic interface has a corresponding LPSPEC statement that captures the information provided by the user. The LPSPEC statements from the above interaction are listed in

Figure 3-3 in the order in which they were generated by the user. The LPSPEC command processor can check for missing or ambiguous information and request clarification from the user.

```
>>>>>> Compiling LPSPEC File: mkship.fst <<<<<<

Statement No: 1
create_block(make_and_ship,[factories,warehouses])

Statement No: 2
link_block(space,explicit,[factories,warehouses],t)

Statement No: 3
def_transport(widgets,[factories,warehouses],1,#)

Statement No: 4
def_activity(factories,[widgets],x,[rawmats],[a],[widgets],cst,
cost,#,#,#,linear,product_mix)

Statement No: 5
def_set(factories,[north,south,west])

Statement No: 6
def_set(warehouses,[nwh,swh,wwh])

Statement No: 7
def_set(widgets,[w1,w2,w3])

Statement No: 8
def_set(rawmats,[r1,r2,r3])

Statement No: 9
optimize(min,make_and_ship,cost,symbolic)
```

Figure 3-3: LPSPEC for First Principles Approach

After compilation of the LPSPEC statements, the inferencing rules are invoked and the algebraic terms of the model are created and assembled into their proper positions in the algebraic statement by the LPFORM Analyser System. Finally, the model is cross-referenced to its data and to entries in the data dictionary.

The internal tableau representation (Figure 3-4) and model data dictionary (Figure 3-5) are the final result of the reasoning process in LPFORM.

```
PROBLEM/MODEL/FRAGMENT = make_and_ship.

ROW\COL      X(f,f)            T(f,w,i)              RHS
OBJ=         +S{f;i}cst[f;i]   +S{f;w;i}tr[f;w;i]    MIN
Use[f;r]     +S{i}a[f;r;i]                           < +b[f;r]
Bal[f;i]     +i[f;i]           -S{w}1[f;w;i]         = +0[f;i]
Supply[w;i]                    +S{f}1[f;w;i]         > +d[w;i]
```

Figure 3-4: Internal Tableau for Make-and-ship Model

The internal tableau can be used to check the formulation. It has been designed to display

simultaneously both the algebraic and the tableau (block) structure of an LP problem. If the problem is large, the display is spread across several screens.

In Figure 3-4, there are columns for each decision variable and rows for the objective and each constraint. The columns and rows are labeled by the decision variables and RHS constants together with their indices. Summations are identified by an 'S' followed by the list of indices over which the summation is to be performed enclosed by braces. Note that the balance equation for widgets has been correctly generated by the system.

The data dictionary for the Make-and-ship model is shown in Figure 3-5. The symbols on the left appear in the algebraic statement. The roles of some of the constraints have been inferred by the system. For example, LPFORM has recognized Factories as a "From-Block" and b as the right-hand-side of a "Use" constraint.

```
                * Symbol convention of make_and_ship *

Set Reference:
SYMBOL:     SET NAME:
-------------------------------------------------------------
f          : Factories
             Meaning: block, from_block.
r          : Rawmats
             Meaning: input.
w          : Warehouses
             Meaning: to_block.
i          : Widgets
             Meaning: output, commodity.

Activity Reference:
SYMBOL:     ACTIVITY (VARIABLE):
-------------------------------------------------------------
X(f,i)     : X(Factories,Widgets)
T(f,w,i)   : T(Factories,Warehouses,Widgets)

Coefficient Reference:
SYMBOL:     COEFFICIENT (DATA):
-------------------------------------------------------------
cst[f;i]   : Cst[Factories,Widgets]
a[f;r;i]   : A[Factories,Rawmats,Widgets]
i[f;w;i]   : I[Factories,Warehouses,Widgets]
i[f;i]     : I[Factories,Widgets]
O[f;i]     : O[Factories,Widgets]
b[f;r]     : Rhs?*^factories^rawmats[Factories,Rawmats]
d[w;i]     : Rhs?*^demand^warehouses^widgets[Warehouses,Widgets]
tr[f;w;i]  : Obj?*^factories^warehouses^widgets[Factories,
             Warehouses,Widgets]
```

Figure 3-5: Data Dictionary for Make-and-Ship Model

This is the basic representation from which problem statements for different matrix generators can be constructed in a straight-forward fashion. Note, however, that the problem representation above is

purely symbolic and contains no information on data values. This information must be included in the problem statements sent to the matrix generator. Basically, any data values input interactively by the user, for example, the elements of the sets in the above example, or the retrieval and data manipulation commands input by the user or inferred by the system, are output verbatim to the next phase of the model generation process. More details are given in Ma et al [1987].

3.2. Model Mapping Approach

A major goal of LPFORM is to allow model elements developed in one project to be reused in others. This facilitates a bottom-up approach to model building in which submodels are tested and then combined into larger models. LPFORM, contains a number of standard templates (for Transportation, Blending, Product-mix, Process-selection, etc.). Alternatively, users can build their own models using the first principles approach and store them in the model bank for later use.

Referring to Figure 3-1, it is intuitively obvious that the Make-and-ship model consists of a product-mix problem and a transportation problem. This approach is implemented in LPFORM using the *CALL-MODEL* (C-M) command. Figure 3-6 shows the formulation of the same problem using the model mapping approach.

The C-M command "maps" the names of indices, variables and data coefficients in the stored template into those that will be used in the new model. Thus, in this example, "from-block", "to-block", and "commodity" become, respectively, "factories", "warehouses" and "widgets". The standard templates are defined to be as general as possible. If the new model has no need for some of the indices in the stored template, they can be dropped. Conversely, the user can add indices to a model by using the Replicate command (see Ma et al [1987]).

4. Inference Process for Generating LP Models

This section of the paper explains how statements such as those shown in Figures 3-3 and 3-6 are converted into an algebraic problem statement. To ensure a robust and widely applicable approach, LPFORM focuses on properties that are valid across all LP's and brings in more subject-specific information only as needed. The properties of LPs can be divided into syntactic, semantic and domain-specific categories. Syntactic information consists of symbols and the rules that combine the symbols into a model. Semantic information incorporates knowledge about the physical entities being represented. In order to provide the maximum generality, we concentrate on the physical principles that underlie all linear programs. These include the properties of resources, commodities and networks, and ideas from the field of dimensional analysis (Kurth [1972]). Domain-specific information provides individual instances of

```
>>>>>> Compiling LPSPEC File: mkship.map <<<<<<

Statement No: 1
create_block(make_ship,[factories,warehouses])

Statement No: 2
def_set(factories,[north,south,west])

Statement No: 3
def_set(warehouses,[nwh,swh,wwh])

Statement No: 4
def_set(widgets,[w1,w2,w3])

Statement No: 5
def_set(rawmats,[r1,r2,r3])

Statement No: 6
call_model(transportation,make_ship,
     [from_block,to_block,commodity],[factories,warehouses,widgets],
     [flow],[t],
     [gain_or_loss],[1])

Statement No: 7
call_model(product_mix,make_ship,
     [block,input,output],[factories,rawmats,widgets],
     [volume],[x],
     [tech_coef,available_input],[a,b])

Statement No: 8
optimize(min,make_ship,cost,symbolic)
```

Figure 3-6: LPSPEC Statements for Model-Mapping Approach

the general concepts and information on special requirements associated with these instances. For example, in a job shop application, a lathe belongs to the class "machines", which is a subclass of "resources" at the most general level. Associated with the class of machines may be policies and constraints that can be automatically incorporated in the model by the system. Thus, domain-specific knowledge enhances the power of the system in aiding the formulation process. Our research to date concentrates on syntactic and semantic information, developing general principles applicable to all model building areas. The use of domain related knowledge in formulating LPs is described in Binbasioglu and Jarke [1986].

4.1. The Reasoning Process

The reasoning process has the following steps:

1. The user specifies the problem using the graphic interface; LPSPEC statements are generated.

2. The LPFORM Analyser parses the internal representation and performs some consistency checks.

3. The network structure of the problem is analysed; if the problem is a pure network, the

appropriate model template is invoked.

4. If there are non-network components (e.g. production activities associated with one or more blocks), the appropriate constraint fragments and templates are invoked to generate the "model pieces".

5. The model pieces are assembled into their positions in the algebraic representation; consistency checks are performed, redundant pieces are discarded and missing pieces are added.

6. The internal representation of the algebraic statement and the model data dictionary are generated.

7. The data coefficients in the problem statement are associated with values stored in external tables and relational databases.

8. The input to the matrix generator is generated.

9. The output of the tableau generator is fed to the solver and the problem is solved.

10. A report-writer and analysis system is invoked to process the results of the LP run.

4.2. Generation of the Model Components

Section 3 described the various modes of input available to the user. Much of the expertese embodied in the system consists of rules for translating from inputs given in these different forms into the algebraic components of the model. The basic components in the internal representation of LP constraints are "model pieces" which are algebraic terms with their associated summations. Larger model components such as constraints, subproblems and complete LP problems are simply collections of model pieces. This approach allows great flexibility in generating LPs and in combining templates of previously stored models into larger models.

The various objects in the user specification each generate one or more model pieces from more fundamental templates stored in the system's model base. Thus, an inventory icon results in a set of pieces embodying the standard inventory relationships, while a link between blocks contributes pieces corresponding to a transportation problem and so on. The model pieces comprising the standard component types are stored in a general form so that they can be tailored to fit the particular problem; thus, indices can be added or dropped as needed.

For example, in the graphical representation in Figure 3-1, "Warehouses" is an "exogenous demand" block (a block that has inputs but no outputs or internal activities). In the first principles approach, the following constraint fragment is automatically generated: $\geq d_{w,t}$. This consists of one "piece" (the inequality and the coefficient). Similarly, the Def-activity command in Figure 3-3 will generate a constraint fragment with two pieces:

$$\sum_{i \in Widgets} a_{f,r,i} X_{f,i} \leq b_{f,r}, \qquad \forall r \in RawMats, f \in Factories$$

A major area for research involves the discovery of appropriate standard components and the rules by which they can be generated from the user's input and synthesized into the complete LP.

4.3. Problem Synthesis Using Syntactic Information

We are developing rules for combining model pieces into a complete LP using syntactic information consisting primarily of index and variable names Murphy et al [1987]). This is analogous to solving a jigsaw puzzle, and we call the mechanism in Step 5 above, the "Puzzler".

We define an index set to be a collection of symbols denoting the indices (subscripts) associated with a variable or data coefficient. Each individual index identifies a 'dimension' of the data or variable and takes on values within some well-defined domain. For a given constraint and variable with coefficients in that constraint, let:

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

To illustrate these definitions, consider the following example:

$$\sum_{i} \sum_{j \in J(i)} a_{i,j,k} X_{i,j,k} \leq b_{k}, \quad \forall k$$

then V = {i,j,k}, C = {i,j,k}, S = {i}, P = {j}, Q = {i}, R = {k}.

In a properly specified LP where all indices are explicit:

(1) $\quad R = (V \cup C \cup Q) - (S \cup P)$.

This relationship helps LPFORM: (1) infer the complete problem from its component parts, (2) adjust the model when a part of the model is replicated in some dimension such as place or time, and (3) check the correctness of the final formulation.

Each model piece has row and column "labels". The row label for a piece includes its row index set (computed from the right-hand side of (1)) plus semantic information on the meaning of the indices and the physical units associated with the activity. The column label for a left-hand side piece consists of the variable name, the variable index set, V, and semantic information as for the row label. The output from

Steps 3 and 4, above, is a set of model pieces. In Step 5 the Puzzler searches through these pieces to determine the unique row and column labels. This establishes the overall dimensions of the tableau. Next, the pieces are assigned to the tableau based on their row and column labels. Redundant pieces are discarded and, where possible, missing pieces are automatically generated as described below. Finally, the Puzzler performs some consistency checks - for example, the body of the tableau must have at least one piece assigned to every row and column. The Puzzler can also warn the user if the model contains disjoint subproblems.

To illustrate the process for the Make-and-ship problem, consider the tableau representation in Figure 4-1.

Column Label:	[X,f,i]	[T,f,w,i]	RHS
Row Label:			
OBJ:	$\sum_i \sum_f cst_{f,i} X_{f,i} + \sum_f \sum_w \sum_i tr_{f,w,i} T_{f,w,i}$		
[f,r]	$\sum_i a_{f,r,i} X_{f,i}$		$\leq b_{f,r}$
[f,i]	$- X_{f,i} +$	$\sum_w T_{f,w,i}$	$= 0$
[w,i]		$\sum_f T_{f,w,i}$	$\geq d_{w,i}$

Figure 4-1: Illustration of Jigsaw Puzzle Reasoning

The first constraint is generated by the product-mix sub-problem and the third by the transportation sub-problem. In the second constraint, which represents the balance equation for widgets, the second term comes from the Transportation sub-problem. The first term in this constraint is supplied automatically by LPFORM. The rule used to do this is to search for matching column and row index sets and to supply a diagonal sub-matrix if no term is present in the corresponding position in the tableau. In this case, -1's are placed in the tableau.

The success of this process depends on the ability to distinguish all the rows and columns of the tableau. The column and row labels contain information in addition to the index sets because the latter do not necessarily contain enough information to ensure uniqueness. Some initial theoretical results concerning what can and cannot be done with index information are given in Murphy et al [1987].

5. Knowledge Representation

Knowledge representation affects both efficiency and functionality. Functionally, the system should: (1) provide a source of knowledge about the models in the model base and their structure. (2) provide guidance to users in the selection and composition of components of larger models, (3) facilitate the acquisition of domain specific knowledge as outlined below.

The basic entities of linear programs (activities, constraints, resources, commodities) and other organizational entities (blocks, links, and submodels) are represented by a network of frames. A frame is a stereotyped representation of some situation which is adjusted to a current situation by changing certain details (Winston [1984]). As a data structure, a frame consists of "slots" containing the descriptive attributes of the object, default values and or procedures that should be invoked if needed, and relationships to other objects. Thus, a frame for a complete LP contains slots with information on the objective, activities and constraints (a product-mix model, has slots noting that the objective is to maximize profit, the activities are production levels of different products, and the constraints are resource limits). In our context, another slot points to the model pieces associated with the template.

As discussed above, part of the knowledge contained in LPFORM is in the form of rules to generate the model pieces and to combine them into a problem statement. Other types of knowledge can be used, not only to help translate from the user input to the final algebraic statement, but also to provide more intelligent support to the model builder. Three important sources of knowledge are the types of physical transformation associated with activities, template models and domain-specific information.

Knowledge Concerning Transformations

Linear programming models involve three basic transformations: transformations in place, in time and in form. A transformation in form converts input "whats" into output "whats" while leaving place and time identical. Place and time transformations make analogous adjustments to "where" and "when". As shown in Figure 5-1, a transformation in form has indices corresponding to inputs and outputs, indices for "where" and "when" and an added index for "how" if there are alternative approaches. Transformations in place are distinguished by having indices on "what", "when" and "how", plus "from where" and "to where". Transformations in time have indices on "what", "where" and "how" and "from when" and "to when". In many cases, these indices are implicit rather than explicit - for example, production activities are usually indexed by their associated outputs i.e. only on the "to what" dimension. Compound transformations involving more than one of the fundamental transformations are possible.

PLACE	FORM	TIME
From where To where	Where	Where
What	From what To what	What
When	When	From when To when
How	How	How

Figure 5-1: Index Roles in Transformations

The data structure used to represent the index set for each variable and coefficient has slots for each of the roles in Figure 5-1. Information on the meaning of indices allows LPFORM to improve on the syntactic rules for composing LPs and to detect missing model pieces in some circumstances.

Knowledge in Templates

Each template has an associated frame with slots recording its usage in different models and pointing to its component model pieces. As discussed above, users can add to the model base at any time. A query facility allows users to browse through the the model base. It is possible, for example, to retrieve all models using a given resource or all with a common activity. A procedure to generate a precedence map of all models having common inputs and outputs, is being implemented. We are currently studying the appropriate set of standard templates to be included in the model base.

Domain-Specific Knowledge

Domain-specific knowledge can be used to make the system more powerful in particular applications. First, the data dictionary of general concepts (activities, resources, etc.) can be extended to include more terms thereby allowing users to state their problems in more familiar language. Thus, labor can be related to the general concept of resource by an "is-a" relationship and specific classes of labor to the more generic category, labor, and so on.

Second, the different classes of entitities can be associated with information that either must, or might, be applicable to particular models. Thus, with labor, any union restrictions must be embodied in constraints in the model, while various optional work force smoothing relationships might be incorporated at the user's discretion. This will allow the system to operate as an expert assistant suggesting possibilities for consideration by the modeler.

Finally, local terms can be related to an organization's data base. For example, each category of labor could be linked to tables containing current wage rates, standard piece rates and so on. This would allow the system to automatically generate database retrieval statements to link the symbolic statement of the problem to current data values.

6. Conclusion

Our strategy has been to develop a general approach to model formulation that is applicable to all LPs. We have done this by concentrating on syntactic and semantic properties. More powerful systems, suitable for non-expert users as well as expert users, will probably require the acquisition and use of domain-specific knowledge.

The prototype system is now operational and is able to solve a range of problems. There are a number of interesting questions concerning the approach that can only be resolved through usage. The most important of these is whether the approach outlined in this paper will, in fact, improve productivity and lead to higher quality models. Other questions concern the effectiveness of the graphic interface in reducing the cognitive load on the modeler and improving the quality of the models produced.

In the near future, we intend to test the LPFORM system and to improve its capability in the linear programming domain. After that, we will investigate the possibility of applying a similar approach to other modeling situations.

References

1. O. Balci (1985), "Requirements for Model Development Environments," Technical Report CS83022-R, Department of Computer Science, VPI&SU, Blacksburg, VA.

2. M. Binbasioglu (1986), "Knowledge Based Modelling and Support for Linear Programming," Ph. D. Dissertation, New York University.

3. M. Binbasioglu and M. Jarke (1986), "Domain-Specific DSS Tools for Knowledge-based Model Building," *Decision Support Systems*, vol 2, June-July.

4. J. Bisschop and A. Meeraus (1982), "On the Development of a General Algebraic Modeling System in a Strategic Planning Environment," *Math. Prog. Study 20*, 1-29.

5. R. L. Breitman and J. M. Lucas (1987), "PLANET: A Modeling System for Business Planning", *Interfaces*, Vol 17, No 1, Jan-Feb, 1987.

6. G. B. Dantzig (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.

7. D. Dolk (1987), "Model Management Systems: A Perspective," presented at NATO/ASI Conference "Mathematical Models for Decision Support", Val D'Isere, France.

8. EVA UIMS/GDB (1987), User Manual, Expert Vision Associates, Cupertino, California.

9. R. Fourer, D. M. Gay and B. W. Kernighan (1987), "AMPL: A Mathematical Programming Language," AT&T Bell Laboratories, Murray Hill, N.J.

10. A. M. Geoffrion (1987), "An Introduction to Structured Modeling," *Management Science*, vol

33, no. 5, pp. 547-588.

11. H. J. Greenberg (1983), "A Functional Description of ANALYZE: A Computer-Assisted Analysis System for Linear Programming Models." *ACM TOMS 9*, 18-56.

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

13. E. Johnson (1987), "Directions for Mathematical Programming Software and Integer Programming Reformulations," presented at NATO/ASI Conference "Mathematical Models for Decision Support", Val D'Isere, France.

14. R. Kurth (1972), *Dimensional Analysis and Group Theory in Astrophysics*, Pergamon.

15. M. Lenard (1986), "Representing Models as Data," *Journal of Management Information Systems*, vol 2, no 4 pp. 36-48.

16. P. Ma (1987), "An Intelligent Approach to Formulating Linear Programs", Ph.D. Dissertation, New York University.

17. P. Ma, F. H. Murphy and E. A. Stohr (1986), "The Science and Art of Formulating Linear Programs," *IMA Journal of Mathematics in Management*, (to appear).

18. P. Ma, F. H. Murphy and E. A. Stohr (1987), "Design of a Graphics Interface for Linear Programming Models," Working Paper, Center for Research in Information Systems, Graduate School of Business Administration, New York University.

19. F. H. Murphy and E. A. Stohr (1986), "An Intelligent System for Formulating Linear Programs," *Decision Support Systems*, Vol 2, No 1, Jan-Feb, 1986.

20. F. H. Murphy, E. A. Stohr and P. Ma (1987), "Composition Rules for Building Linear Programming Models from Component Models," Working Paper, Center for Research in Information Systems, Graduate School of Business Administration, New York University.

21. R. E. Nance and O. Balci (1983), "The Objectives and Requirements of Model Management," Technical Report CS83024-R, VPI&SU, Blacksburg, VA.

22. L. Schrage (1987), *Linear, Integer and Quadratic Programming with LINDO*, Scientific Press, Palo Alto.

23. E. A. Stohr (1985), "A Mathematical Programming Generator System" Working Paper, Center for Research in Information Systems, Graduate School of Business Administration, New York University.

24. J. S. Welch (1987), "PAM-A Practitioner's Approach to Modeling," *Management Science*, vol 33, no 5 pp 610-625.

25. P. H. Winston (1984), *Artificial Intelligence*, Addison-Wesley, Boston, MA.