Design of a Graphics Interface for Linear Programming

Pai-chun Ma
Graduate School of Business Administration
New York University
New York, New York

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

and

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

September 1986

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

Working Paper Series

CRIS #136

GBA #86-101

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

Table of Contents

1. Introduction 1
2. Stages in Solving an LP Problem 2
3. System Architecture 6
4. Comprehensive Example 9
    4.1. The Energy Model 10
    4.2. First-principles Approach 11
    4.3. Model Mapping Approach 21
5. Interface Design Strategy 24
    5.1. Problem Representation 25
    5.2. Reducing Cognitive Complexity 27
    5.3. Added Functionality 28
    5.4. Added Functionality 30

Reference 30

I. Appendix: Design of The Graphics Interface 32
    I.1. Top Border 34
    I.2. Right Border 35
    I.3. Bottom Border 38

List of Figures

Figure 2-1: Software coverage of conceptual stages 4
Figure 3-1: Integrated LP System Diagram 7
Figure 3-2: User Interface and Internal Problem Representations 8
Figure 4-1: Graphic View of Energy Problem 10
Figure 4-2: Mathematical Formulation of Energy Problem 12
Figure 4-3: Link Blocks at Set Level 13
Figure 4-4: Defining the Commodities to be Transported 14
Figure 4-5: Create Blocks at subsequent levels 15
Figure 4-6: Define Production Activity 16
Figure 4-7: LPSPEC for First Principles Approach 17
Figure 4-8: Internal tableau Representation of Energy Model 18
Figure 4-9: Data Dictionary for Energy Model 19
Figure 4-10: Formulation Statements for APL Tableau generator 20
Figure 4-11: MPS Format Problem Statement for Solver 21
Figure 4-12: Mapping a Transportation Model on to the Sources - Conversions Link 23
Figure 4-13: LPSPEC Statements for Model-Mapping Approach 23
Figure I-1: Primary Graphics Screen 33

1. Introduction

Linear programming (LP) was first used in 1947 in the context of planning activities in the military [7]. Since then it has become a well accepted and widely used optimization technique for various applications in industry. See Schrage [20] for examples. There have been continuing algorithmic improvements in LP. However, the technique for building LP models has not changed significantly during the last decade. Currently, formulating linear programs is an art requiring considerable expertise and painstaking attention to detail. A major problem is that of size. Large LPs may consist of thousands of decision variables and constraints. A small mistake may lead to an unbounded solution, infeasibilities that are hard to detect, or worse still, a plausible model that gives wrong results.

This paper describes the design of an interface for a software system (LPFORM) that uses artificial intelligence techniques to help operations researchers and managers formulate large linear programs (LPs). The design and knowledge-based aspects of LPFORM are described more fully in [16], [11], [12]and [13]. Here we describe the prototype interface system and show how it will be used by means of a comprehensive example. We discuss the interface broadly in terms of the problem representations and problem-solving strategies provided as well as the menus, screens, etc. of the physical interface.

LPFORM is intended to help users through the grueling steps of constructing the model, developing labeling schemes for the rows and columns and organizing the data that determine the coefficients. 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. The objective is to develop an interface that employs graphics to

depict real world objects and that allows users to define an LP problem in non-mathematical terms. Knowledge-based techniques are then used to interpret this problem statement, to build an internal representation of the model, and to generate the required input for an LP solver.

In Section 2, we describe the process of formulating an LP and discuss the kinds of problem representation and software systems that have been developed to help in this task. Section 3 describes the architecture of LPFORM. Section 4 provides a detailed example that illustrates many features of the proposed interface. With this background, we discuss the design strategy in Section 5. Finally, a fairly comprehensive description of the main screen design is given in the Appendix.

2. Stages in Solving an LP Problem

The process of solving an LP problem by computers can be decomposed into five conceptual stages: problem investigation, model formulation, data administration, algorithmic solution and report generation and analysis. We will be concerned only with the first four stages here. The report and analysis stage is covered in Greenberg ([10]). Each stage involves a translation between different representations of the problem or model (see Figure 2-1).

Stage 1, *problem investigation*, is fuzzy and informal due to the richness, variety, and ambiguity of the real world. It is possible that a human expert will always be required to perform this task. The reasoning process at this stage is unclear. However, experts can be observed performing the following: (a) developing a system diagram of blocks and arrows to depict the flows, inputs, outputs, and activities (Figure 4-1, (b) discovering sub-structures of the problem that are identical or similar to existing

models, (c) identifying the relevant data and constructing the appropriate data tables, (d) defining activity and constraint sets and (e) identifying special management policies and requirements. This 'scratch' information may be produced in a random order and may not be complete necessitating later revision. The graphical interface to LPFORM is designed to accept all of the above kinds of information.

Stage 2, *model formulation*, uses the above information to create a symbol convention (labeling scheme) together with a dictionary for sets, variables, and coefficients, and to generate a symbolic formulation in algebraic notation and/or a matrix format. At the moment this step is performed by humans because the given information is often vague and because it requires expertese on the nature of LP models, and knowledge of the syntax of the input language for the tableau generator. LPFORM attempts to automate this part of the formulation process.

Stage 3, *data administration*, associates numeric data values with the algebraic formulation and prepares the 'matrix deck' for the tableau generator. Two major tasks must be performed. The *data collection* task requires human expertise because data sources are neither unique nor standard. For example, if a cost coefficient is stored in a relational database, then the formation of a database query to generate it depends highly on the modeler's knowledge of the database. Eventually, we hope to be able to generate the correct query automatically - however this is a longer range research goal. The *problem statement generation* task is usually partially mechanized. This takes the given data, (labeling scheme, algebraic form and coefficients), and constructs the LP problem statement according to the needs of a particular tableau generator. Each data coefficient in the symbolic statement must be instantiated by explicit arithmetic assignment, by table declaration, or by a database query.

Stage 4, *algorithmic solution* is the most easily mechanized. The solution algorithm used by the solver ("optimizer") is relatively trouble free due to the progress made by management scientists and the advancement of computer technology.

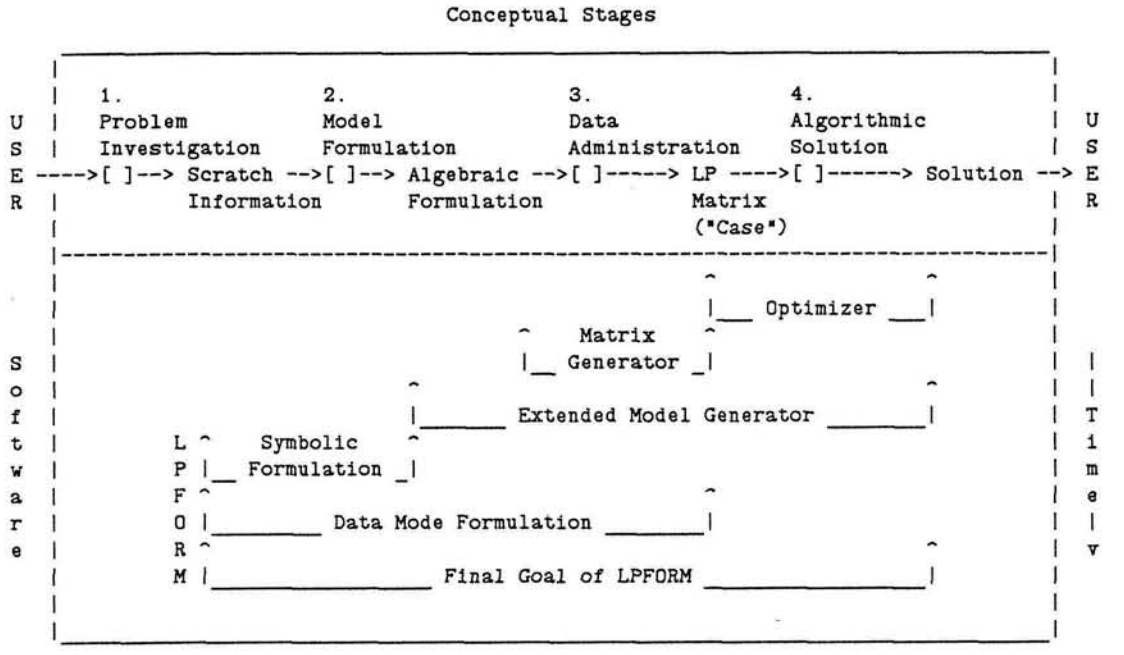

Figure 2-1: Software coverage of conceptual stages

Historically, the first software systems were solvers or "optimizers", built around the algorithm codes. These usually accept input as a listing of columns from the LP matrix, and output vectors of activity levels for both the primal and dual [8]. Some improvements in the input format (the "matrix deck"), documentation of output, and solution algorithms were made later. Systems (such as MPSX, [15] and Apex, [1]) deal at the level of numeric data. The MPS input format used by MPSX is a kind of standard which allows for the sparsity of the matrix (Figure 4-11). However this is a poor human interface since every nonzero element of the matrix must be individually specified by stating its value and column and row labels.

The second group of software systems are matrix generators which help automate the clerical task of constructing the "matrix deck". They accept data tables and a set of specially designed statements as input. However, the translation from the algebraic formulation to the matrix generator statement still needs to be done by hand. Example software systems of this group are OMNI [17] and DATAFORM [5].

The third group of LP software systems are extended model generators. They provide data manipulation facilities and/or use an algebraic representation as a modeling language. Data manipulation capabilities include data editing, acquiring data from databases, model editing, report generation and analysing the solution. Examples of systems in this class are AMPS [8], PAM [19], GAMS [14] and PLATOFORM [18]. Data generators for particular classes of problem have also been developed [9] and [4]. Finally, there are systems written in PL/1 [22] and APL [6] that provide data generation capabilities for MPSX.

LPFORM follows the trend in LP software by extending software support into earlier formulation stages. It accepts graphical and data-related information from the user and translates this into an algebraic formulation. In symbolic mode, only the algebraic formulation is generated. In data mode the data coefficients in the algebraic statement are bound to values in the database. We are not aware of any other running expert system to assist in generating LP problem statements. However, Binbasioglu, [3] has developed a knowledge-based formalism to describe and define LPs in the domain of manufacturing production.

We conclude this section by comparing the various alternative input representation schemes. It is quite inefficient for a modeler to work on the *matrix form* of any

problem with reasonable complexity [8]. The matrix form does not have an appropriate level of generality for a modeler who is developing the initial design or making major revisions. The *algebraic form* is a much more efficient vehicle for expressing the model in the first place and for understanding its general structure. Moreover, the algebraic form is concise, provides good documentation and is independent of particular data values. It has three draw-backs as a representation scheme. First, it is still somewhat tedious to specify large models correctly (attention has to be taken to ensure that indices range over the correct sets and that all relevant equations are included and so on). Secondly, it does not give an immediate visual picture of the model as does the graphic form described below. Finally, and most importantly, it is not understandable by anyone other than an LP expert. The *graphical interface* described in Section 4 and the Appendix, involves a completely different interface. It will employ icons representing real world entities such as inventories, demand and supply points and material flows plus other icons that represent the more abstract activities of a linear program and templates of previously defined problems.

3. System Architecture

Figure 3-1 shows the system architecture of the LPFORM system. The diagram follows the stages for formulating and solving LP problems given in the last section.

The integrated system will be composed of five systems loosely coupled via communicating files:

1. The LPFORM system, ([16], [11]).

2. A Tableau Generator [23], which is similar in function to the GAMS system, [14] (both take an algebraic approach).

3. IBM's MPSX system for solving linear and integer mathematical programs [15] (or the LINDO system, [20]).

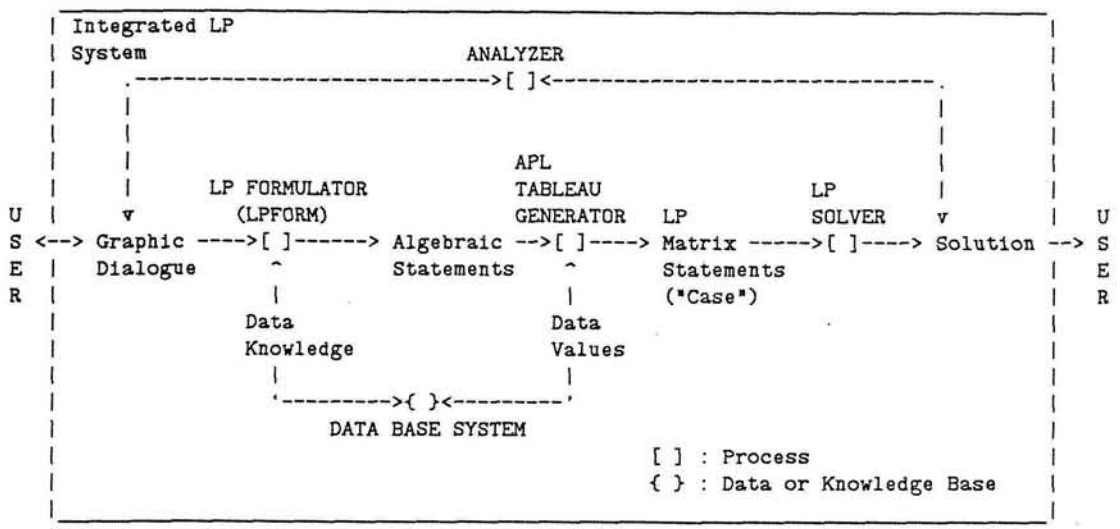

Figure 3-1: Integrated LP System Diagram

4. IBM's SQL database management system (DBMS) [2].

5. A tableau solution analyzer (ANALYZE, [10]).

At the current stage of development, only the first three systems have been linked.

Figure 3-2 shows the major components of the LPFORM subsystem. All of these components exist in prototype form and are being tested as part of our current project work. The one exception to this is the Graphics Subsystem which is described in this paper and on which work has only just begun.

LPFORM allows a number of different problem representations to be generated and displayed to the user during the problem formulation process. These are illustrated in the top half of Figure 3-2. The user first defines the problem using a graphical symbolism. This is translated into LPSPEC statements (Figure 4-7). As the work proceeds, the user may view a graphic showing the hierarchical and network structure of the problem or, alternatively, a 'picture' of the tableau in a highly summarized form, Figure 4-8. Finally, the user may wish to inspect the data dictionary (Figure 4-9) or the al-

gebraic statement of the problem (Figure 4-10).

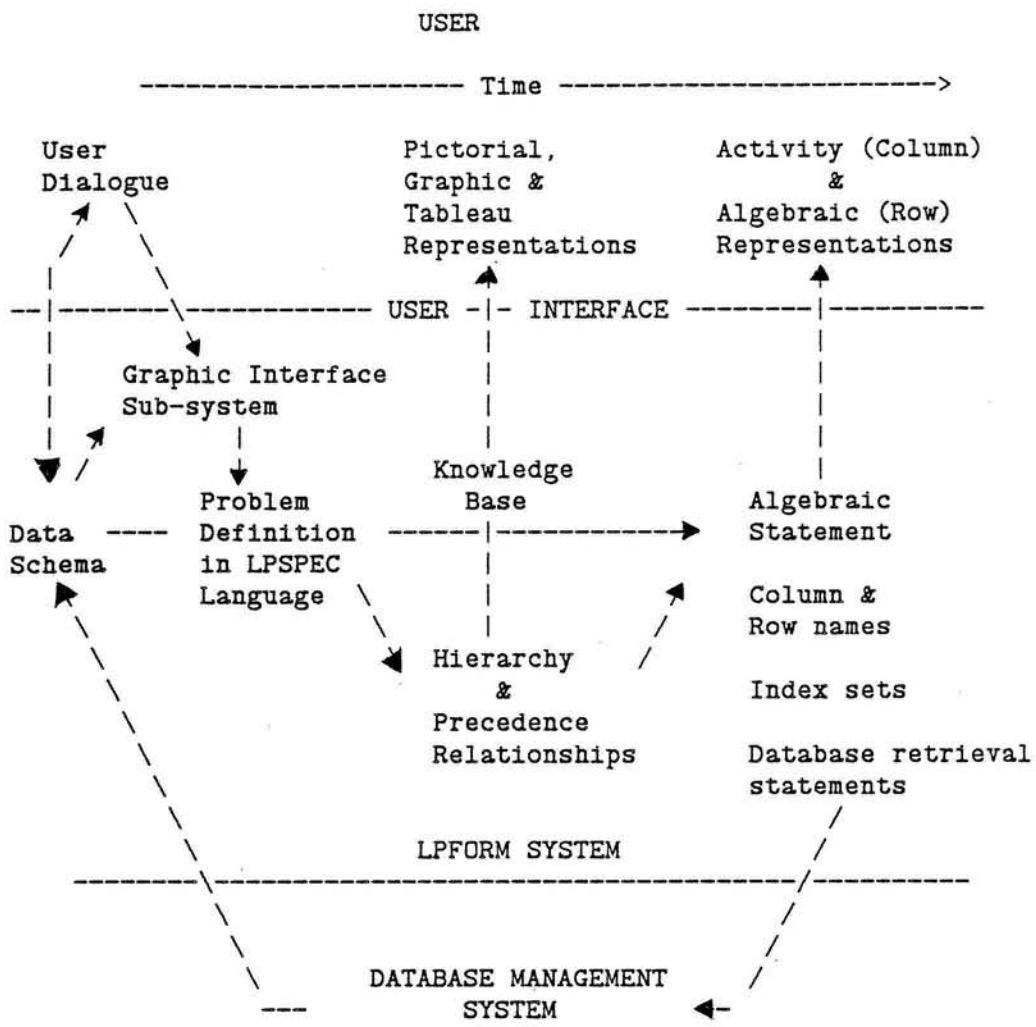

Figure 3-2: User Interface and Internal Problem Representations

Internally, the system maintains the data structures shown in the lower half of Figure 3-2. A knowledge base internal to the system is used to translate between these internal and external representations.

4. Comprehensive Example

This section of the paper contains an example in which LPFORM is used to model an energy problem. This is explained through the following sequence of eight different problem representations: (1) a literal description of the problem, (2) a graphical view of the problem, (3) the mathematical formulation as it might appear in a journal article or modeler's notebook, (4) the graphical screen interfaces presented to the user, (5) the LPSPEC language statements, (6) an internal tableau representation, (7) the algebraic statements output to the Tableau Generator, and finally, (8) the MPS form of the problem statement as input to the solver. The final five representations are part of LPFORM. The figures illustrating representations (5) through (8) contain computer outputs from a test run. The design of the user interface is outlined in the Appendix, the LPSPEC language is discussed in detail in [12], and the APL Tableau Generator in [23].

The same model is developed using two alternative approaches. The first uses basic concepts of LPFORM to construct the model from 'first principles'. The second involves the use of pre-existing LP model templates which are 'mapped' on to the energy problem. The interactive interface will be illustrated in detail only for the first-principles approach. The first-principles approach is discussed in 4.2 and the model mapping approach in 4.3. The formulations resulting from both approaches are the same.

These two approaches are at two opposite ends of a make-your-own/rely-on-others spectrum. There are in general, many other ways to build models in LPSPEC including the obvious compromises between the above two approaches. The other main dimension on which formulations can differ for the same problem involves the question of

whether the system is used in 'symbolic' or 'data' mode, [11]. The examples only demonstrate the symbolic approach.

The Appendix describes the design of the main graphics screen.

4.1. The Energy Model

Representation 1: English Narrative

The model is concerned with the optimal distribution and production of energy for the national economy. There are alternative sources (foreign and domestic) of raw energy (oil, gas, and coal). Each form of raw energy is transported to all the conversion centers (refineries and electric-utilities), and sinks (residential, transportation, and industrial). The conversion centers take raw energy as input and produce processed energy (gasoline and electricity, respectively) as output. The production processes at the conversions can be modeled as simple 'product-mix' problems. Each conversion center has its own technology coefficients for converting raw energy to processed energy. The processed energy is transported to each of the three sinks. Each transportation route for each type of raw energy has its own cost. Find the least cost transportation and production pattern.

Representation 2: Graphical

The problem is sketched in graphical form in Figure 4-1. The 'block' for conversions is marked to indicate the existence of production activities.

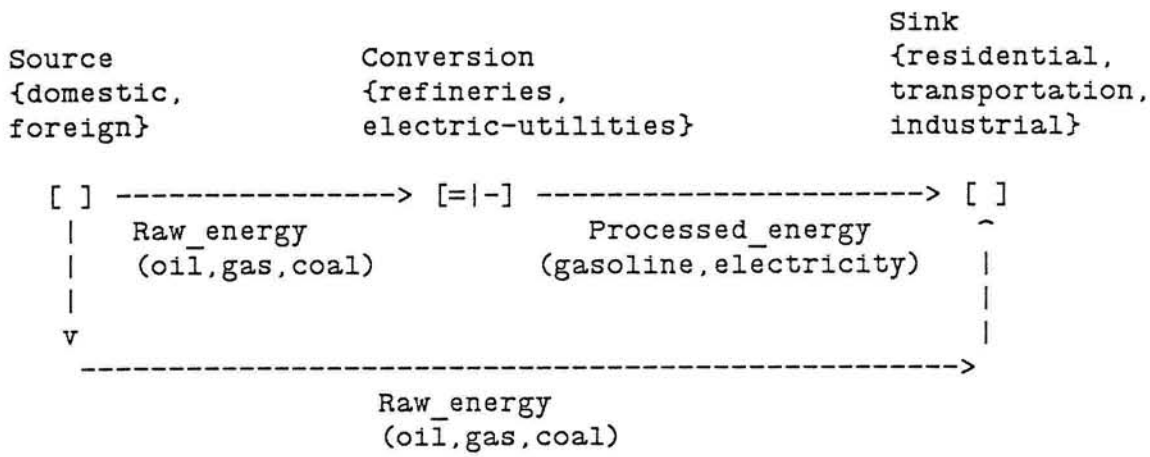

Figure 4-1: Graphic View of Energy Problem

Representation 3: Mathematical Notation

Let us define the following decision variables:

$TSC_{so,co,re}$ amount of raw energy, re, transported from source, so, to conversion, co.

$TSS_{so,si,re}$ amount of raw energy, re transported from source, so, to sink, si

$TCS_{co,si,pe}$ amount of potential energy, re, transported from conversion, co to sink, si.

$X_{co,pe}$ amount of processed energy, pe, produced at conversion center, co

Then Figure 4-2 gives the algebraic formulation for this problem in the usual mathematical notation.

4.2. First-principles Approach

In this section we follow the steps taken by a typical user of LPFORM in formulating the above LP problem.

Representation 4: Graphical Interface To Lpform

Figures 4-3 through 4-10 show some of the screens generated by a user defining the energy problem from first principles. The order in which the problem is defined is somewhat arbitrary. The strategy, in this case, was to decompose the problem hierarchically into two layers. The first layer consists of generalizations - Sources, Conversions and Sinks and the transportation links between them. The second layer is a more detailed representation of the problem in terms of the various types of source (domestic and foreign), conversion(refineries and electric utilities) and sinks (residential and industrial). This is actually the level of detail at which the model will be instantiated. The advantage of this top-down approach is that it simplifies the problem definition because lower levels of the problem inherit properties of higher levels automatically.

$$\text{MIN} \sum_{co \in Conversion} \sum_{pe \in Processed\_energy} cc_{co,pe} X_{co,pe}$$
$$+ \sum_{so \in Source} \sum_{co \in Conversion} \sum_{re \in Raw\_energy} tcscr_{so,co,re} TSC_{so,co,re}$$
$$+ \sum_{so \in Source} \sum_{si \in Sink} \sum_{re \in Raw\_energy} tcssr_{so,si,re} TSS_{so,si,re}$$
$$+ \sum_{co \in Conversion} \sum_{si \in Sink} \sum_{pe \in Processed\_energy} tccsp_{co,si,pe} TCS_{co,si,pe}$$

SUBJECT TO

Raw Energy Supply Constraint:

$$\sum_{co \in Conversion} TSC_{so,co,re} + \sum_{si \in Sink} TSS_{so,si,re} \leq ssr_{so,re'} \; \forall \begin{cases} re \in Raw\_energy \\ so \in Source \end{cases}$$

Balance Constraint - Processed Energy:

$$X_{co,pe} + \sum_{si \in Sink} TCS_{co,si,pe} \geq 0, \; \forall \begin{cases} pe \in Processed\_energy \\ co \in Conversion \end{cases}$$

Material Balance - Production of Processed Energy using Raw Energy

$$- \sum_{pe \in Processed\_energy} tc_{co,re,pe} X_{co,pe} + \sum_{so \in Source} TSC_{so,co,re} \geq 0, \; \forall \begin{cases} re \in Raw\_energy \\ co \in Conversion \end{cases}$$

Demand Constraint for Raw Energy at Sink:

$$\sum_{so \in Source} TSS_{so,si,re} \geq dsr_{si,re'} \; \forall \begin{cases} re \in Raw\_energy \\ si \in Sink \end{cases}$$

Demand Constraint for Processed Energy at Sink:

$$\sum_{co \in Conversion} TCS_{co,si,pe} \geq dsp_{si,pe'} \; \forall \begin{cases} pe \in Processed\_energy \\ si \in Sink \end{cases}$$

Figure 4-2: Mathematical Formulation of Energy Problem

The main screen used in the interaction (see Figure 4-3) is explained in detail in the Appendix I. Briefly, the commands in the right border are used to define the data and structure of the LP. To place an object on the screen, USERS point (with the cur-

sor 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.

In the energy example, the first step was to define the highest level of blocks (for Sources, Conversions and Sinks) using the *CREATE-BLOCKS (C-B)* command and to link them using the *LINK-BLOCKS (L-B)* command. Figure 4-3 shows the screen at the end of this step. Note that the linkages are directed.

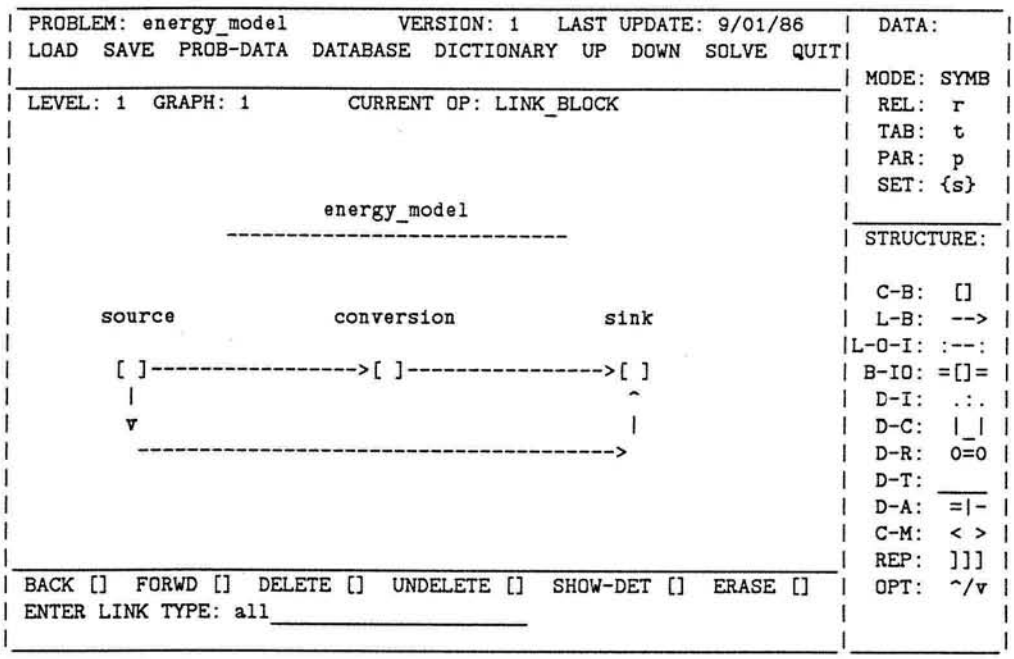

Figure 4-3: Link Blocks at Set Level

In the second step, the user specifies the commodities that are transported on each arc using the *DEF_TRANSPORT* command as shown in Figure 4-4

In the third step, the user defines the next lower level of blocks (Figure 4-5). If no further links or activities are defined at this level, the lower level blocks automatically inherit all the properties of their parents. The linkages (and commodities transported) at this level do not have to be specified in detail but are inherited (by default) from

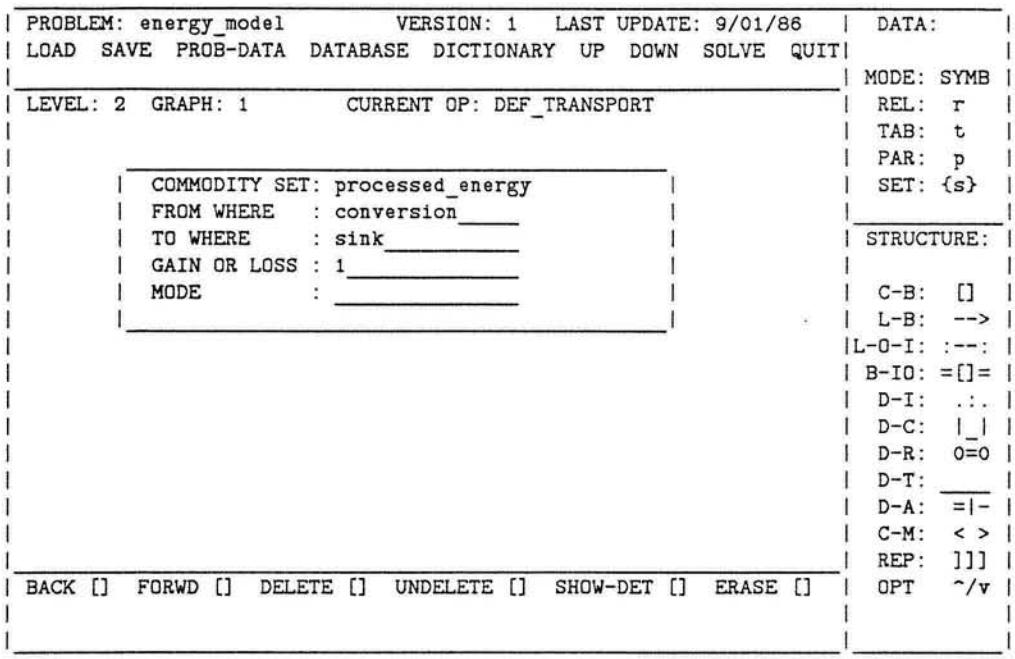

Figure 4-4: Defining the Commodities to be Transported

those in the top level diagram. Thus, the single link between Sources and Conversions in the first level is translated by the *ALL* parameter of the *LINK-BLOCKS* command at level 1 to the four arcs shown between the Sources and Conversions in the second level (Figure 4-5). At this point, the network structure of the problem is completely specified.

In the fourth step, the user specifies the production activities in the conversion blocks using the *DEF-ACTIVITIES* command. The screen in the upper part of Figure 4-6 is obtained by pointing to the electric utilities block and then to the D-A (*DEF-ACTIVITY* command) on the right of the screen. This is modeled after the activity modeling approach in [7]. An activity is defined by its inputs, outputs, activity coefficients, and objective coefficients (profits or costs). Some slots in Figure 4-6, are optional and are treated as hints to the user to enter the relevant information for an activity. The user simply spaces over slots that are not relevant. An unknown value,

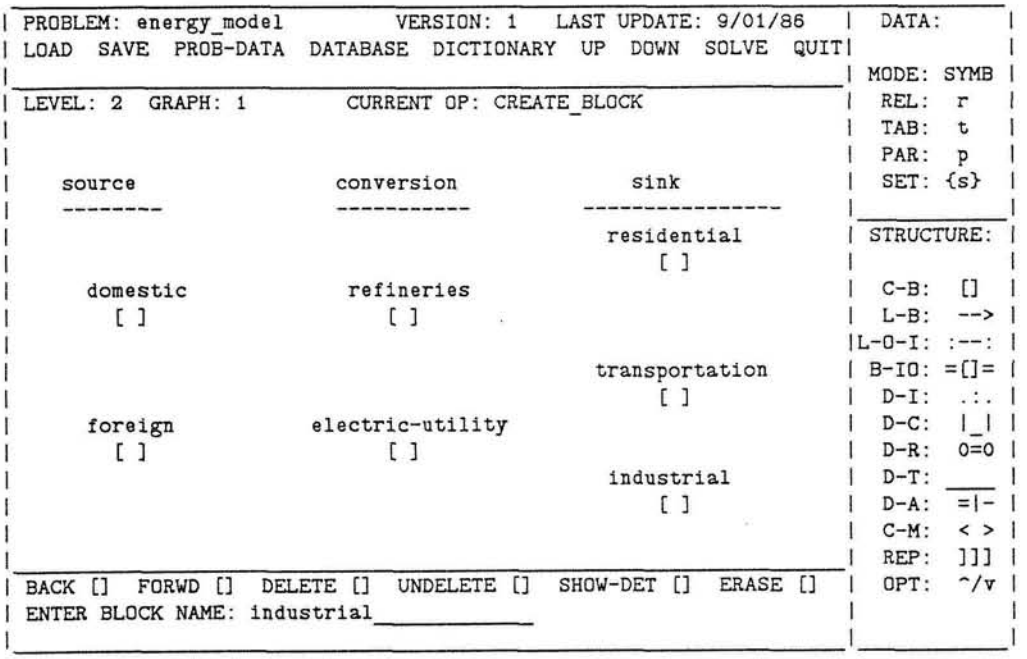

Figure 4-5: Create Blocks at subsequent levels

"?", can be entered in a slot if the user knows that a value is needed but can not specify it immediately. 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.

In the fifth step, the user specifies the types of commodities in more detail using the *DEF-SETS* command. This is a simple prompt-response type of interface in which, for example, the user specifies that Processed_Energy is really a set of different commodities (Electricity and Gasoline).

The sixth and 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

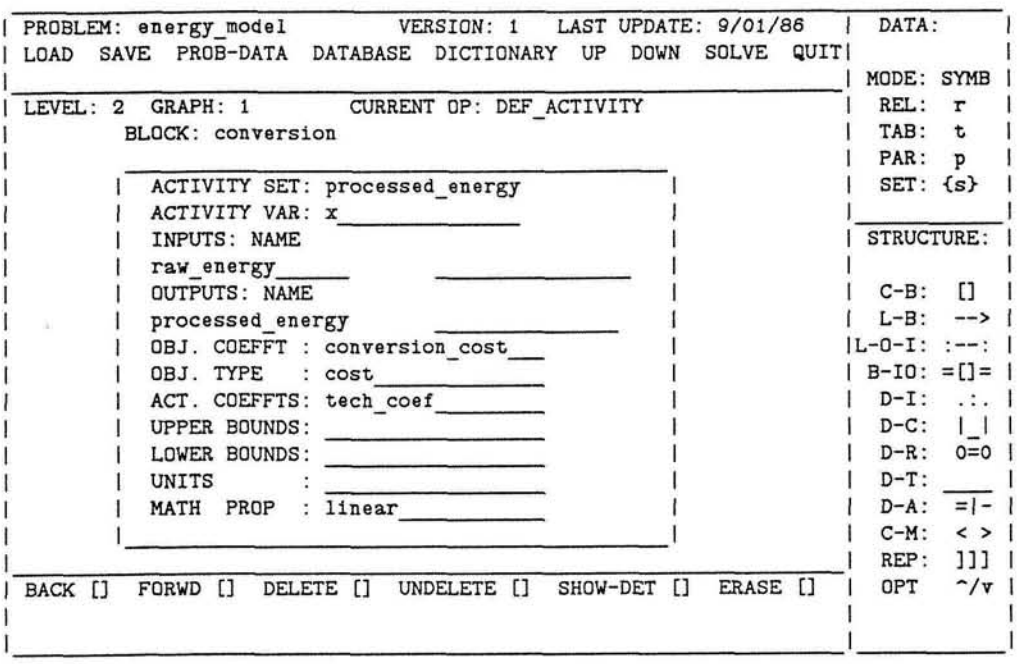

Figure 4-6: Define Production Activity

statements.

REPRESENTATION 5: LPSPEC STATEMENTS

Each of the commands in the graphical interface has a corresponding LPSPEC statement that captures the information provided by the user. The LPSPEC statements generated by the above interaction are listed in Figure 4-7 in the order that they were generated by the user's interaction. The LPSPEC command processor can check for missing or ambiguous information and request clarification from the user.

In LPSPEC, the value, "#", represents information which is not available; inferences regarding this slot will be suppressed.

Representation 6: Internal Tableau

After compilation of the LPSPEC statements, the inferencing rules are invoked, and the model pieces are created and assembled into their proper positions in the

```
create_block(energy_model,[source,conversion,sink]).

link_block(all,source,conversion,tsc).
link_block(all,source,sink,tss).
link_block(all,conversion,sink,tcs).

def_transport(raw_energy,source,conversion,1,#).
def_transport(raw_energy,source,sink,1,#).
def_transport(processed_energy,conversion,sink,1,#).

create_block(source,[foreign,domestic]).
create_block(conversion,[refinery,electric_utility]).
create_block(sink,[residential,transportation,industrial]).

def_activity(conversion,processed_energy,x,[raw_energy],[tech_coef],
             [processed_energy],conversion_cost,cost,#,#,#,linear).

def_set(raw_energy,[oil,gas,coal]).
def_set(processed_energy,[gasoline,electricity]).

optimize(min,energy_model,cost,symbolic).
```

Figure 4-7: LPSPEC for First Principles Approach

tableau by the puzzler (see [13]. Finally, the model is cross-referenced to its data and to entries in the data dictionary.

The internal tableau representation (Figure 4-8) and model data dictionary (Figure 4-9) are the final result of the reasoning process in LPFORM. This is the basic representation from which problem statements for different tableau generators can be generated in a straight-forward fashion. One example is shown in Figure 4-10.

The internal tableau is most useful for an expert wishing to check the results of the model building process. 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. It may also be sent to the printer for documentation purposes.

In Figure 4-8, there are columns for each decision variable and rows for the objective and each constraint. The columns and rows are labelled 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 between raw energy being moved from source to conversion and used as production input at conversion is correctly synthesized. Furthermore, the balance between processed energy being produced from hypothetical conversion centers and moved from conversion centers to sinks is also synthesized.

```
PROBLEM/MODEL/FRAGMENT = energy_model.

ROW\COL        X(co,pe)              TSC(so,co,re)
OBJ=           +S{co;pe}cc[co;pe]    +S{so;co;re}tcscr[so;co;re]
Use[so;re]                           +S{co}1[so;co;re]
Use[co;pe]     +1[co;pe]
Supply[co;re]  -S{pe}tc[co;re;pe]    +S{so}1[so;co;re]
Supply[si;re]
Supply[si;pe]

 ..................................... continued .....................................

ROW\COL        TSS(so,si,re)                 TCS(co,si,pe)                 RHS
OBJ=           +S{so;si;re}tcssr[so;si;re]   +S{co;si;pe}tccsp[co;si;pe]    MIN
Use[so;re]     +S{si}1[so;si;re]                                           < +ssr[so;re]
Use[co;pe]                                   -S{si}1[co;si;pe]             > +0[co;pe]
Supply[co;re]                                                              > +0[co;re]
Supply[si;re]  +S{so}1[so;si;re]                                           > +dsr[si;re]
Supply[si;pe]                                +S{co}1[co;si;pe]             > +dsp[si;pe]
```

Figure 4-8: Internal tableau Representation of Energy Model

The data dictionary for the energy model is shown below. The set names shown on the left help clarify the meanings of the variables and coefficients. A facility will be developed to allow the user to annotate the model dictionary with longer names. Note that the ? sign accompanying the data items indicates that the symbols have not yet been bound to their data.

Representation 7: Algebraic Statements For External Tableau Generator

Figure 4-10 shows the algebraic statements that are output to the APL Tableau generator. Statements beginning with an '*' are comments and those beginning with an 'E' are executable APL statements used to generate (or retrieve) and process the data

```
                    * Symbol convention of energy_model *

Set Reference:
SYMBOL:          SET NAME:
-------------------------------------------------------------------
co             : Conversion
                 Meaning: block, transhipment_node.
pe             : Processed_energy
                 Meaning: output, commodity.
re             : Raw_energy
                 Meaning: input, commodity.
si             : Sink
                 Meaning: to_block.
so             : Source
                 Meaning: from_block.

Activity Reference:
SYMBOL:          ACTIVITY (VARIABLE):
-------------------------------------------------------------------
X(co,pe)       : X(Conversion,Processed_energy)
TSC(so,co,re)  : TSC(Source,Conversion,Raw_energy)
TSS(so,si,re)  : TSS(Source,Sink,Raw_energy)
TCS(co,si,pe)  : TCS(Conversion,Sink,Processed_energy)

Coefficient Reference:
SYMBOL:          COEFFICIENT (DATA):
-------------------------------------------------------------------
cc[co;pe]      : Conversion_cost[Conversion,Processed_energy]
1[so;co;re]    : 1[Source,Conversion,Raw_energy]
1[so;si;re]    : 1[Source,Sink,Raw_energy]
1[co;si;pe]    : 1[Conversion,Sink,Processed_energy]
1[co;pe]       : 1[Conversion,Processed_energy]
0[co;pe]       : 0[Conversion,Processed_energy]
tc[co;re;pe]   : Tech_coef[Conversion,Raw_energy,Processed_energy]
0[co;re]       : 0[Conversion,Raw_energy]
ssr[so;re]     : Rhs?*^supply^source^raw_energy[Source,Raw_energy]
dsr[si;re]     : Rhs?*^demand^sink^raw_energy[Sink,Raw_energy]
dsp[si;pe]     : Rhs?*^demand^sink^processed_energy[Sink,Processed_energy]
tcscr[so;co;re] : Obj?*^source^conversion^raw_energy[Source,Conversion,Raw_energy]
tcssr[so;si;re] : Obj?*^source^sink^raw_energy[Source,Sink,Raw_energy]
tccsp[co;si;pe] : Obj?*^conversion^sink^processed_energy[Conversion,Sink,
                  Processed_energy]
```

Figure 4-9: Data Dictionary for Energy Model

for the problem. In Figure 4-10, the execute statement is used to generate the needed sets. The DATA= statement checks to see if the data items needed by the problem are present in the workspace. If they are not present, or if they have nonconformable dimensions, the user is asked to input the data items interactivly. Note that this provides a means to run LP programs that were defined in 'symbolic mode' in LPFORM. The body of the LP specification consists of the objective and constraint sets separated by comment lines. The correspondence between this part of the problem

statement and normal mathematical notation is fairly close. The "S" symbol stands for the Greek sigma character used to specify a summation. The limits of the summations are specified in the following line (as is common in mathematical notation). The "FOR" lines specify the index sets for the rows of the constraint sets.

```
*NAME energy_model
* CONVERSION = 'refinery,electric_utility'
E CONVERSION <- 1 THRU NCO <- 2
* PROCESSED_ENERGY = 'gasoline,electricity'
E PROCESSEDENERGY <- 1 THRU NPE <- 2
* RAW_ENERGY = 'oil,gas,coal'
E RAWENERGY <- 1 THRU NRE <- 3
* SINK = 'residential,transportation,industrial'
E SINK <- 1 THRU NSI <- 3
* SOURCE = 'foreign,domestic'
E SOURCE <- 1 THRU NSO <- 2
*
DATA= NCO,NPE,NRE,NSI,NSO
DATA= CC(NCO#NPE),TCCSP(NCO#NSI#NPE),TCSCR(NSO#NCO#NRE),TCSSR(NSO#NSI#NRE)
DATA= DSP(NSI#NPE),DSR(NSI#NRE),SSR(NSO#NRE),TC(NCO#NRE#NPE)
*
VAR=TCS(co,si,pe), co in Conversion, si in Sink, pe in Processedenergy
VAR=TSC(so,co,re), so in Source, co in Conversion, re in Rawenergy
VAR=TSS(so,si,re), so in Source, si in Sink, re in Rawenergy
VAR=X(co,pe), co in Conversion, pe in Processedenergy
*
MIN
  S S cc[co;pe]X(co,pe) +S S S tcscr[so;co;re]TSC(so,co,re)
: +S S S tcssr[so;si;re]TSS(so,si,re) +S S S tccsp[co;si;pe]TCS(co,si,pe)
co in Conversion, pe in Processedenergy, so in Source, co in Conversion,
: re in Rawenergy, so in Source, si in Sink, re in Rawenergy, co in Conversion,
: si in Sink, pe in Processedenergy
*
FOR so in Source FOR re in Rawenergy
  S TSC(so,co,re) +S TSS(so,si,re) < ssr[so;re]
co in Conversion, si in Sink
*
FOR co in Conversion FOR pe in Processedenergy
  X(co,pe) -S TCS(co,si,pe) > 0
si in Sink
*
FOR co in Conversion FOR re in Rawenergy
  - S tc[co;re;pe]X(co,pe) +S TSC(so,co,re) > 0
pe in Processedenergy, so in Source
*
FOR si in Sink FOR re in Rawenergy
  S TSS(so,si,re) > dsr[si;re]
so in Source
*
FOR si in Sink FOR pe in Processedenergy
  S TCS(co,si,pe) > dsp[si;pe]
co in Conversion
```

Figure 4-10: Formulation Statements for APL Tableau generator

Representation 8: MPS Format

Figure 4-11 shows (parts of) the matrix deck that was input to LINDO for a test of the energy model. In this case the row labels are simply numbers and the column names are the variable names qualified by numeric indices. The 'Rows' section defines the type of constraint ('L' for less-than-or-equal, 'G' for greater-than-or-equal, etc.). The 'Column' section lists the row names and non-zero values for each variable and the 'RHS' section does the same for the RHS coefficients.

```
NAME    ENERGY.APL   8/29/1986   19:29:35
ROWS
  N    1
  L    2
  L    3

  .    .
  G   15
COLUMNS
    TCS111        1             1.00
    TCS111        8            -1.00
    TCS112        1             1.00
    TCS112        9            -1.00

     .            .              .
    X22          17            -0.30
RHS
    RHS           2           100.00
    RHS           3           100.00

     .            .              .
    RHS          20            13.20
ENDATA
```

Figure 4-11: MPS Format Problem Statement for Solver

4.3. Model Mapping Approach

A useful facility in any modeling environment is to be able to combine previously defined and tested models into a larger system. In LPFORM, the component models may be either standard LP models (such as Transportation, Blending, Product-mix, Process-selection) or standard constraint types (such as inventory, supply availability, demand requirement, and so on). These are prestored in the system. In addition, users can define their own models and store them as model templates in the system's model bank. Thus, to build a complete refinery model, one might start by building and test-

ing standard models for the crackers and converters. Then these could be retrieved later for use in the complete refinery model.

Referring to the graphic representation in Figure 4-1, it seems intuitively obvious that the model consists of three separate transportation problems (one for each of the arcs in the top level diagram) and a product-mix problem for the conversions. This approach is implememented in LPFORM using the *CALL-MODEL* command. In the user interface this results in the screen shown in Figure 4-12 which shows part of the interaction when the user is defining the transportation model between the sources and conversions. The user is asked to 'map' the names used in the template model (for index sets, variable names and data coefficients) to those to be used in the new model. The system is able to automatically link the models based only on this information.

The user then selects this screen another three times for the other two transportation models and once for the product-mix problem. Finally, the *DEF-SETS* and *OPT* commands are used, as in the first-principles approach, to complete the specification of the model.

The LPSPEC statements corresponding to the model-mapping approach are shown in Figure 4-13. The names listed on the left in the figure were stored with the template; those on the right were supplied by the builder of the energy model. The template names are supposed to be meaningful to the user but are simply place-holders to LPFORM. There may be no need for some of the indices in the template. In this case the template model will be converted to a simpler model with fewer indices if the user simply types '#' instead of supplying a name for the index. Conversely, the user can add complexity to a model by using the replicate command (see Appendix).

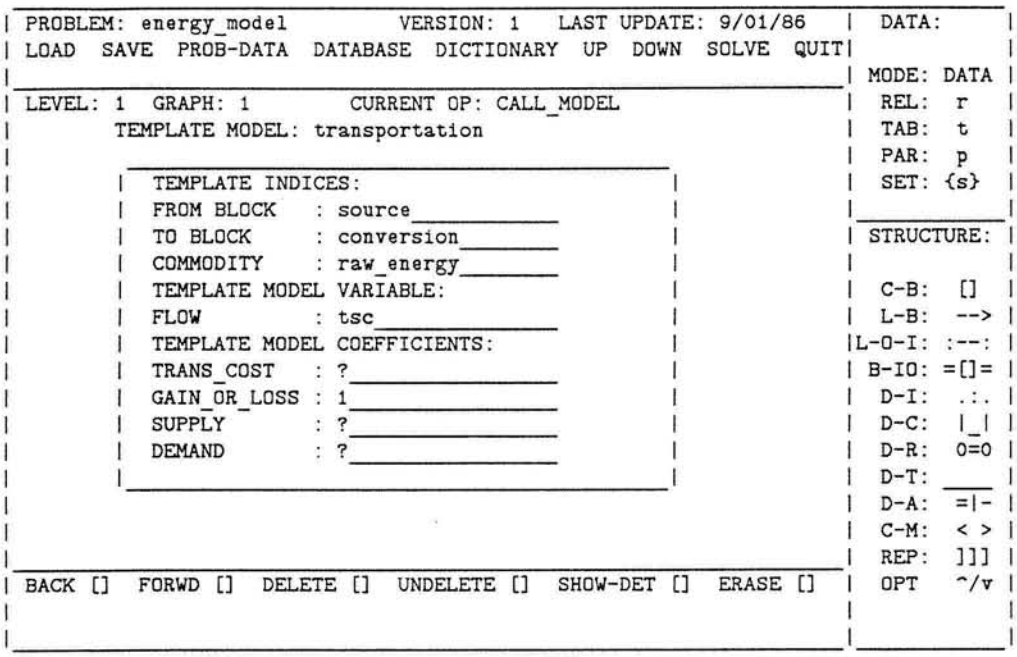

Figure 4-12: Mapping a Transportation Model on to the Sources - Conversions Link

```
call_model(transportation,                          energy_model,
           [from_block,to_block,commodity],         [source,conversion,raw_energy],
           [flow],                                  [tsc],
           [trans_cost,gain_or_loss,supply,demand], [?,1,?,?]).

call_model(transportation,                          energy_model,
           [from_block,to_block,commodity],         [source,sink,raw_energy],
           [flow],                                  [tss],
           [trans_cost,gain_or_loss,supply,demand], [?,1,?,?]).

call_model(transportation,                          energy_model,
           [from_block,to_block,commodity],         [conversion,sink,processed_energy],
           [flow],                                  [tcs],
           [trans_cost,gain_or_loss,supply,demand], [?,1,?,?]).

call_model(product_mix,                        energy_model,
           [block,input,output],               [conversion,raw_energy,processed_energy],
           [volume],                           [x],
           [profit,tech_coef,available_input], [conversion_cost,tech_coef,?]).

def_set(source,[foreign,domestic]).
def_set(raw_energy,[oil,gas,coal]).
def_set(conversion,[Refineries,electric_utilities]).
def_set(processed_energy,[gasoline,electricity]).
def_set(sink,[residential,transportation,industrial]).

optimize(min,energy_model,cost,symbolic).
```

Figure 4-13: LPSPEC Statements for Model-Mapping Approach

Before leaving this section, we make several observations about the interface

design. First, users will be able to move back and forth between a broad overview of their problem and minute details. They can specify pieces of the final system randomly as they come to mind rather than being forced to adhere to any strict order. A large portion of the intelligence underlying LPFORM involves knowledge of how bits and pieces of LP problems can be synthesized to a complete, consistent problem statement. Secondly, the user and the system will jointly define the system via a two-way flow of information. The system will do what it can to build the problem statement but will be in constant need of information and confirmation from the user.

5. Interface Design Strategy

Simon, [21], describes decision-making in terms of three stages of problem solving:

(1) Intelligence
(2) Design
(3) Choice

Most research has been directed towards the choice stage. In fact, the purpose of linear programming itself is to help users make optimal choices. Little research and few methods are available that can help in discovering a problem (intelligence) or in formulating models (design). This system described in this paper is directed towards the latter problem.

The four major elements in the design strategy are:

(1) A graphical, non-mathematical representation for LPs.

(2) Intelligent support for a number of different problem-solving strategies that reduce the complexity of the formulation process.

(3) Added functionality.

(4) The physical characteristics of the interface (screen design, iconic representation, etc.)

The following paragraphs describe the guidelines used to design the system. Some of them are fairly routine, others are based more on conjecture than on solid evidence. Where this is the case, the arguments for and against are summarized briefly in 5.1, 5.2, and 5.3.

5.1. Problem Representation

1. A "Natural" representation

Much of the expertise used by humans in formulating mathematical models is concerned with a translation between "real-world" objects and relationships and mathematical objects and relationships. This involves both semantic and syntactic knowledge that takes many years to acquire. The first task in designing a system to help formulate LPs was to provide a vocabulary (high level interface) to describe the real world, which is either (1) the same as that used by experts and novices or (2) easily learnt. Given a problem statement involving "real" objects and relationships, the second task is to use whatever internal representations are most convenient to translate to a valid mathematical problem statement.

The "natural" representation illustrated in Section 4, uses graphics to depict the time and space dimensions of the problem and icons to represent activities, inventories and other concepts. Although much expertise is incorporated in the system, it does not emulate the thought processes currently used by human experts. Thus, we do not follow the sequence of steps advocated by many textbooks (e.g. identify essentially mathematical objects such as decision variables, constraints and objective functions). In fact, the

users may think in real terms and state their problem in a manner closer to that used by managers than by mathematicians.

2. Algebraic Representation

The algebraic form (see Figure 4-10 has many advantages in terms of conciseness and generality as described in the previous section. Moreover, it is believed that experienced users will the algebraic form of their models in order to verify their formulation. While there are many proponents of algebraic formulation languages, the most widely used large-scale LP generators (e.g. OMNI [17]) take a matrix or "tableau" view of the problem. As a result, most experienced LP modelers think in tableau terms and only use the algebraic formulation in the initial development of the model and as a means of documentation. To some extent, these users will need to reorient themselves to the proposed system.

3. Mutiple Problem Representations.

The graphic and algebraic representations have already been discussed. The system will also provide an output that represents the tableau structure and another that is column (activity) oriented. Both of these are in symbolic rather than in numeric form. Experienced users should be able to use these representations to validate their models. Finally, the data dictionary (Figure 4-9) provides a significant short term memory aid and documentation for other users of the system.

5.2. Reducing Cognitive Complexity

The complexity of large-scale mathematical programming is a major economic and psychological deterrent to its effective use. The following ideas are being incorporated in LPFORM to reduce this problem.

1. **Hierarchical, Top-down, Formulation Process.**

   As illustrated in Figure 4-3, LPFORM allows networks to be specified in layers of increasing detail. The use of hierarchical structures is known to help human problem-solving. However, the basis for the structuring must be clear to the user. The particular form of hierarchical decomposition that is being incorporated in LPFORM will need to be tested.

2. **Different Approaches to Specifying Problems.**

   There are many ways of specifying the same problem in LPFORM. For example, a large problem can be specified in detail from "first principles" or built from many different "template" problems; data can be bound to the coefficients by direct input from the keyboard, by specifying prestored tables or by database queries. The objective is to allow users great freedom in the form of the inputs they need to specify. This richness has greatly complicated the design and increased the size of the system.

   The trade-off behaviorally, is between a system that constrains users to a fixed input representation and sequence of specification steps and one where individual styles and situations can be accommodated. The danger of the latter approach is that users will find it complicated and confusing. The challenge here is to build enough intelligence into the system to allow it to

correctly infer user intentions, and to be helpful in cases where it has insufficient evidence to proceed without more input from the user.

3. Piece-meal Approach to Problem Specification

A major design objective is to build a system that allows users to randomly specify small pieces of the problem. The intelligence within the system will be able to solve the resulting "jig-saw" puzzle by integrating the pieces into a single problem statement. This should relieve users of many tedious book-keeping chores. While the benefits seem obvious at first sight, this approach also raises some interesting issues. The first is technical. Can we build a system that can reliably infer correct formulations from partial specifications? The second is behavioral. Will this approach reduce user understanding of the system they are building and thereby eliminate one of the major benefits of the modeling process? On the other hand, it is conceivable that relief from tedious book-keeping work will encourage users to experiment with alternative specifications and thereby improve their understanding.

5.3. Added Functionality

A necessary condition for adoption of any computerized system is that it provides desired functionality and/or allows increased efficiency in performing necessary processes. The design principles discussed above influence the efficiency and effectiveness of LPFORM. Those mentioned in this section do this more directly by providing functions not present in other systems.

1. Database

A database allows direct access to application data and, (perhaps more im-

portant to the formulation process), to meta knowledge about the data. Users can learn about the data available in the system and specify database queries as part of the process of formulating problems. LPFORM itself, can use knowledge of the structure of the database at many points in the inference process.

2. **Consistency Checking**

The current practice is to first generate and test a trial tableau and then analyze it if there are problems. The analysis involves algorithms that discover, ex-post, the network structure underlying the matrix representation. LPFORM allows a completely different approach since the tableau matrix is constructed from the underlying network and not vice versa. Consistency checks will be performed at all stages as the LP is formulated. Checks on the generated tableau may still be necessary but fewer errors should be discovered and the total time to produce a correct formulation should be reduced.

3. **Symbolic formulation**

The first goal in LPFORM is to develop a problem statement in symbolic form. The second is to associate the symbols used in the formulation with actual data values. This strategy of separation will allow the model to be used with different sets of data and allows the size of the problem to vary freely.

5.4. Added Functionality

The user interface has been illustrated in the previous section. The Appendix contains a fairly complete specification of the main screen and brief definitions of all the commands.

Reference

1. *APEX II Reference Manual.* Control Data Corporation, 1974. Publication No. 59158100.

2. Astrahan, M. M., and Chamberlin, D. D. "Implementation of a Structured English Query Language". *Communications of the ACM 18,* 10 (October 1985), 580-588.

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

4. 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.

5. Creegan, Joseph B., Jr. Dataform, A Model Management System. , Ketron, Inc., Arlington, Virginia, November, 1985.

6. Crowder, Harlan. An APL-MPSX/370 Linear Programming Data Interface. RC9014(#394SO), IBM Thomas J. Watson Research Center, Yorktown Hieghts, New York, 1981.

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

8. Fourer, Robert and Michael J. Harrison. A Modern Approach to Computer Systems for Linear Programming. 988-78, Massachusetts Institute of Technology, Cambridge, Massachusetts, March, 1978.

9. Gershon, Mark E. L.P.-BLEND: Documentation and User Manual. Temple University, Philadelphia, PA, 1985.

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

12. Ma, P., F. H. Murphy and E. A. Stohr. A Representation Scheme for an Intelligent LP System. Center for Research in Information Systems, Graduate School of Business Administration, New York University, New York, 1986. Working Paper.

13. Ma, P., F. H. Murphy and E. A. Stohr. LPSPEC: A Language for Representing Linear Programs. Center for Research in Information Systems, Graduate School of Business Administration, New York University, New York, 1986. Working Paper.

14. Meeraus, A. *General Algebraic Modeling System (GAMS): User's Guide, Version 1.* Development Research Ceter, World Bank, 1984.

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

16. Murphy, F. H. and E. A. Stohr. "An Intelligent System for Formulating Linear Programs". *International Journal of Decision Support Systems 2*, 2 (March 1986).

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

18. Palmer, K. H., N. K. Boudwin, H. A. Patton, A. J. Rowland, J. D. Sammes, and D. M. Smith. *A Model-Management Framework for Mathematical Programming.* John Wiley & Sons, New York, 1984.

19. *PAM: A Practitioner's Approach to Modeling, Volume I - Primer.* Ketron, Inc., Arlington, Virginia, 1985.

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

21. Simon, Herbert A. "Rational Decision Making in Business Organizations". *AER* (September 1978), 493-513.

22. Slate, L. and K. Spielberg. "The Extended Control Language of MPSX/370 and Possible Applications". *IBM Systems Journal 17*, 1 (1978), 64-81.

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

I. Appendix: Design of The Graphics Interface

The primary graphics screen used by the system is shown in Figure I-1. This is the focus of the interaction with the users while they are defining their LP problems. Note that many other screens will be used and that a number of operations performed using the primary screen (such as data entry) will require access to subsidiary screens.

The preliminary design described here requires the user to point to a command and then to the object or area of the screen on which that command is to operate. Thus to establish a flow between demand and supply points the user will point to the *LINK_BLOCKS* command on the right of the screen and then to the two blocks that are to be connected.

Depending on the physical devices available, the pointing device could be a light-pen, a mouse or simply the cursor positioned by the arrow keys on the right-hand pad of an IBM PC (perhaps in combination with certain function keys). The ability to use the system by positioning the cursor using the keyboard will be a default to increase useability.

The screen is divided into four areas each having a distinct role in the interaction:

Graphics Section

The user constructs a graphical representation of the problem to be formulated in this area.

Top Border

The 3 lines at the top of the screen govern major actions that can be taken by the user. These are 'Main Menu Commands'. Examples include loading a different problem definition, saving the current one or attempting to have the LPFORM system

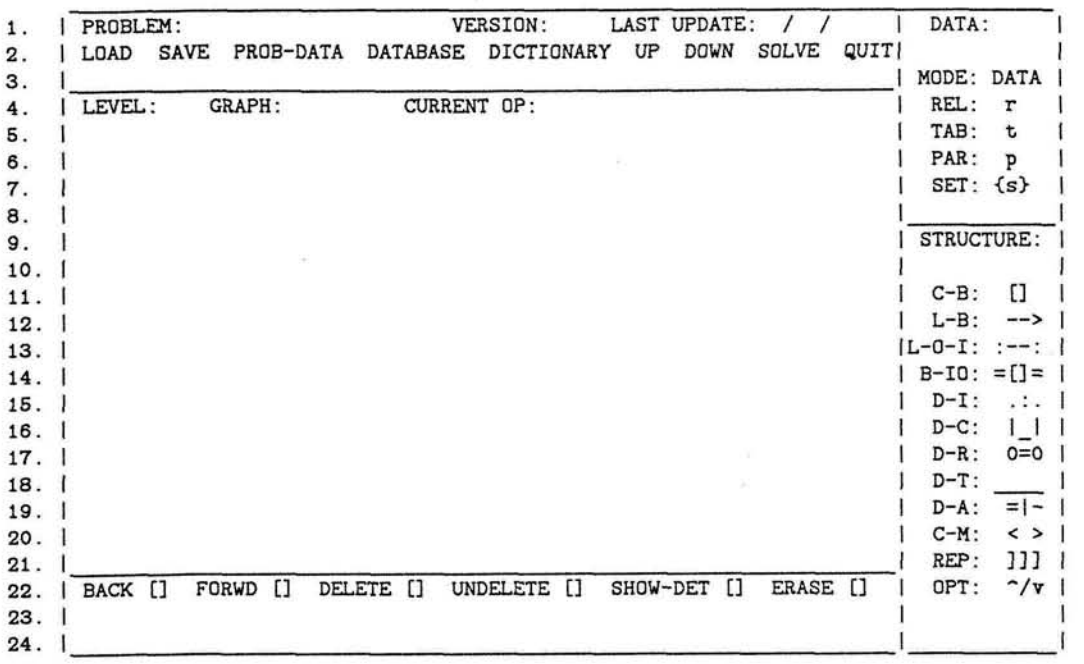

Figure I-1: Primary Graphics Screen

'solve' for the formulation of the LP. A number of other screens use this top border.

Bottom Border

These three lines are used to control the interactive graphics process. The prompts are 'Control Commands'. For example, using 'Back' allows the user to move backwards to previous problem states while 'Forwd' reverses this. The last two lines are used for system prompts and user responses respectively.

Right Border

This contains short nmemonics and icons associated with the 'Data Commands' that link the problem to its data and the 'Structure Commands' that are used to define the graph displayed in the center of the screen. Each command corresponds to a single statement in the LPSPEC language. The user points to one of these commands and then to a position in the center of the screen. A longer explanation of the command appears at the top of the graph screen to the right of Current Op:. The user is then led

through a series of questions associated with the command (these are displayed on line 23 and user responses are entered on line 24). Thus if D-T *(Def-Transported)* is chosen the user is prompted to choose an arc and a fill-in the-blanks screen is displayed to obtain information on the commodities transported, the arc capacities and any associated gains or losses.

If the cursor is used as the pointing device the following keys are used to 'jump' into another area:

Home: Moves up an area e.g. from the bottom border into the graphics section of the screen or from the latter to the top-border.

End: Moves down an area (reverse of Home).

CTL ->: Moves from the graphics screen to the top command in the right border.

CTL <-: Moves from the right border to last position occupied in the graphics section of the screen (or to the top left corner).

We now describe the use of these four areas in more detail.

I.1. Top Border

Line 1: Identifies the problem statement being defined or updated by the user.

Line 2: Contains the menu choices described below. A choice is made by positioning the pointer on an item and pressing "enter" or by simply typing the 1st letter of the desired choice.

Line 3: This displays a simple explanation of the choice pointed to on line 2.

Menu Choices:

Load: Load an existing problem definition. If this is chosen, the names of all previously defined problems are displayed on line 3.

Save: Save the current problem definition. The names of currently defined

problems appear on line 3 if the user decides to rename the problem.

Prob-data — Display all data associated with the current problem.

Database — Access the database management system.

Dictionary — Access the dictionary for the current problem and all other previously defined problems. The dictionary contains the complete documentation of each problem together with the final formulation.

Up — Move up to a higher level in the problem definition (after saving the information on the current screen). Thus from Level 3 Graph 2 the system would move to the parent level 2 screen.

Down — Move down a level in the problem definition. The first graph on the lower level is accessed. The user can step through the graphs on a level by typing Esc -> or Esc <-.

Solve — Requests the LPFORM system to attempt to deduce the correct problem statement. Correctly formulated problems may then be sent to the LP Solver.

Quit — Go to a higher level menu without saving the information currently on the screen (the user is given a chance to reverse this choice).

I.2. Right Border

The mnemonic plus the icon are used to help remind users of the available commands. The icon is actually moved to the appropriate place on the screen while the definition process is in progress. The F2 key can be used to toggle the display of icons in the graphics screen area. Displaying the icons helps the user to distinguish which parts of the problem have been defined to date.

Data Commands

The first part of the border deals with the data aspects of the problem.

Mode: — Can have values 'Data' or 'Symb'. The former means that data is to be bound to the symbols in the algebraic statement; the latter means that only a symbolic statement is to be generated.

Rel: Associate a relation in the DBMS with a symbol in the algebraic statement of the problem. A database query statement is used to define a table of coefficients.

Tab: Associate a data table with a symbol in the problem statement. Tables can be previously defined or interactively input by the user.

Par: Parameters are scalar values - since tables and sets are used for related values.

Set: Associate a set of objects with a symbol in the problem statement. If the mode is 'Symb' it is not necessary to explicitly define the members of a set. If the mode is 'Data' the elements of sets can be defined using database query statements, previously defined sets or interactively input by the user.

Structure Commands

These are used to define the structure of the LP problem. They are in 1:1 correspondence with statements in the LPSPEC language [12].

C-B: *Create-Blocks*
Create a block or a set of blocks on the screen. Small square indicates the position of each block; the user is prompted for the names of each block.

L-B: *Link-Blocks* Specify that a directed arc is to be drawn between two blocks on the screen. The user is prompted for specific properties of the arc - but these can be ommitted at top levels in the hierarchy.

B-IO: *Block-Inputs-and-Outputs*
A multi-commodity network may be specified by defining the inputs to and outputs from blocks. The user is prompted to indicate a block and to name the input and output sets associated with that block.

L-O-I: *Link-Outputs-to-Inputs*
Link all outputs of blocks to corresponding inputs - where the match is made on the basis of common commodity flows. Usually, the B-IO command will have been used prior to this command.

D-A: *Def-Activities*
Define a set of activities in a block. The user is prompted for the name of the set, for its class (e.g. production or blending) and for the names of the input and output sets. A sample screen was shown in Figure 4.

D-I: *Def-Inventories*

Specify that inventories are to be accounted for in a block. The user is prompted for the type of inventory (input, work-in-process or output) and for any special restrictions or bounds. This command is a specialized version of the Def-Activities command.

D-C: *Def-Commodities*

Specify that certain commodities are to be known by the system. Commodities are to distinguished from resources when they occur as inputs or ouputs of activities. The user is prompted for the corresponding set name.

D-R: *Def-Resources*

Specify that certain resources are to be known by the system. Resources are to be distinguished from commodities when thay occur as inputs or ouputs of activities (different constraint types are generated). The user is prompted for the corresponding set name.

D-T: *Def-Transported*

The user may explicitly define the properties of an arc (commodities transported, capacity limits, gains or losses etc.).

REP: *Replicate* Designated regions of the graph can be replicated if they have a repeating common structure. Usually, this command is applied to a single block which is replicated over space or time. In affect, this means that an additional index is added to every activity within a block and every flow to or from the block. The user is prompted for the set over which the replication is to take place. A ']]]' icon is affixed to the upper right-hand corner of a replicated block.

OPT: *Optimize* Used to specify the direction of the optimization (when this can't be inferred by the system). The user may also specify the objective function as an arithmetic expression in the variables.

Note that the two "LINK" commands create transportation activities while the *DEF-ACTIVITIES* and *DEF-INVENTORIES* commands create more general kinds of activities.

I.3. Bottom Border

The commands shown on line 22 are called 'Control' commands. These are used in conjunction with the 'Construct' commands in the right-hand border to give the user increased control over the interaction. For example, to enable him/her to back out of a sequence of 'construct' operations or to delete/revise previously defined parts of the problem specification. Only objects currently displayed on the screen are affected by these commands.

Line 23 displays prompts from the system for both the 'Construct' and 'Control' commands. For example, when the View-Detail command is being used the user is informed that pressing the ESC key will restore the Primary screen.

Line 24 is used to receive inputs from the user in response to the 'Construct' or 'Control' prompts that are displayed on line 23.

Control Commands

Back: Move back through a single (right-hand border) operation to the state before that operation was performed. Repeatedly using this command will eventually undefine and delete everything on the current screen.

Forwd: Reverses the affect of a 'Back' operation allowing the user to restore previous steps. Note that the user may back back to a certain problem state and then revise it before repeating a previous sequence of definitions automatically.

Delete: Used to delete an object on the screen - either permanently or (in conjunction with the 'Undelete' command to move it to another position).

Undelete: Reverses the action taken by the Delete command - the object is restored at the last cursor position indicated in the Graphics Section of the screen.

View-Detail: Used to see the detail behind an object on the screen. Those objects with icons attached (if icon display is set 'on' using the Esc-F2 key)

have much hidden detail. The Primary screen is stored 'as is' and a new detail screen is displayed. For an arc, the detailed specification of the associated flows etc. are displayed. For a block, a sequence of screens specifying the activities, inputs and outputs, etc. of the block are displayed. The Esc key will restore the Primary screen.

Erase: Allows the user (after a warning prompt from the system) to erase the entire specification shown on the current screen. Note that moving to the top border and using the quit command has the same affect except that the user is moved up to the calling menu.