LINEAR MODELING TOOLS

T. Hürlimann

Working Paper No. 187

Talk at the 11th European Congress on Operational Research
RWTH Aachen, July 16-19 1991.

July 1991

INSTITUTE FOR AUTOMATION AND OPERATIONS RESEARCH

University of Fribourg

CH-1700 Fribourg / Switzerland

Bitnet: HURLIMANN@CFRUNI51

phone: (41) 37 21 95 60 fax: 37 21 96 70

Linear Programming Modeling Tools

Tony Hürlimann, Dr. lic. rer. pol.

Key-words: Model Building, Expert Systems.

Abstract:
During the last four decades, Operations Research has concentrated considerable effort on designing techniques for solving linear and integer programming models. Now large and complex LP's can be solved. Another topic in modeling, surprisingly neglected, is model management: tools for model formulation, model analysis, model documentation, and result generating.
We describe a modeling language, named **LPL,** (Linear Programming Language), and other modeling tools which may be used to build, modify, and document mathematical models. The LPL language has been successfully applied to generate automatically MPS input files and solution reports of large linear and mixed integer programs. The available LPL compiler translates LPL models to the input code of any LP/MIP solver, calls the solver automatically, reads the solution back to its internal representation, and writes user defined reports in form of tables. The system can also be used simply to manipulate multi-dimensional tables, graphs or logical statements. LPL is still under development at our Institute.

1. INTRODUCTION

There is one main reason that linear programming is not more used in practice. Much work is needed to get the model set up and the data collected and compiled into the appropriate order. This is also true for integer, logical, and non-linear programming models, although for their models another reason is even more important: the solution might no be found in reasonable time. At least in the integer programming, important advances have been made to reformulate the model such that the model can be easier to be solved (Jeroslow 1989). We are not concerned by these advances here, but will concentrate on the first reason that model management itself has been neglected.

By model management, I mean model creation, model modification, model survey, model views, model documentation, model analyze, model result specification, etc There are reasons why model management has been neglected by the Operations Research community. First of all, most effort are still concentrated on solution techniques which have dramatically advanced since several years. Big LP models can now be solved in a fraction of time on a PC. Faster computer and sophisticated algorithms have been developed. On the other side, the business of modeling is a wide range of activities. Modeling is done by people with very different backgrounds in various contexts, and it is difficult to develop modeling tools which might be used by a wide range number of people. Most modeler still develop and use their own ad hoc tools to manage their models.

There are important disadvantages in doing so. Models are difficult to maintain with a changing crew. Model transparency may suffer and portability to different environments are limited. Often model or parts of it could also be used in a other context, but reusability is almost impossible. Operations Research journals are full of articles describing a special implementation of a model and its environment. The cost to develop its own tools should not be underestimated. That's the main reason why a decision makers tend to use rules of thumb rather than the troublesome way of model building.

It would certainly be of great use, if decision makers dispose of some 'universally usable' modeling tools and methods to do their job. I am convinced that such tools are not only needed, but they are also possible. It is even one of the greatest challenge facing the Operations Research to design and the construct computer based modeling environments. [Geoffrion 1988].

In this paper, I present briefly some 'main stream' tools such as matrix generators, spreadsheets, and database management systems, since they are widely used. I try to explain why these instruments are interesting but not sufficient and why we need a 'universally usable' modeling language which allow to represent a wide range of different models. There are mathematical models (linear, integer, or not-linear) which can be represented by a formal language close to the common mathematical notation. Then, I present a relatively modest modeling language LPL (linear programming language) based on such a notation.

The main advantage of such a language is that it is close to a common notation and known to many modeler in different domains. The expense to learn such a language is limited. Another benefit of this approach is that it is extensible. Non-linear models, logical models, and even fuzzy models [Zimmermann 1991] can be integrated by adding more features to the modeling language. A third and often overlooked advantage of a common modeling language for logical and mathematical models is that logical models such as knowledge bases can often be (automatically) translated to integer programs or vice versa to exploit efficient solvers and algorithms. It is also well known that different algebraic representations of the same integer program may behave very differently in computation. While one formulation might be intractable, the other might be easy to solve. So, while the modeler formulates his model, the modeling management system could translate the model to an easier form. It is obvious that a common modeling language facilitates these translations. A forth advantage is that a model - stored in the form of text stream (say language) - needs only a parser to translate the model into different *views* and can be easily ported to other environments. This could be compared with a page description language like PostScript which allows to store a page in a text stream which can be accepted by different environments.

View is one of the key terms in modeling: while the model may have a single (internal) presentation, it may have many views. Views is used here in the same manner as in the database terminology. The database scheme and their data are stored internally in a specified format. The user of the database may then define 'views' which show the data under specified angles. Data are selected, manipulated, and formatted to concentrate the user to limited aspects of the data. The same facilities are use in model management. Mostly, the modeler is only interested to partial aspects of his model at the same time and he wants to see them in different manner (graphically, as tables, mathematically, etc).

I'll close my talk by presenting some graphical tools which represent linear model structures, which will be developed soon at our Institute.
Our goal is to create a integrated multi-view architecture modeling tool which supports a wide range of user to create, manipulate, and describe their models. There is still a long way to go and our goals are ambitious.

MODELING TOOLS, WHAT FOR?

The linear programming model is in some sense very simple

maximize : c'x
subject to : Ax <= b

The model management, therefore, seems to be limited mainly to manipulate the three tables b, c, and A. So what's the point about model management? The point is, that the table A may be quite big and filled up with a great percentage of zero values. Typically for model, where the matrix A has more then 1000 rows and columns, the density (proportion of non-zeroes) is less then 1%. It would be a enormous waste of space to fill up the whole table with almost all of zeroes! But storage is not the main point. There exists sophisticated tools to manipulate sparse tables. A more serious problem is the view problem. It is difficult to keep the survey of such models. Furthermore, the matrix A is - in general - not available directly, but it must be computed from different smaller data tables available to the modeler. This is a major problem for bigger models. Another problem is that the modeler does not know in advance how big the matrix A is, that means he does not know the exact number of variables and constraints entering the model. This is only a result of the table manipulation just described. Saying that (linear) modeling is reduced to manipulate just the three tables b, c, and A, would be the same as saying that any database can be reduced to one big two-dimensional table. It is true, but might not be of great use to the database user. We need tools to create and manipulate model quickly and efficiently. What is needed are tools which

- support the modeler to create his model
- allow to represent the model in a unique scheme
- allow to view the model under different aspects
- allow to analyze and querying the model

I show the modeling process using a concrete problem. At the beginning, the modeler

has, in general, some vague ideas about his problem. Suppose the problem is “to build a certain number of teams for a soccer tournament“. There is already a big step in clarifying, how this problem is to formulate more precisely. A more precise formulation might be

“assign 180 (10-12 year-old) players to 15 teams such that the following condition are fulfilled

- a player must be only in one team
- total skill level of a team must be at least 59
- total player count per team must be 12
- total team age must be at least 124
- certain players must be in certain teams
- certain players are rejected from certain teams
- certain players must be together in the same team
- certain players must never be together in the same team“

This step is normally a “creative“ step, which means that it is difficult to formalize it. Nevertheless a model management system might already give some support. One may imagine that “semantical nets“ used in the AI terminology or some sort of “associative graphs“ (to use some rather vague term) may help the modeler. As far as I know, no tools have been developed to support the modeler in this “creative phase“. It seems that we are at the very beginning of this exciting research domain.

A second step is to translate the relatively precise linguistic formulation into a formalized form which then can be processed by the computer. This step is another “creative“ step. The result of this step might be the LPL model structure found in Appendix A. This step is more easy to do, but still difficult and time-consuming enough to support the user with (not yet developed) modeling tools. Of course, there is no obligation to enter the model directly in a modeling language such as LPL. Other - maybe graphical - tools can be more appropriate to create the model, which then translates to the modeling language formulation automatically. We are developing different graphical tools at our Institute, which allow to create models in a sometimes more convenient way.

The next step is then to enter the data, that is to instantiate the model to get a concrete model. Also this step is more easily done by some table manipulation software like database systems instead of entering the data directly into the modeling language as text stream. Data table software, such as database systems, can also do exhaustive tests on data consistency.

Hence, while I insist to have a modeling language which allows to define the entire model concisely, I do not think that model manipulation is reduced to manipulate the

modeling language formulation of the model.

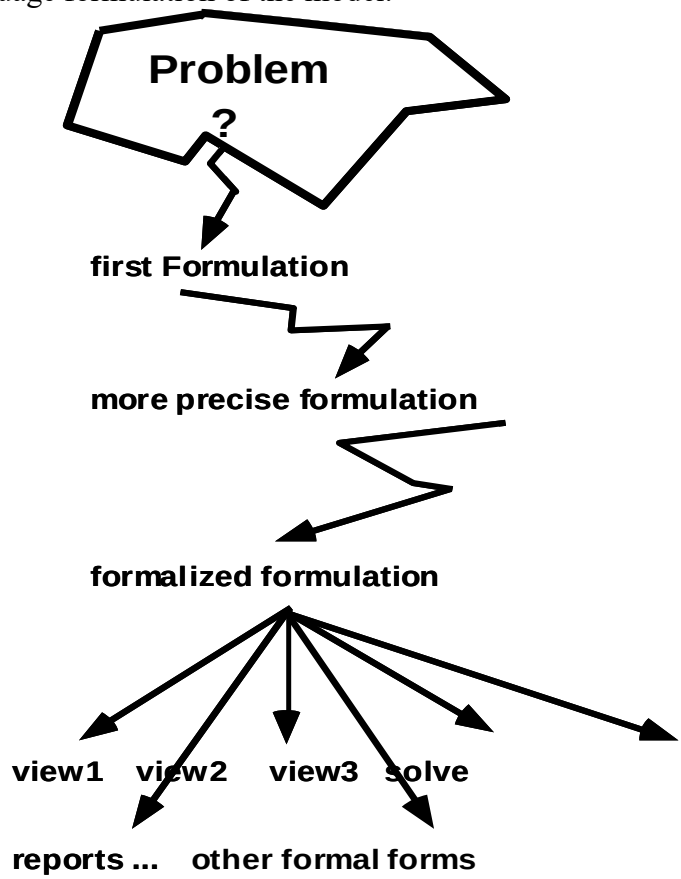

Figure 2

I describe now some approaches.

APPROACHES IN MODELING

Of course, any tool manipulating these tables for the user is welcome. A *matrix generator* is a code that fills up the matrix A. Often matrix generator is also called a code which produced a (FORTRAN)-which then fills up the matrix. Many matrix generator have been designed for model manipulation. But matrix generators have often been cumbersome to use and might introduce errors which were difficult to detect. To write a matrix generator program is more a job of a programmer than of a modeler. Nevertheless, matrix generators are still used and might be a excellent tool to model management. (see Fouurer [83] and Geoffrion [89] for detailed reports on this).

A more natural way to do model management actually - since they are in the main stream of commercial data processing - are *spreadsheets*. The matrix A, as well as the vector b and c or some subtables can be entered easily into spreadsheet tables. One of the main advantage is that spreadsheets not only allow to store numeric and string

constants but also formula. Any cell, or whole blocks of cell, can be specified by formulas. Model results produced by a solver can be written back to the spreadsheets and all kinds of pre- or post-calculations are possible. Although the operations with different spreadsheet tables at the same time can be quite complicated, spreadsheets are widely and increasingly used to manipulate models. An important disadvantage is that spreadsheets are not generic - one cannot build indexed objects. Each object must be entered explicitly into a cell. Tables may not be 'blown up' by simple instructions.

Another approach to model management is to use the now popular *relational databases*. Typical tasks in model management are subtable selection or table join or some other tables manipulations. Relational database systems are just designed to do that kind of operations very well. A important advantage of this approach is that the user (the modeler) must only design the structure of his tables (the scheme) by a database definition language and fill them up with the corresponding tables. Any manipulation can be done by a powerful database manipulation language. This approach is also very flexible and transparent: the table contents as well as the table structure may be modified any time.

A example drawn from Johnson [1989] is used to illustrate the concepts behind this approach. This is a simplified version of the model discussed in Mairs [1978]. "The problem is to decide how much of each product to produce at the plants, how to ship to warehouses, and tranship to demand-centers subject to the constraint that a warehouse has to ship all of the demand for all products to any demand-center that it ships to. In other words, each demand-center is assigned a single warehouse that must meet all of its demands for the several products." [Johnson 1989].
Figure 1 gives a rough graphical overview of the problem. The basic building blocks are the PLANTS, PRODUCTS, WAREHOUSES, and CENTERS. We take these four names for the *index-sets* used in the model. The modeler must now supply four data tables. The tables contain columns (called fields) and rows (called records) following the database terminology. Following Johnson, we define the four subsequent database tables:

PRODUCTION with the fields: PLANTS, PRODUCTS, CAPACITY, PCOST
SHIPPING with the fields: PLANTS, WAREHOUSES, SCOST
TRANSHIP with the fields: WAREHOUSES, CENTERS, TCOST
DEMAND with the fields: CENTERS, PRODUCTS, AMOUNT

The problem has three type of unknowns: how much to produce, how much to ship from a plant to a warehouse, and how to assign the center to its distributing

warehouse. The later can be simulated by a 0-1 variables, whereas the others are numeric unknowns.

To specify the variables, three further tables can be declared as:

PRODUCE with the fields: PLANTS, PRODUCTS, VALUE
SHIP with the fields: PLANTS, WAREHOUSES, PRODUCTS, VALUE
ASSIGN with the fields: WAREHOUSES, CENTERS, VALUE

where VALUE is the corresponding unknown amount of the variables. Any solver will have to fill these fields with the solution.

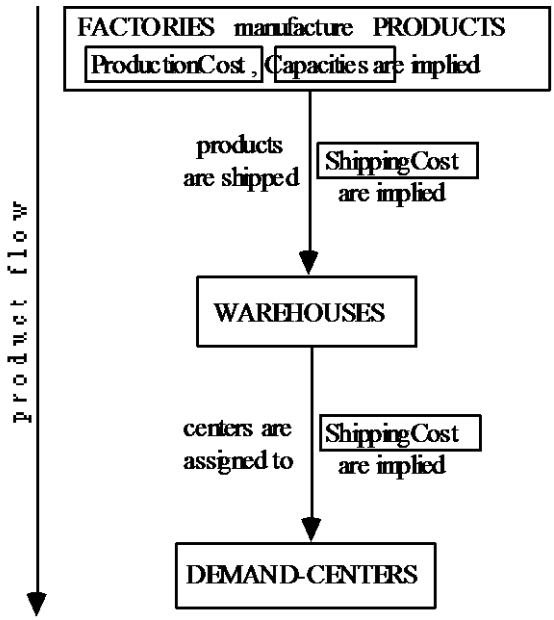

Figure 1

Similarly, three type of constraints may be defined represented as tables

PRODROW with the fields: PLANTS, PRODUCTS, VALUE
SHIPROW with the fields: WAREHOUSES, PRODUCTS, VALUE
CENTROW with the fields: CENTERS, VALUE

Semantically, some tables are linked together. The tables PRODUCE, p.e., is a subtable of the table PRODUCTION. PRODUCE can be produced from PRODUCTION by a *selection* operation. The same is true for the ASSIGN tables which is a subtable of TRANSHIP. On the other hand, some other tables can be

produced by a *join* operation from other tables. SHIP, p.e., is a join of the two tables PRODUCTION and SHIPPING. Figure 4 shows all tables which are produced from other tables.

An instantiation of the four data tables of our example [Johnson 1989] are shown in Figure 2.

PRODUCTION

PLANTS	PRODUCTS	CAPACITY	PCOST
topeka	chips	200	230
topeka	nachos	800	280
ny	chips	600	255

SHIPPING

PLANTS	WAREHOUSES	SCOST
topeka	topeka	1
topeka	ny	45
ny	ny	2
ny	topeka	45

TRANSHIP

WAREHOUSES	CENTERS	TCOST
topeka	east	60
topeka	south	30
topeka	west	40
ny	east	10
ny	south	30
ny	west	80

DEMAND

CENTERS	PRODUCTS	AMOUNT
east	chips	200
east	nachos	50
south	chips	250
south	nachos	180
west	chips	150
west	nachos	300

Figure 2

The six other tables can be produced easily by SQL commands shown in Figure 3. They will produces the six variable tables (three variables and three constraints) shown in Figure 4. Note that the field value is not filled. This is the solvers task!

```
SELECT plants, products, 0.0 AS VarValue
        INTO produce FROM production

SELECT plants, warehouses, products, 0.0 AS VarValue
        INTO ship FROM production, shipping
        WHERE production.plants = shipping.plants

SELECT warehouses, centers, 0 AS VarValue
        INTO ASSIGN FROM transhipp


SELECT plants, products, 0.0 AS ConValue
        INTO prodrow FROM production

SELECT warehouses, products, 0.0 AS ConValue
        INTO shiprow FROM shipping

SELECT centers , 0.0 AS ConValue
        INTO centrow FROM transship
```

Figure 3

To form the matrix A of the model, more table joins are needed. The matrix is formed into blocks where each block is itself a table produced by a SQL command. The blocks may be defined as shown in Figure 3a.

MATRIX (as MPSX column section)

BLOCKNAME	ROWNAME	COLNAME	VALUE
BLOCK11	PRODROW	PRODUCE	-1
BLOCK12	PRODROW	SHIP	1
BLOCK22	SHIPROW	SHIP	-1
BLOCK23	SHIPROW	ASSIGN	DEMAND.AMOUNT
BLOCKCAP	UPPER	PRODUCE	PRODUCTION.CAPACITY
BLOCKRHS3	CENTROW	UPPER,LOWER	1
BLOCKOBJ1	OBJECTIVE	PRODUCE	PRODUCTION.COST
BLOCKOBJ2	OBJECTIVE	SHIP	SHIPPING.COST
BLOCKOBJ3	OBJECTIVE	ASSIGN	∑(PRODUCT) DEMAND.AMOUNT * TRANSHIP.COST

Figure 3a

All nine tables in Figure 3a can be produces easily by further SQL commands. Appending these tables together produce the entire matrix table of our linear programming model. Another manipulation using the database interpreter will produce the needed format for the solver, the MPSX file. A method to do this (using the dBASE database language on a PC) has been described by Pasquier [1986].

PRODUCE

PLANTS	PRODUCTS
topeka	chips
topeka	nachos
ny	chips

SHIP

PLANTS	WAREHOUSES	PRODUCTS
topeka	topeka	chips
topeka	topeka	nachos
topeka	ny	chips
topeka	ny	nachos
ny	ny	chips
ny	topeka	chips

ASSIGN

WAREHOUSES	CENTERS
topeka	east
topeka	south
topeka	west
ny	east
ny	south
ny	west

PRODROW

PLANTS	PRODUCTS
topeka	chips
topeka	nachos
ny	chips

SHIPROW

WAREHOUSES	PRODUCTS
topeka	chips
topeka	nachos
ny	chips
ny	nachos

CENTROW

CENTERS
east
south
west

Figure 4

We summarize the database approach: All we need for this problem example, are four user defined data tables and some executable code written in a database manipulation language to produce the whole model in the needed form. This approach is very

flexible. Solution reports can be produced in the same manner. The powerful database language can manipulate the whole model in a convenient way. Of course, one needs to know a database language. But they are now very common and can be used in many application fields. Using this kind of approach for model management is very promising and the structured modeling language [Geoffrion 1987] is based on this approach, although structured modeling contains a explicit modeling language as well.

A MODEL LANGUAGE

The database approach might be so convincing that it is hard to see why modeler need still other, more specialized and appropriate tools for model management. There are many reasons. Non-linear or logical (rule based) models are more difficult to be manipulated by the database approach. Models are created and used by people with different cognitive skills. Many people, p.e., use the mathematical language to build their models. It would be natural for them to have a language close to the mathematical notation to create the model. LPL (linear Programming Language) has been originally designed for this purpose.

The last example could be defined in LPL as following. Note how close this tables are to the database tables. In a sense LPL is, like other modeling languages, quite close to a database manipulation language! Implemented as interpreter - instead of a compiler - it would be another data manipulation language.

```
SET   "the four index-set declaration"
  plants; products; warehouses; centers;

COEF  "the four data table declaration"
  CAPACITY, PCOST (plants,products);
  SCOST(plants,warehouses);
  TCOST(warehouses,centers);
  DEMAND(centers,products);

VAR
  PRODUCE(plants,products);
  SHIP(plants,warehouses,products);
  ASSIGN(warehouses,centers) BINARY;

EQUATION
  PRODROW(plants,products);
  SHIPROW(warehouses,products);
  CENTROW(centers);
```

There is a difference between this formulation and the database approach: The domains (the index-sets) must be explicitly defined. Giving these domains, each table has limited maximum size, which is the Cartesian Product of all its attributes.

In the formulation above, there is another difference: The VAR and EQUATION tables are not derived from the four COEF tables as in the database formulation. Therefore, “PRODUCE(plants,products)“ will declare the whole Cartesian Product of

all (plants, products) tuples. But we want only to have a subset of this entire table. LPL allows conveniently to do sub-domain declaration by adding a condition to the declaration as following

```
VAR
  PRODUCE(plants,products | PCOST);
  SHIP(plants,warehouses,products | PCOST and SCOST);
  ASSIGN(warehouses,centers | TCOST) BINARY;

EQUATION
  PRODROW(plants,products | PCOST);
  SHIPROW(warehouses,products | SCOST);
  CENTROW(centers);
```

Example: PRODUCE is declared for any (plants,products) tuple for which PCOST(plants,products) is different of zero. The number of generated variables depends of the data entries in PCOST. This is exactly what is needed.

Another possibility to produce sparse tables is through *compound sets*, which define the allowed tuples. We add the four compound sets PRODUCTION, SHIPPING, ASSIGNING, TRANSSHIPPING and get the new version in LPL

```
SET
  plants; products; warehouses; centers;  "simple index-sets"
  production(plants,products);            "compound index-sets"
  shipping(plants,warehouses,products);
  assigning(warehouses,centers);
  transshipping(warehouses,products);

COEF  "the four data table declaration"
  CAPACITY, PCOST (production);
  SCOST(shipping);
  TCOST(assigning);
  DEMAND(centers,products);

VAR
  PRODUCE(production);
  SHIP(shipping);
  ASSIGN(assigning) BINARY;

EQUATION
  PRODROW(production);
  SHIPROW(transshipping);
  CENTROW(centers);
```

To define the equations we enter the algebraic notation as following

```
EQUATION
  PRODROW : -PRODUCE + sum(warehouses) SHIP = 0
  SHIPROW : -sum(plants) SHIP + sum(centers) demand*ASSIGN = 0;
  CENTROW : sum(warehouses) ASSIGN = 1;
  minimize OBJECTIVE : sum(production) pcost*PRODUCE
        + sum(shipping) scost*SHIP
        + sum(assigning) tcost*(sum(product) demand)*ASSIGN;
```

The mathematical model is as following:

Four sets

$F = \{1 \ldots f_n\}$	the set of plants (factories)
$P = \{1 \ldots p_n\}$	the set of products

$W = \{1 \ldots w_n\}$ the set of warehouses

$C = \{1 \ldots c_n\}$ the set of demand-centers

With $f \in F, p \in P, w \in W, c \in C$, the parameters are:

cap_{fp} the capacity to be produced (in tons)

pc_{fp} the production costs (in \$)

sc_{fw} the shipping costs (in \$)

tc_{wc} the transshipping costs (in \$)

d_{cp} demands at the centers (in tons)

and the variables are:

X_{fp} the quantity produced (in tons) (only defined, if pc_{fp} is defined) (cannot exceed cap_{fp})

S_{fwp} the quantity to be shipped (in tons) (defined only if pc_{fp} and sc_{fwp} is defined)

A_{wc} = 1, if the center is assigned to the warehouse, otherwise = 0 (defined only if c_{wc} is defined),

The constraints are as following:

All produced quantities must be shipped to warehouses:

$$X_{fp} = \sum_{w \in W} S_{fwp} \quad \text{for all} f \in F, p \in P$$

The demands must be exactly satisfied (nothing must be stored):

$$\sum_{f \in F} S_{fwp} = \sum_{c \in C} d_{cp} \cdot A_{wc} \quad \text{for all } p \in P, w \in W$$

A center can be supplied only from one single warehouse:

$$\sum_{w \in W} A_{wp} = 1 \text{ for all } p \in P$$

Of course, the production capacities cannot be exceeded:

$$X_{fp} \leq cap_{fp} \quad \text{for all } f \in F, p \in P$$

The objective is to minimize overall costs, i.e. production, shipping, and transshipping costs:

$$\sum_{f \in F, p \in P} pc_{fp} X_{fp} + \sum_{f \in F, p \in P, w \in W} sc_{fwp} S_{fwp} + \sum_{c \in C} tc \cdot \left(\sum_{p \in P} d_{wp} \right) \cdot A_{cw}$$

An instantiation of the four data tables in a possible LPL format can be found in Figure 5. Other formats are possible. *The description above together with the data tables in Figure 5 defines the whole model formally* and can be processed directly by the LPL compiler. This is one of the main advantages of using a modeling language:

the whole model is in a unique, concise, and readable format - readable by the modeler *and* the machine- and can be processed from that form. Other advantages have already been mentioned. Appendix B shows a LP model, called EGYPT) with 145 constraints and 350 variables formulated in the LPL modeling language.

```
SET
  plants = / topeka ny /;
  products = / chips nachos /;
  warehouses = / topeka ny /;
  centers = / east south west /;

COEF
  CAPACITY, PCOST = /
    topeka   chips   200   230
    topeka   nachos  800   280
    ny       chips   600   255  /;

  SCOST = /
    topeka   topeka   1
    topeka   ny       45
    ny       ny       2
    ny       topeka   45  /;

  TCOST = /
    topeka   east    60
topeka       south   30
topeka       west    40
ny           east    10
ny           south   30
ny           west    80  /;

  DEMAND = /
    east   chips    200
    east   nachos    50
    south  chips    250
    south  nachos    80
    west   chips    150
    west   nachos   300  /;
```

Figure 5

GRAPHICAL TOOLS

LPN (Linear Programming Network) is a graphical representation of a LP model, which can be build automatically from a mathematical representation such as LPL. Each (indexed) variable is represented by a square and each (indexed) restriction as a circle. Squares can be connected to circles, but squares or circles cannot be connected to each other by lines. The resulting graph is a bipartite graph [Hürlimann 1987]. A circle is connected to a square, if and only if the variable shows up in the restriction. If the variables are not indexed, the graph corresponds to the fundamental graph of Greenberg [1981]. A similar graph was used by Egli [1980] to document the EP-model, a multi-period, agricultural model, which is a major tool for agricultural policy in Switzerland. LPN is a very convenient tool for moving within the model and for analyzing the model structure. Substructures such as netforms or flow structures can be revealed quickly. Interesting operations such as filtering and dis-aggregating

(unfolding the indexed objects!) can be implemented Appendix C shows the EGYPT model in an LPN representation, which has been produced automatically from the LPL representation.
Another graphical representation is the *Genus Graph (GG)* defined by Geoffrion [1987]. The GG shows a 'bottom-up' structure of the whole model and can be - like the LPN graph - produced automatically from a mathematical presentation . The same operations such as filtering and dis-aggregating are possible.

A *block structure* (see Appendix D for model EGYPT) is still another representation of the model structure. The whole model structure is exposed as a condensed matrix block. The PAM modeling environment is entirely based on this approach [KETRON 1986].

All three mentioned representations are tools which reveal different aspects of a *model structure*, independently of a specified *model instance*. The same tools may be used to show a model instance by dis-aggregating the whole model. By dis-aggregating, I mean to unfold the model using some or all index-sets. The corresponding representation, if all index-sets are unfolded, are the *fundamental graph* (defined by Greenberg [1981]), the *element graph* (defined by Geoffrion [1987]), and the *matrix picture*, which shows the whole matrix on textual or on bitmap level (see Appendix D for a bitmapped picture of the EGYPT model).
The power of these tools comes from the idea of (partially) unfolding. To 'visit' the model, one may use first the folded model, then unfold some specific model parts to see the interested details. 'Visiting' and querying the model is one aspect, modifying the other. All mentioned tools can also be used to edit the model. A research group at our Institute is developing a tool which allow this manipulation.So to be concrete: We create modeling tools which allow the translation LPL --> LPN, but also the translation LPN --> LPL. The modeler may edit the model in LPL form using a word processor or in LPN using a graphical editor or some other forms and the translation to the other forms is done automatically by the - yet to implement - model manager.

APPENDIX A

The possible model structure of the soccer model in LPL is as following

```
SET
  t;                    "teams"
  p;                    "players"
  Must_be_in(p,t);      "player p must be in team t"
  rejected_from(p,t);   "player p must not be in team t"
  together_groups(p,p); "players which must be together"
  never_groups(p,p);    "players which must never be together"

COEF
  Skill(p);       "Skill of player p"
  Age(p);         "Age of player p"

VAR
  Work(p,t)  BINARY;  "is 1, if player p is in team t, else 0"

EQUATION
  MAXIMIZE obj:             "maximize the assignment"
  Bounds(p)                 "player p must be only in one team"
  Skill_level(t);           "total skill level of team t must be at least 59"
  Heads(t);                 "total player count per team t must be 12"
  Team_age(t);              "total team age must be at least 124"
  Must(Must_be_in);         "player p must be in team t"
  Reject(rejected_from);    "player p is rejected from team t"
  Same(t,together_groups);  "players which must be in the same team"
  Never(t,never_groups);    "players which must not be in the same team"
END
```

The formulation of the constraints may be added on a latter time and are as following

```
MODEL
  obj: SUM(p,t) Work;
  Bounds(p): SUM(t) Work = 1;
  Skill_level(t): SUM(p) Work * Skill >= 59;
  Heads(t): SUM(p) Work = 12;
  Team_age(t): SUM(p) Work * Age >= 124;
  Must(Must_be_in): Work = 1;
  Reject(rejected_from): Work = 0;
  Same(t,together_groups[i,j]): Work[i,t] - Work[j,t] = 0;
  Never(t,never_groups[i,j]): SUM(j) Work[j,t] <= 1;
```

The next step is to define the data and to enter them to the model

```
{$R1}  "reset the random generator"
SET
  t  = / T1:T15 /;
  p  = / P1:P180 /;
  Must_be_in  =   /  P1 T2 ,  P2 T6 , P3 T7 /;
  rejected_from = / P10 T1 , P20 T2 , P1 T5 , P1 T6 , P1  T8 , P1 T9 , P5 T7 /;
  together_groups
    = /  P2   P3 , P112  P76 , P89   P9 , P34 P135 , P1 ( P35  P47  P81  P98 ) /;
  never_groups = /  P21  P22 , P55  P56 , P11 ( P35 , P45 , P56 , P89 , P21 ) /;
COEF  Skill(p) = trunc(rnd(3;8));   "between 3 and 8"
      Age(p)   = trunc(rnd(10;12)); "10 or 11"
{---------------------- end data ----------------------}
```

The whole soccer model is now formulated in LPL. Note that this model has totally 2700 (=180*15) single 0-1 variables. The model has been solved on a PC 80386 with a Coprocessor and 4 MB RAM with XA386 in 30 minutes.

APPENDIX B

LP-Model EGYPT: 145 rows, 350 columns, 1268 nonzeroes, matrix density: 2.5%

This is a one-period model of the Egyptian fertilizer industry developed at the World Bank (Choksi, Meeraus, and Stoutjesdijk 1980). Fertilizer is produced at different plants or imported, and consumed in the different regions of the country. The amount consumed in each region of all final products (the fertilizer) is given. The production uses several raw materials and intermediate products bought domestically or aboard. The total utilization by the processes cannot exceed a certain capacity at any plant. The production is also limited by the given input-output coefficients of the processes. Transport costs, distances, and the prices of the commodities (raw materials, intermediate products) are known. What is the level of processes at each plant; what is the shipped amount of all commodities between the plants and the regions; how much fertilizer must be imported, if we want to minimize the overall purchase and transportation costs?

This model is a relative simple model which combines a production, a transportation, and a consumption model of a whole industry. First, we give the entire formulation of the model in LPL form. A data set and model results are added.

The LPL representation of the model structure is as following (I omit the declaration of the sets and the data)

```
VAR
  Z(plant,process | p_pos);             "level of possible process at plant"
  Xf(c_final,plant,region | cp_pos);    "amount of final product shipped from plant to region"
  Xi(c_ship,i=plant,j=plant | cp_pos(,i) AND cc_pos(,j)); "amount of intermediate shipped between
plants"
  Vf(c_final,region,port);              "amount of the final product imported by a region from a
port"
  Vr(c_raw,plant | cc_pos AND p_imp>0); "amount of raw material imported use at a plant"
  U(c_raw,plant | cc_pos AND p_dom>0);  "amount of raw material at a plant purchased domestically"
  Psip;         "domestic recurrent cost"
  Psil;         "transport cost"
  Psii;         "import cost"

MODEL
  MINIMIZE Psi:  Psip + Psil + Psii;  "minimize the overall costs"
  mbd(nutr,region):                   "total nutrients supplied to a region for each final
product"
          sum(c_final) fn*(sum(port) Vf + sum(plant) Xf) >= sum(c_final) fn*cf75;
  mbdb(c_final,region | cf75>0):      "total of each final product supplied to a region"
          sum(port) Vf + sum(plant) Xf  >= cf75;
  mb(c=commod,pl=plant):              "production and consumption at a plant, plus the inter-plant
                                       shipments, plus imports and domestic purchases must exceed
                                       the total shipped for final products for each commodity"
          sum(process) io*Z + sum(p2=plant) Xi[c,pl,p2] + sum(p3=plant) Xi[c,p3,pl]
          + Vr + U  >=  sum(region) Xf;
  cc(plant,unit | m_pos):             "total utilization by all processes at each plant and for
each unit"
         sum(process) util*Z  <=  util_pct*icap;
  ap:  Psip = sum(c_raw,plant) p_dom*U;
  al:  Psil = sum(c_final,plant,region)  tran_final*Xf + sum(c_final,port,region)  tran_import*Vf
             + sum(c_ship,pl=plant,p2=plant) tran_inter*Xi[,pl,p2] + sum(c_raw,plant)
tran_raw*Vr;
  ai:  Psii/exch = sum(c_final,region,port)  p_imp*Vf + sum(c_raw,plant) p_imp*Vr;
```

APPENDIX C

Two aggregated LPN representation of the EGYPT model can be seen in the following graphs. The variables show up as squares and the restrictions as circles.

APPENDIX D

A simplified block representation of the EGYPT model.

	Z	Xf	Xi	Vf	Vr	U	Psip	Psil	Psii	RHS
mbd		1		fn						fn*cf75
mbdb		1		1						cf75
mb	io	-1	1	1		1				0
cc	util									util*icap
ap:						-p_dom	1			0
al		-tr	-tr	-tr	-tr			1		0
ai				p_imp	p_imp				1	0
OBJ							1	1	1	

The next figure is a (bitmapped) picture of the matrix of model EGYPT. Each non-zero within the matrix is shown as a black point.

REFERENCES

DOLK D.R. [1987], Relational Data Models and Relational Data Base Systems, NATO ASI on Mathematical Models for Decision Support, Springer.

EGLI G. [1980], Ein Multiperiodenmodell der linearen Optimierung für die schweizerische Ernährungsplannung in Krisenzeiten, Dissertation, University of Fribourg (Switzerland).

FOURER R., [1983], Modeling Languages Versus Matrix Generators for Linear Programming, in: ACM Transactions on Mathematical Software, Vol. 9, No. 2, June 1983, pp. 143-183.

GEOFFRION A.M. [1987], An Introduction to Structured Modeling, Management Science Vol.33, p.547-588.

GEOFFRION A.M. [1989], Computer-Based Modeling Environments, European Journal of Operational Research, 41(1989), pp. 33-45.

GREENBERG H.J. [1981], "Graph-Theoretic Foundations of Computer-Assisted Analysis", Greenberg H.J. and Maybee J.S., (eds.), Computer Assisted Analysis and Model Simplification, Academic Press, New York, pp. 481-495.

GREENBERG H.J., MURPHY F.H. [1991], Views of Mathematical Programming Models and Their Instances, Working Paper.

HÜRLIMANN T. [1987], LPL: A Structured Language For Modeling Linear Programs, Dissertation, Peter Lang Verlag, Bern.

HÜRLIMANN T. [1990], Reference Manual for the LPL Modeling Language, Version 3.5, Institute for Automation and Operations Research, Working Paper No. 175, June, Fribourg.

HÜRLIMANN T. [1990a], LPL: A Modeling System, Institute for Automation and Operations Research, Working Paper No. 177, June, Fribourg.

HÜRLIMANN T. [1990b], LPL: Three LP Model Examples, Institute for Automation and Operations Research, Working Paper No. 180, November, Fribourg.

JOHNSON E.L. [1989], Modeling and Strong Linear Programs for Mixed Integer Programming, in: Algorithms and Model Formulations in Mathematical Programming, ed. Stein W.W., NATO ASI Series F, Vol.51, Springer, p.1-43.

JEROSLOW R.G. [1989], Logic-based Decision Support, Mixed Integer Model Formulation, North-Holland, Amsterdam.

KETRON [1986], PAM, a Practitioner's Approach to Modeling, Part 1: Primer, Ketron Management Science, Inc, Arlington, Virginia, revised September 1986.

LENARD M. [1986], Representing Models as Data, J. Management Information Systems, Vol. 2, p.36-48.

MAIRS T.G., WAKEFIELD G.W., JOHNSON E.L., SPIELBERG K. [1978], On a Production Allocation and Distribution Problem, Management Science Vol. 24, No 15, p.1622-1630.

PASQUIER J., HÄTTENSCHWILER P., HÜRLIMANN T., SUDAN B. [1986], A

convenient technique for constructing your own MPSX generator using dBASEII, Angewandte Informatik Vol 7, p 295-300.

ZIMMERMANN H.J., [1991], Fuzzy Set Theory and its Applications, Second Edition, Kluwer Academic Publ., Boston.