COMPOSITION RULES FOR BUILDING LINEAR PROGRAMMING MODELS FROM COMPONENT MODELS

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

Edward A. Stohr
New York University
Leonard N. Stern School of Business
Department of Information Systems, 0266
44 West 4th Street, Room 9-170
New York, NY 10012-1126
(212) 998-0800

Pai-chun MA
College of Business Administration
University of Delaware
Newark, Delaware

Revised March 1990

Working Paper Series
STERN #IS-87-30

Abstract

This paper describes some rules for combining component models into complete linear programs. The objective is to lay the foundations for systems that give users flexibility in designing new models and reusing old ones, while at the same time, providing better documentation and better diagnostics than currently available. The results presented here rely on two different sets of properties of LP models: first, the syntactic relationships among indices that define the rows and columns of the LP, and second, the meanings attached to these indices. These two kinds of information allow us to build a complete algebraic statement of a model from a collection of components provided by the model builder.

1. INTRODUCTION

Recent hardware and algorithmic advances are providing order of magnitude improvements in the computer time needed to solve large linear programming (LP) models. Consequently, a smaller proportion of the costs of building and running models is being spent on the solution phase and a larger proportion on the building and interpretation phases. There is a corresponding need to develop more sophisticated model management techniques to aid in formulating, documenting, managing and interpreting LP's. At the same time, there have been enhancements in computer interfaces, allowing more options in the design of software systems.

The need for improved model building techniques and the development of new approaches to computer interfaces have led to a renewed interest in model building technologies. Example systems include PLANET (Breightman and Lucas, 1987), which is used for various planning problems in General Motors; GAMS (Brooke, Kendrick and Meeraus, 1988), AMPL (Fourer, Gay and Kernighan, 1988), LPL (Hurliman, 1989), and MODLER (Greenberg, 1989), which are algebraically oriented; PAM (Welsh,1987) and MATHPRO (Hirshfeld, 1988), which are block-structure/table oriented approaches, and an emerging class of graphically oriented systems such as LPFORM (Ma, Murphy and Stohr, 1989) and GIN (Sharda and Steiger, 1989).

Large linear programs almost always consist of a collection of linked small models. From the algorithmic perspective, discovering the "embedded" networks leads to faster techniques for solution. From the modeling perspective, looking at the model as a network containing non-network components allows us to break down the problem of formulation into separate, comprehensible pieces. In this paper, we describe a method for taking component parts and combining those parts into a complete LP matrix. The same method also allows us to combine different problems or subproblems together. For example, a modeler might separately develop a production model and a transportation model and at a later time want to combine them to form a more comprehensive model. The modeler need only look at one part of the problem at a time and then is given considerable assistance in combining the component models.

Traditional matrix generator languages such as OMNI and Dataform support the strategy of building models from more elementary components. In these systems, the model can be decomposed into a series of separately generated data tables which are then linked together to form the complete LP model by a program written in these languages. Matrix generation for the Project Independence Evaluation System (PIES), on which one of the authors worked, provides an extreme example of this approach in that it involved managerial divisions as well as data and software segmentation.

Each staff member who was responsible for a specific energy sector was assigned the task of generating condensations of activities in the form of coefficient tables for that sector. The sector specific tables were then combined into the complete LP by a matrix generator that also added transportation links. The person in charge of matrix generation had to coordinate the flow of information. Any change in the model, such as a change in the number of replications of a submodel, quickly turned into a major coordination problem. The communication burden was enormous, leading to serious personnel and model management problems (see Murphy, Conti, Sanders and Shaw, 1988).

What was needed in this system, and is needed in others, were methods for eliminating common errors in model component coordination and recognizing incompatibilities prior to the major expenditure incurred in matrix generation. To provide a perspective on our approach to this problem, we start with the steps in building an LP model (see Figure 1).

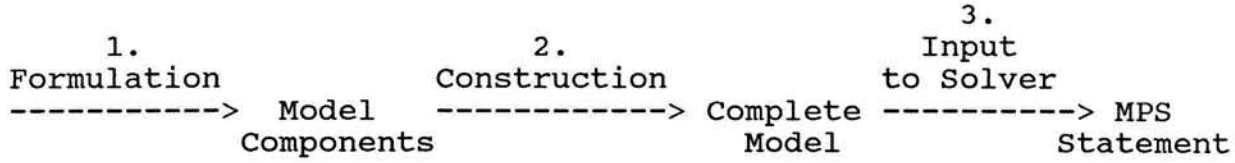

Figure 1
Steps in the Formulation of Input to a Solver

Step 1 can be performed interactively, while steps 2 and 3 are automated by the modeling system and may not be distinct in that the MPS statement can be the only statement of the complete model. The MPS input format is a standard followed by all major

LP solvers. Activities (columns) and constraints (rows) are given 8 letter names; nonzero data values at each row and column intersection are explicitly written as a long list of triples (row label, column label, coefficient value). The three classes of modeling systems mentioned above all produce an MPS file in step 3, but take different approaches with regard to steps 1 and 2. In table-driven systems, formulation focuses on the data tables containing the coefficient values; these are the Model Components which are then linked to form the complete model prior to the generation of the MPS file. In algebraic systems, the user writes the model in algebraic form; the Model Components are the algebraic equations plus the data tables. In graphic systems, the modeler conceives the problem as a network with embedded non-network components. The Model Components are then the graphic objects together with the data tables.

Note that algebraic systems such as GAMS, AMPL and MODLER provide a row perspective to the modeler in that the information is ordered by constraint, while the traditional matrix generators such as DATAFORM, GAMMA and OMNI use a column orientation. Each approach has advantages. The row orientation is consistent with the way Westerners read and facilitates constraint generation in integer programming. On the other hand, since columns typically have fewer elements than rows, practitioners often find that a column perspective facilitates model comprehension and that with large models it is easier to break the model into separate blocks

of activities that are later linked. Also, activities are more likely to change than rows (see Beale, 1968).

In LPFORM we combine graphic-, row- and column-oriented perspectives. In addition to drawing a graph of the major system components, one can define activities directly by their inputs and outputs (column perspective) or directly enter constraints in a standard algebraic notation (row perspective). The Model Components produced by step 1 in LPFORM consist of graphic objects, activity representations, "pieces" of algebraic statements (see below) and data tables. During the construction phase, the graphic and tabular information is used to produce a complete and consistent set of algebraic pieces. These are then combined to form a complete model in standard algebraic format.

We do not discuss the formulation step further in this paper. Rather, we concentrate on the construction step starting from the point at which the model components in step 1 have been generated as algebraic fragments. Our discussion, therefore, applies to other modeling systems as well as to LPFORM. Our objective is to provide flexibility with respect to the construction of complete models from their component models and submodels. At the same time, we want to ensure the consistency of the formulation and to provide meaningful diagnostics when mistakes occur.

2. INTRODUCTORY CONCEPTS AND OVERVIEW

To illustrate these concepts, we need the following definitions. A term is either the product of a coefficient and a variable or a right-hand-side (RHS) coefficient. An algebraic piece is a term on the left-hand side (LHS) of a constraint together with its associated summations (if any) or an RHS coefficient with its equality or inequality relation.

Our goal is to construct the complete algebraic statement of a model starting from a collection of either algebraic terms or pieces. For example, if we have the following three model pieces:

$$\sum_{j\epsilon J} c_j x_j, \quad \sum_{j\epsilon J} a_{ij} x_j, \text{ and } \leq b_i \quad i\epsilon I, \tag{1}$$

we know intuitively that they fit together to form the LP:

$$\begin{aligned} &\min \sum_{j\epsilon J} c_j x_j \\ &\text{subject to:} \\ &\sum_{j\epsilon J} a_{ij} x_j \leq b_i \quad i\epsilon I \qquad (2) \\ &x_j \geq 0 \quad j\epsilon J. \end{aligned}$$

More formally, we know that (2) is correct because there are no unsummed indices in the objective function and the only unsummed index in the constraint matches the index on the right-hand side.

We consider three possible characterizations of the initial pool

of terms and pieces:

(a) A complete collection of pieces (as in (1)).

(b) A complete collection of terms i.e. we know all the activities and coefficients plus the RHS pieces and need to infer the summations to construct the pieces.

(c) A complete collection of LHS pieces i.e. we need to infer the RHS pieces in order to complete the model.

In practice, we need to be able to handle all three cases and mixtures of them. We also need to be able to handle cases where the information is not complete (perhaps by formulating queries to elicit the required information). However, this is beyond the scope of this paper.

If we have all of the pieces (case a), together with some semantic information, there is a simple mechanical process to define the model. This is described in the next section. If we have the complete collection of terms (case b), we can still construct the complete algebraic model but we need more information. The added information describes the physical nature of the system being modeled. This is captured by the meaning of indices - whether they represent "form", "time" or "place" (i.e. "what", "when" and "where"). Section 4 discusses the generation of summation information from terms and some of the problems in dealing with form indices. Section 5 shows how semantic information (form, time and place) can be used to circumvent these problems. Section 6 describes a procedure for constructing models from terms. This procedure is illustrated in section 7.

In Section 8 we provide a brief outline of case c in which RHS coefficients are inferred from the existence of other information. We do not cover case c in detail for want of space; however many of the principles developed for case b apply also to case c. In the last section, we describe other work that has been done in this area and outline further avenues for research.

3. CONSTRUCTING A COMPLETE MODEL FROM ALGEBRAIC PIECES

Given that we are presenting procedures for building an LP from its components, it is important to provide a definition for the validity of the resulting model. We assume that all the coefficients and variables have been named consistently for the problem at hand. That is, if various components have originated in different models then their names have been "mapped" into those for the new model (see Ma, Murphy and Stohr, 1989). The definition of validity has to be limited in the sense that it cannot account for mistakes by the model builder. We use the following:

> Model Validity: A model is valid if the index set on each LHS piece matches the index set on its associated RHS piece and only one model can be formed from the terms or pieces that are given.

This is not a rigorous definition in that one could form multiple models by first building a model and then dropping off pieces. However, we presume that if a piece exists, it must be used, ruling out this possibility. Once we have introduced the

semantics of form, time and place in Section 5, we will be in a position to say something about whether a piece should exist or not.

If we have all the pieces for the model, the only task is to assemble them into the correct rows in the final algebraic statement. To do this, we first need to name the rows uniquely. Because the same constant value (say zero) can appear in many rows, we assume that the RHS pieces are distinguished by their indices and that constant values are stored along with their indices. To illustrate, the b in (2) is simply a placeholder for the coefficient, while the index i distinguishes the rows. We call the collection of indices on an RHS piece an index set.

Next, we need to determine the row to which each LHS piece should be assigned. We do this by determining the index set for the piece. We define an index set to be a set of index symbols. For example, the two indices i and j on the coefficient a in (2) constitute its index set {i,j}. We define the index set for a term to be the union of the index sets for the variable and coefficient.

The index set for a piece is computed from the index set for the term. The effect of a summation is to remove the summed index from the term index set. For example, in (2) the index j is removed from the index set {i,j} for $a_{ij}x_j$ to arrive at the index

set (i) for the piece, $\Sigma a_{ij}x_j$. If there is a partial sum, that is, an index is summed over a subset of elements in the set it indexes, then an index is added to the index set to denote the subset over which the sum is taken. Suppose, for example, in a multiperiod model the available capacity in period t is equal to the sum of all capacity acquired in period t and prior periods. That is, we have a piece:

$$\sum_{j \leq t} y_j \qquad (3)$$

In this case, the index set for the piece is found by dropping the index j and adding the index t that indicates the different subsets that are formed by the summations.

Sometimes an index is a function and not just a simple symbol. For example, suppose we can make an investment i in any year t that provides a return of r_{it+n} in period t+n. Then the cash flow constraint in period t+n includes a term $r_{it+n}x_{it}$. Thus, we have both the index t on the variable and the index function t+n on the coefficient. If we establish the convention that the index function on the coefficient determines the appropriate index for the piece index set, then we know that the appropriate index on the variable is the inverse of the function of the coefficient index. This is illustrated by the index on the coefficient r in (4) in the next section.

It is possible to define an index function implicitly. For

example, the cash flow coefficient could be defined as r_{itn} without the function t+n explicit. That is, a third subscript could be used to define the cash flows. We, therefore, need the following requirement:

> Index function requirement: All index functions must be stated explicitly.

Given our convention, this leads to the following rule:

> Index function rule: When an index that is not summed appears as the independent variable in an index function on a coefficient, use the function value to compute the element of the index set for the piece and treat the occurrences of different index functions on terms with the same variable as if they had different variables.

That is, the value of an index function replaces the index even if the domain index appears in the variable index set, eg. t+n appears in the index set for $r_{it+n}x_{it}$.

We assume that the model builder has placed enough indices on the variable or coefficient (including constants) of every LHS term so that the index set of every term contains the index set of every row with which the variable intersects. Note that indices are often dropped from coefficients to simplify data entry when the same values are used for different variables. The above assumption merely requires that a record be kept of all potential coefficient indices. We can then pattern match using

> Row Construction Rule: For a given RHS piece, select all LHS pieces whose row index sets match the index set of the RHS piece.

The validity of this rule can be seen immediately. Assume we have a piece that should not be used in a row where there is a match in index sets. Then this piece cannot be used elsewhere since there is only one RHS piece with a matching index set. By the uniqueness of the index sets on the RHS pieces, there is only one row for each piece and the model is unique. Because of the added information on coefficient indices, this rule is an extension of domain checking as implemented in GAMS.

Note that two constraints that are upper and lower limits on the same LHS do not cause a problem in the application of the row construction rule or the uniqueness of row names. We need only require uniqueness up to the point where the LHS is different.

The row construction rule breaks down if the pieces are not assigned sufficiently explicit index sets during the formulation process. For example, when representing resources that last more than one period such as labor or equipment, we often generate two rows in the model, one row to account for the availability of the resource and another for its utilization. Thus, the following is a typical formulation for a manpower planning problem:

h_t = number of employees hired at the beginning of period t
f_t = number of employees fired at the beginning of period t
e_t = number of employees available in period t
x_{jt} = amount of product j produced in period t
r_{t+1} = fraction of employees that return in period t+1 from period t (do not quit at the end of the period)
a_j = number of employees required to produce a unit of j

Availability:	$r_t e_{t-1} + h_t - f_t - e_t = 0$	(4)
Utilization:	$\sum_j a_j x_{jt} - e_t \leq 0.$	

The availability row determines the amount of resource available in each period and the utilization row describes how the available resource is used. Both rows have the same index, t. (If in the model there were other inputs besides labor in the production process, we would add a labor index indicating that these two rows measure labor at time t.)

Given just the 5 pieces listed above and attempting to apply the row construction rule, we would assign the two e_t terms to different rows, but would not know what to do with the other pieces without further information. In practice, and particularly when using traditional matrix generators, model builders make the rows uniquely identifiable by including an index that differentiates availability from utilization. If the applicable value of this index is included in each of the LHS and RHS pieces, the index function and row construction rules can be used to give the correct result. However, we need the following:

> Instance Rule: When a piece uses an instance of an index rather than the index, the instance of that index must appear in all pieces in the constraint.

Note that the above solution places a requirement that semantic information be included (eg. to differentiate instances from indices) in the algebraic terms generated from the formulation stage.

4. GENERATING PIECES FROM TERMS

We now consider case b from Section 2, where we have the terms for the problem and need to construct the algebraic statement by determining the appropriate summations for each LHS term as well as assigning it to the appropriate row. This is somewhat like the standard textbook modeling exercise in which students are presented with data (equivalent to our coefficients and their index sets) together with some descriptive text which helps determine the activities. The results presented here are an extension of the rules used in Welsh (1987) for combining tables to complete a model. In this section, the completeness requirement in the statement of case a in section 2 is relaxed slightly, because we are able to generate some pieces from a knowledge of other terms.

To generate a model from terms rather than pieces we must first construct the pieces. We begin by showing how LHS pieces for bound rows can be generated knowing only the RHS limits. Then we consider the more general problem.

In traditional matrix generators, variable names are simply concatenations of symbols. Thus, the x in x_j is not differentiated from the j by the matrix generator. Variable names simplify algebraic statements, however, because we

eliminate the need to indicate the relevant subset of variables on the right-hand-side of each equation. Of course, the semantic reason for differentiating variable names from indices is that variables represent actions and indices specify instances of these actions. Conversely, the rows of an LP usually describe what is being acted upon. However, the rows can also be used to place bounds on the actions. If we know that an RHS coefficient represents a bound on a variable, (and we know which variable is bounded) we can easily generate the LHS piece. This is the variable itself in the case of simple upper or lower bounds. More generally, we have the following:

> Bounds Construction Rule: Given that an RHS piece represents a bound row, form the LHS piece by summing all indices on the variable that are not in the index set of the RHS piece.

Again, the formulation phase has to generate the appropriate semantic information; in this case the variable name to be associated with the RHS bound piece.

We now turn to the more general case. A simple piece construction rule would be as follows:

> False piece construction rule: If the index set of a LHS term contains the index set of a RHS piece, sum over all of the indices on the term that are not on the RHS piece and place the piece so constructed in the row. Repeat for all terms.

We now provide two examples where it is not clear how to construct a piece from a term or other pieces using this rule in

the absence of further information. Suppose we have the terms $1_{ijk}x_{ijk}$ and $a_{ijk}x_{ijk}$ and three rows with indices i, j, k. Can we formulate a model from this information using the above rule? The answer is no. The rule would give us the following formulation:

$$\begin{aligned} &\sum_j \sum_k (a_{ijk}+1)x_{ijk} \le s_i \\ &\sum_i \sum_k (a_{ijk}+1)x_{ijk} \ge d_j \\ &\sum_i \sum_j (a_{ijk}+1)x_{ijk} \le c_k \end{aligned} \qquad (5)$$

But, suppose we are formulating a transportation model where we ship a product from i to j by transportation mode k, with a capacity of c_k for mode k, and each unit we ship takes a_{ijk} units of capacity. In this case the correct formulation is:

$$\begin{aligned} &\sum_j \sum_k x_{ijk} \le s_i \\ &\sum_i \sum_k x_{ijk} \ge d_j \\ &\sum_i \sum_j a_{ijk}x_{ijk} \le c_k. \end{aligned} \qquad (6)$$

One form of semantic information that can resolve ambiguities such as the above is units analysis, also known as dimensional analysis (see Bradley and Clemence, 1988, for an approach to including units and related information in a GAMS style modeling system). Each LHS coefficient has units of "something per unit of activity." In the multimodal transportation problem above, suppose the units on the first two rows are "tons" and the units on the third row are "vehicles." Using the following units rule,

we would have formulated this model correctly given all of the pieces.

> Units Rule: All terms that appear in a row have the same units as the row.

However, if our vehicle capacities were also measured in tons, our units analysis would not help us. Before resolving this issue, we present another problem.

In this example the index we wish to sum is on the coefficient and not the variable. Say we have the following pieces:

$$\sum_k b_{hik} x_k \qquad (7a)$$

$$\sum_k d_{hjk} x_k . \qquad (7b)$$

These fit into rows with index sets {h,i} and {h,j}. Suppose we have determined that there is another row with index set {h} in the problem but have no LHS terms. We might try to construct a piece:

$$\sum_k a_{hk} x_k . \qquad (8)$$

Our tentative piece construction rule fails again because we do not know whether to construct the new piece by summing over i in the (7a) piece or j in the (7b) piece, or, alternatively, whether we should ask the user to supply the coefficient name and values for (8).

In the next section we show how the problems raised by these examples can be resolved by further information that ascribes meaning to the symbols.

5. USING FORM, TIME AND PLACE INFORMATION

Linear programs represent physical actions on physical things that are described by three dimensions: what they are, where they are and when they are there. Analogous to the notion of state in dynamic programming, we can say that a row defines some state, that is, something, somewhere at some time. The right-hand side defines the starting level of the state and activities that intersect this row change the level of the state. Each state includes three generalized dimensions of form, time and place. We say "generalized" because we may use several indices to define a dimension, e.g. city and state to define location. For convenience we treat a generalized dimension as a single index in the remainder of the paper. To complete the definition of the state for a row, we add a generalized "attribute" index. This allows us to differentiate, for example, utilization from availability in (5) or to indicate input or product attributes such as octane in a blending model.

Activities can be described by the states (or a subset of states) changed. We add a dimension to an activity, which we refer to as a "mode" index since it indicates alternative ways of changing

the levels of states using inputs in different mixes.

Every LP constraint describes a restriction on something and is measured in units of that something. Thus, it has either an explicit or implicit index for form. Time and place indices enumerate occurrences of the form index. Rarely can we construct a meaningful constraint by summing terms over all values of the form index, and when we can, we wind up defining a new index for form. An example occurs in a feedmix problem when we sum over all of the different kinds of grain to get a more abstract entity (say) "animal feed" which we recognize as a new form index.

Form indices can also take on a dual role. They can not only indicate form but also instances of transformations. The simplest case occurs in the product mix problem (e.g. (2)) where j is not only a form index indicating products but also indicates different activities. The distinction seems academic but is crucial to a resolution of the problem in (6). To determine if a form index indicates form, note that each coefficient represents a rate of change of something. Consequently, there is some state whose level is changed by the term. We use this as the form index for the term. Thus, in (2), i becomes the form index in the row index set for the term $a_{ij}x_j$.

We are now in a position to formulate the multimodal transportation problem (6) correctly. First, we add an index, p,

that is the form index for product (or is an instance of the form index if there is only one product) to all terms and the first two RHS pieces. We know that the states for the first two constraints are different locations for products, the state for the third is vehicles and that p is the form index associated with $l_{ijkp}x_{ijkp}$ and k with $a_{ijkp}x_{ijkp}$. Then it is clear that $l_{ijk}x_{ijkp}$ should be associated with $\leq s_{ip}$ and $\geq d_{ip}$ and $a_{ijkp}x_{ijkp}$ with $\leq c_k$.

To resolve the problem (7) and (8), if h is the form index associated with $a_{hk}x_k$, then i and j enumerate instances and we can sum either b_{hik} over i or d_{hjk} over j to compute a_{hk}. Note that since the terms are summed in fixed proportion one new object is constructed for each activity, ie. an instance. We then place a symbol indicating the instance on each piece in (8), denoting what is formed from the sum of the inputs.

We can now define two rules for forming pieces from terms. In the first rule we only consider the case where the indices to be summed are on the variable. That is, we are avoiding the problem of constructing new terms that appears in (7) and (8).

> Piece Construction Rule 1: If the index set of a term contains the index set of a RHS piece and the form indices on the term and RHS piece are the same, then construct a piece from the term by summing the indices not in the index set of the RHS piece.

Rule 1 may generate pieces for use in other rules.

From our resolution of the issue associated with (7) and (8), we also have the following:

> Piece Construction Rule 2: If there is a form index on a term that is an element of a set denoting the constituents of an object, sum over the elements of this set and add the instance of an index that denotes the new object.

An example of an object is animal feed in the feed-mix constraint mentioned above (=Σgrains).

These two rules allow us to form new pieces from existing terms or pieces. The potential for working with pieces occurs if the model builder specifies pieces instead of terms from the beginning but does not identify all of their variations.

Piece construction rules 1 and 2 do not treat cases where partial sums are involved. Here we have an index (say t) in the index set of the RHS piece that is not in the index set of the term. We need to know that some index on the term should be summed over a subset indicated by the t index in the index set of the RHS piece. There are two equivalent ways of learning this information. The first is to input all partial sums directly. That is, if a term should appear in a partial sum, the piece with the partial summation indicated, should be generated during the formulation phase. The other way is to include semantic information with the piece that leads directly to the partial sum. An example of the latter would be information of the form

"capacity added in years t-n,...,t is available in year t." For algorithmic simplicity, we assume that all partial sums are given directly.

6. AN ALGORITHM FOR COMPOSING AN LP FROM ITS PIECES AND TERMS

Suppose we have the following information:

1. All LHS terms for regular (not bound) rows, all RHS pieces and all pieces with partial sums.

2. Knowledge on all indices as to whether they represent form, time, place, mode or added attribute, and knowledge of the row form index (if there is more than one form index on a term).

3. All abstractions that allow us to sum over form indices.

We also repeat the assumptions we made in Section 3:

1. The variables, terms and indices are named appropriately for the model being constructed.

2. The RHS pieces are uniquely identified by their index sets and instances of indices (up to upper and lower limits on the same constraint).

3. If pieces and terms exist, they should be used.

4. Each index set on a term either contains the index set of each row set it must intersect, or is an element of a subset whose index (or instance) is on a RHS piece.

5. All index functions are known.

The following algorithm constructs an LP where there is never more than one piece for a given variable and row combination and each piece generated belongs in the row to which it is added.

1. For each bound constraint, form from the associated variable

the LHS piece of the bound using the bounds construction rule as follows:

a. For each index on the variable that is not in the row index set and not involved in a partial sum, sum over the whole range of the index. For each instance of an index on the RHS piece, replace the index on the variable with its instance.

b. If the row index set contains indices that enumerate the subsets for partial sums, determine the associated subsets and sum the variable over these subsets. (Note that controlling indices for partial sums are not part of the variable index set, and, therefore can be recognized easily in the case of bounds.)

The remaining rows are constructed as follows:

2. For each RHS piece find all terms and pieces with partial sums whose index sets contain the RHS index set and where the instances of indices match.

a. Apply construction rules 1 and 2 in order.

b. Assign all matches of LHS pieces to the row associated with the RHS piece.

It is clear from the above discussion that only one piece is generated from each term or piece for each row/activity combination. If two terms with the same variable generate pieces for a single row, then the pieces have identical coefficient values. Note that the only way for an extraneous piece to be generated for a row is that there is a match in form, time and place, plus any attribute indices. Since the semantics are consistent, there must be a missing attribute on the constraint that would rule out one of the transformations, i.e there was an error made when the terms were constructed. Thus, we have met our restricted definition of model validity. More importantly, the model makes sense semantically. The form indices match and

all instances of terms with matching form indices are used. Any term, that, by its semantics, affects the level of some state is included in the row that defines that state.

Note that an LP formed by the above procedure is valid in the sense that all of the pieces are combined properly. If the modeler leaves out a term or piece that cannot be inferred from existing terms or pieces and the LP is fully connected, there is no way to identify the missing piece in general. Also, if a row index set is missing, this cannot be identified unless there is an unusable fragment. Domain knowledge concerning constraints that are often required can be used to suggest possible problems and model refinements to the user thereby addressing both of these limitations. However, this is outside the scope of this paper.

7. AN EXAMPLE ILLUSTRATING THE PROCEDURE

In this section, we construct a multiperiod production/ distribution model from indices, terms, RHS pieces, and one piece with a partial sum. For notational convenience, we use lower case letters to indicate indices and the corresponding upper case letter to indicate the set. When we use an upper case letter for an index, it indicates an instance of the index.

Indices:

j = product

j'= subset of products J'< J
m = material inputs
r = resource inputs (machinery and labor) R = {K,L}
t = time periods
i = time periods
f = factories
w = warehouses
v = vehicles
U = utilization (instance of an index)
A = availability (instance of an index)

Note that the form indices are j, j', m, r, and v.

Variables:

x_{jft} = produce
y_{Kfi} = acquire machine capacity in year i
e_{Lft} = employ labor
h_{Lft} = hire labor
d_{Lft} = dismiss labor
s_{jfwvt} = ship
i_{jwt} = inventory

Left-hand side terms (the form index is in () and the index set in {}):

$a_{Kjft}x_{jft}$ = use of machinery in production (K) {K,j,f,t}
$a_{LjftU}x_{jft}$ = use of labor in production (L) {L,j,f,t,U}
$a_{mjft}x_{jft}$ = use of materials in production (m) {m,j,f,t}
$-1_{LftU}e_{Lft}$ = labor utilization (L) {L,f,t,U}
$1_{LftA}e_{Lft}$ = labor availability (L) {L,f,t,A}
$-1_{LftA}e_{Lft-1}$ = labor availability (L) {L,f,t,A}
$-1_{LftA}h_{Lft}$ = labor hired (L) {L,f,t,A}
$1_{LftA}d_{Lft}$ = labor dismissed (L) {L,f,t,A}
$-1_{jft}x_{jft}$ = production amount (j) {j,f,t}
$1_{jfwvt}s_{jfwvt}$ = quantity shipped (j) {j,f,w,v,t}
$1_{jwt}i_{jwt}$ = inventory at end of period t (j) {j,w,t}
$-1_{jwt}i_{jwt-1}$ = inventory at end of period t-1 (j) {j,w,t}
$r_{jfwvt}s_{jfwvt}$ = capacity miles by vehicle type and trip (v) {j,f,v,w,t}

Partial sum piece:

$$\sum_{i \le t} y_{Kfi} \quad (K) \ \{K,f,t\}$$

Right-hand side pieces:

$\le E_{Kft}$ = existing capacity (K) {K,f,t}
$\le 0_{LftU}$ = labor utilization (L) {L,f,t,U}
$= 0_{LftA}$ = labor availability (L) {L,f,t,A}
$\le M_{mft}$ = material availability (m) {m,f,t}

$\leq C_{mt}$ = total material input limit (m) {m,t}
$= 0_{jft}$ = product balance (j) {j,f,t}
$\leq -D_{jwt}$ = demand (j) {j,w,t}
$\leq R_{tv}$ = ton mile capacity of vehicles (v) {t,v}
$\leq B_{xj'ft}$ = bound on products in subset J' by factory (x) {x,j',f,t}
$\leq B_{xft}$ = bound on total units of output in each factory (x) {f,t}

Constraints:

Using the first rhs piece the indices on one term (applying piece construction rule 1) and the partial-sum piece match (step 2b), we define a capacity constraint:

$$\sum_j a_{Kjft}x_{jft} - \sum_{i\leq t} y_{Kfi} \leq E_{Kft} \quad t\epsilon T$$

The next two constraints on labor have distinguished indices other than form, time or place indices (steps 2a and 2b):

$$\sum_j a_{LjftU}x_{jft} - 1_{LftU}e_{Lft} \leq 0_{LftU} \quad f\epsilon F \; t\epsilon T$$

$$-1_{LftA}e_{Lft-1} + 1_{LftA}e_{Lft} - 1_{LftA}h_{Lft} + 1_{LftA}d_{Lft} = 0_{LftA} \quad f\epsilon F \; t\epsilon T$$

We also have a constraint on the materials (step 2):

$$\sum_j a_{mjft}x_{jft} \leq M_{mft} \quad m\epsilon M \; f\epsilon F \; t\epsilon T$$

In the next constraint we use piece construction rule 1 and sum over all factories to limit the total use of each raw material:

$$\sum_{j,f} a_{mjft}x_{jft} \leq C_{mt} \quad m\epsilon M \; t\epsilon T$$

We have a material balance constraint linking factories and warehouses (step 2):

$$-1_{jft}x_{jft} \quad +\sum_{w,v} 1_{jfwvt}s_{jfwvt} = 0_{jft} \quad j\epsilon J \; f\epsilon F \; t\epsilon T$$

We next have a demand constraint where we use the index function rule on the inventory activity. Note that the index function rule leads to two pieces with the same variable in the constraint:

$$\sum_{f,v} -1_{jfwvt}s_{jfwvt} \; -1_{jwt}i_{jwt-1} \; +1_{jwt}i_{jwt} \; = -D_{jwt} \; j\epsilon J \; w\epsilon W \; t\epsilon T$$

Our last regular constraint is a limit on vehicle capacity:

$$\sum_{j,w,f} r_{jfwvt}s_{jfwvt} \leq R_{tv} \quad t\epsilon T \; v\epsilon V$$

The following two sets of bound rows are then constructed (steps 1b and 1a respectively):

$$\sum_{j\epsilon J'} x_{j'ft} \leq B_{xj'ft} \; j'\epsilon J' f\epsilon F \; t\epsilon T$$

$$\sum_{j} x_{jft} \leq B_{xft} \quad f\epsilon F \; t\epsilon T$$

In this example we did not use composition rule 2: the situations where this applies are relatively infrequent in practice. Note also that all terms that contained the RHS indices of an RHS piece were used in the corresponding rows.

8. GENERATING RHS PIECES

A variation on the above is inferring the existence of RHS pieces (case c from Section 2). This is especially important when we need to combine different submodels because it will often be the case that the output from one model will be the input to another.

If we start with LHS pieces, inferring the existence of a row is simple. The normal forms of constraints are material balances, and constraints where demand is restricted to be less than or equal to supply (or supply greater than equal to demand). However, there are, for example, policy constraints where one specifies a minimum supply reversing the normal inequality. Thus, the relation cannot be determined with certainty by an automated system. If, instead of starting with LHS pieces, we start with LHS terms and pieces with partial sums, we can infer the limited set of possibilities for RHS pieces by noting that the index sets on RHS pieces are subsets of term index sets.

The rows in an LP problem can be regarded as regulators of flows. Thus, every time there is an input to one activity that is an output from another activity, there must be a row in the LP to link the flows. When a flow is either just an input or output to the activities but not both, there is a corresponding nonzero RHS with an inequality or equality relation. When there are both inputs and outputs, one can have a material balance with a zero

on the RHS.

As discussed above, we cannot in general determine the direction of the inequality. However, in an important special case, we can add new rows (balance equations) to our model while at the same time guaranteeing that the signs of the pieces are correct. This occurs when we are combining two or more component models to form a larger model. In essence, the sign of the flow is determined by the semantics of the original models. As long as we can match the inputs to one model with the outputs from the other model, we can determine the links.

We do not have the space to cover the detailed mechanics involved in combining different models here. However, the following provides an overview of our approach. For simplicity, we assume that two models are to be combined and that the names of the objects in the two models are consistent. The combination of the models can be complicated by the fact that they may each contain representations of the same LP activities or use the same supplies and sources. The initial combined pool of pieces from which we build the model will therefore contain duplicate pieces. If we eliminate the duplicates, and further assume that the same resource (say money) occurs in the objective function of each model, the basic idea in combining two models is quite simple: one combines the rows from the two models where the index sets match and both rows have the same form indices. In order to

recognize all common flows, it is useful to include variable nonnegativity constraints explicitly in each model, treating them as terms from which one constructs pieces. An example where this is necessary occurs when a product-mix model is combined with a transportation model; the outflow of products is implicit in the former (captured by the non negativity condition) but explicit (as a supply constraint) in the latter. We need to combine the nonnegativity constraint from the product mix with the supply constraint of the transportation problem, keeping the signs consistent, to obtain a material balance equation. The objective functions of the two models can be simply combined with appropriate changes in the signs of the coefficients if one problem is a maximization and the other a minimization.

The current version of LPFORM has a primitive capability for combining models (Ma, Murphy and Stohr, 1989). We believe that this is an important model management feature that can be developed further. One of the major reasons behind our use of algebraic pieces as fundamental building blocks in the construction phase is to facilitate the reuse of models in this fashion.

9. CONCLUDING COMMENTS

In this paper, we have shown how a complete algebraic statement of an LP problem can be composed from its component algebraic

pieces. This capability is necessary in a system like LPFORM, which translates graphic components into algebraic fragments that then have to be linked together. In algebraic systems such as GAMS in which users input complete algebraic statements in the first place, the techniques developed in this paper can be used to compose new models from previously developed submodels.

The focus of this paper has been on the relatively narrow area of model construction (Figure 1). While this is central to our objective of building truly flexible systems, there are a wide range of modeling and model management issues that we have not covered. Within the construction area we have not discussed the representation of indices and sets in any detail (see Hurliman, 1989, and Geoffrion, 1989). Nor have we touched on the many current developments in other areas of model development and use. Some research directions that should be mentioned are as follows. Starting with the formulation stage, there is a growing literature on graphical approaches to modeling (Glover, Klingman and Phillips, 1990, Jones, 1990, Ma, Murphy and Stohr, 1989, and Sharda and Steiger, 1989), model reuse (Bhargava, Kimbrough and Krishnan, 1989), model building systems (Hurliman, 1989, Lucas and Mitra, 1988, Dhar and Jarke, 1989, and Greenberg, 1989), and structured modeling (Geoffrion, 1987). In the area of model interpretation and use, there has been work on the development of model management systems (Bhargava and Kimbrough, 1989) and on model diagnosis and analysis (Greenberg, 1989).

Our own research is aimed at providing automated assistance for the formulation process. Our goal is to expand the classes of semantic information utilized in the formulation process to include information about problem types and corresponding problem structures. We are attempting to form a semantic net of model types for use as component models. By knowing problem characteristics, we can let the system suggest the appropriate representation for a model component.

Even a brief survey such as the above reveals a broad range of research efforts that would not even have been contemplated five years ago. There is a continuing need to formalize our knowledge about LP models to enhance the capabilities of model management systems.

Acknowledgements: We wish to thank Harvey Greenberg for his helpful comments. This research was supported by Amoco and Shell Companies.

REFERENCES

Asthana, A., F.H. Murphy, and E.A. Stohr (1989), "Tests of an Interface for Formulating Linear Programs," presentation ORSA/TIMS New York, fall.

Beale, M. (1968), Mathematical Programming in Practice, Pittman.

Bhargava, H., S. O. Kimbrough (1990), "On Embedded Languages for Model Management," Proceedings of the 23rd Annual Hawaii International Conference on Systems Sciences.

Bhargava, H., S.O. Kimbrough, R. Krishnan (1989), "Unique Names

Violations: A Problem for Model Integration or You Say Tomato, I Say Tomahto," Working Paper, Wharton.

Bradley G.H. and R. D. Clemence (1988), "Model Integration with a Typed Executable Modeling Language," Proceedings of the Twenty-first Hawaii International Conference on Systems Sciences, Vol III.

Breightman,R.L. and J.M. Lucas (1987),"PLANETS: A Modeling System for Business Planning," Interfaces, 17,1 Jan-Feb.

Brooke, A.,D. Kendrick and A. Meeraus (1988), GAMS: A User's Guide, The Scientific Press, Redwood City, CA.

Dhar, V. and M. Jarke (1989), "On Modeling Systems," workshop proceedings, Information Systems and Decision Processes, Tucson, AZ, Oct. 5-7.

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

Geoffrion, A.M. (1987), "An Introduction to Structured Modeling," Management Science, 33,5, pp. 547-588.

Geoffrion, A.M. (1989), "Indexing in Modeling Languages for Mathematical Programming," working paper, Western Management Science Institute, November.

Glover, F., D. Klingman and N. Phillips (1990), "Netform Modeling and Applications," Interfaces, to appear.

Greenberg, H.G. (1989), "Overview of the Development of an Intelligent Mathematical Programming System (IMPS)," working paper University of Colorado at Denver, December.

Greenberg, H. G. (1987), "Computer-Assisted Analysis for Diagnosing Infeasible or Unbounded Linear Programs," Mathematical Programming Studies vol. 31, pp. 21-29.

Hirshfeld, D. (1988), "MATHPRO," Presentation at the Intelligent Mathematical Programming Systems Symposium, University of Colorado at Denver, December.

Hurliman, T. (1989), "Reference Manual for the LPL Modeling Language," working paper, Institute for Automation and Operations Research, University of Fribourg, Fribourg Switzerland, October.

Jones, C.V. (1990), "An Introduction to Graph-based Modeling Systems" ORSA Journal on Computing, to appear.

Krishnan, R., "PDM: A Knowledge-Based tool for Model Construction," Decision Support Systems, to appear.

Lee, J.S. (1989), "Towards Automatic Selection of Integer Programming Algorithms in a Model Management System," presentation ORSA/TIMS New York, fall.

Lucas, C. and G. Mitra (1988), "Computer Assisted Mathematical Programming Modeling System: CAMPS," Computer Journal.

Ma, P., F.H. Murphy and E. A. Stohr (1989), A Graphics Interface for Linear Programming," Communications of ACM, 32, 8 pp. 996-1012, August.

Murphy, F.H., J. Conti, R. Sanders and S. Shaw, "Modeling and Forecasting Energy Markets with the Intermediate Future Forecasting System," Operations Research, May-June.

Sharda, R. and D. Steiger (1989), "Functional Description of a Graph-based Interface for Network Modeling (GIN)," working paper, Oklahoma State University.

Welch, J.S., Jr. (1987) "PAM- A Practitioner's Approach to Modeling," Management Science, 33, 5, pp. 610-625, May.