韓國經營科學會誌
第25卷 第3號
2000年 9月

GPS와 인공신경망을 활용한 데이터베이스로부터의 선형계획모형 발견법

권오병* · 양진설**

Linear Programming Model Discovery from Databases Using GPS and Artificial Neural Networks

O-Byung Kwon* · Jin-Seol Yang**

Abstract

The linear programming model is a special form of useful knowledge that is embedded in a database. Since formulating models from scratch requires knowledge-intensive efforts, knowledge-based formulation support systems have been proposed in the Decision Support Systems area. However, they rely on the assumption that sufficient domain knowledge should already be captured as a specific knowledge representation form. Hence, the purpose of this paper is to propose a methodology that finds useful knowledge on building linear programming models from a database. The methodology consists of two parts. The first part is to find s first-cut model based on a data dictionary. To do so, we applied the General Problem Solver (GPS) algorithm. The second part is to discover a second-cut model by applying neural network technique. An illustrative example is described to show the feasibility of the proposed methodology.

1. Introduction

Linear programming model formulation from scratch requires knowledge-intensive efforts. The formulation efforts include selecting decision variables and other coefficients, and finding functional and linear dependencies among them. In order to acquire optimal solution, the formulation

* School of Management and Economics, Handong University
** Samsung Life Insurance Co. LTD.

results are often represented as a Structured Modeling schema, which is well known in the Decision Support System (DSS) research area [14]. The Structured Modeling schema is composed of the elements and their calling sequences. The element types are primitive and compound entities, attributes, variable attributes, functions, and tests. Hence, formulating Structured Modeling Language (SML) schema models from databases is equivalent to finding elements from databases and then enumerating them according to the SML syntax. Finding functions and tests are more difficult because they must be defined after discovering numerical and functional dependencies among other attributes. For example, in order to formulate a test element T:DEM such that "T : DEM (FLOW, DEM) /t/ ; @IANDj (@SUMi (FLOWij) = DEMj).", the numerical relationships between FLOW and DEM must be found, via data mining or knowledge discovery from databases(KDD) [17-20].

Data mining is the exploration and analysis, by automatic or semiautomatic means, of large quantities of data in order to discover meaningful patterns and rules[4]. Knowledge discovery, on the other hand, refers to the overall process of discovering useful knowledge from data. Frawley *et. al.* defined knowledge discovery as the non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data[13]. Standard statistical method, market basket analysis, memory-based reasoning, genetic algorithms, cluster detection, link analysis, decision trees, and artificial neural networks have been popularly used for data mining. Among them, artificial neural networks approach to data mining or knowledge discovery is well suited for estimation and prediction knowledge from databases.

Finding complex relationships among database fields remains as a research challenge [12]. Since linear programming models are applicable to this case, depending on conventional knowledge discovery techniques is not efficient. For example, a linear programming model consists of a set of attributes and decision variables that are grouped by a corresponding index function. The attributes are also inter-related to one another in the context of forming modules as complex relationships, including objective functions and constraints Formulating linear programming models is deeply related to data sets and corresponding data dictionary. Hence, it is necessary to develop algorithms that effectively utilize such information. These give our motivation behind building a methodology that efficiently extracts linear programming models from databases without domain-specific formulation knowledge.

In this paper, we will propose a two-phased linear programming model discovery method. The first phase is to find exhaustively well-formed modules from a data dictionary by applying the General Problem Solver (GPS) algorithm. We will call it the first-cut model discovery. The second phase is to validate models among well-formed modules, which is called the second-cut model discovery. The back propagation method, one of the Artificial Neural Network methods, is adopted as a model validation tool.

The remainder of this paper is organized as follows : In Chapter 2, a literature survey on model management and formulation support systems is described briefly. Overall framework for the linear programming model discovery is illustrated in Chapter 3. The first part, the first-cut model discovery from data dictionary, is delineated in Chapter 4. The Chapter 5 shows the second-cut model discovery by applying Artificial Neural Network system. In Chapter 6,

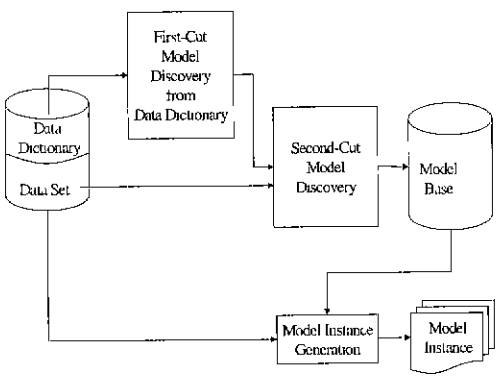
an illustrative example is given in order to show the feasibility of the proposed methodology. Finally, concluding remarks and future research issues are presented in Chapter 7.

2. Model Formulation Support Systems

Model formulation is a process of creating models that are efficiently solvable by some mathematical programming solution method [33]. The prevailing approaches to formulating mathematical programming models involve specific modeling languages and domain knowledge representations. Block-link approach is proposed to support the formulation procedures intelligently [29], and the interface design for software systems, LPFORM, in graphical forms is described [27]. PM*, a logic-based modeling language, is used to divide knowledge into domain specific knowledge in logic forms. Procedural modeling knowledge then treats the domain specific knowledge as an instance of modeling knowledge [5, 6]. A prototype system in IMMPS project to support formulation, discourse, and analysis support is proposed [15]. Case-based reasoning techniques were also applied to model formulation supports [25, 26, 32]. A control blackboard approach to the simulation of control features observed in the expert's model formulation protocols are proposed[31]. In addition, a variety of expert system technologies are proposed as knowledge-based model formulation systems [21, 24]. Finally, an object-oriented approach to model formulation is pursued since it is well suited for the representation and implementation of reusable modeling constructs in a large scale decision support systems [3, 22].

3. Overall Framework

An overall structure of the two-phase model formulation from data model is shown in [Figure 1]. The knowledge of model formulation is divided into three phases : (1) first-cut model discovery from data dictionary, (2) second-cut model discovery and (3) model instance generation. Data dictionary embeds the information regarding modeling constructs. Most of the domain knowledge for model formulation is directly imported from databases. Sets, parameters and decision variables are imported from relations. Indices are deduced from the relational keys specified in data models [11]. In addition, model constructs that are compound of relations, keys of relations, and operators, e.g., arithmetic, sum, product and comparison, can be incorporated by data manipulations. However, the databases do not contain knowledge on constraints and objective functions, which requires additional efforts. To increase the speed of finding models from databases, rather than investigating all possible linear combination of elements, the first-cut model discovery derives some useful knowledge using the restricts from mining a "syntactically verified" set of Structured



[Figure 1] Overall Framework

Modeling elements from the original data dictionary. We will call is a 'metaknowledge' in this paper. The metaknowledge is based on the means-ends analysis. The means-ends analysis applied to General Problem Solver(GPS), is one of the problem solving methodologies in cognitive science [23, 30]. The key activities of the analysis are (1) to analyze current state to a series of differences, (2) to decompose them into a set of subproblems and (3) to select relevant operators to remove the differences. Thus, the metaknowledge consists of identifying the gaps between the current and goal states, decomposing problems based on a 'divide-and-conquer' strategy and solving until no gaps are left. As a result, we can acquire a first-cut model by collecting candidate SM elements. The fist-cut model, as structural domain knowledge, may contribute to the problem identification and second-cut model formulation [9].

The second-cut model discovery is to filter a "semantically verified" set of Structured Modeling elements from the first-cut model. To do so, an Artificial Neural Network model is designed to evaluate how well the elements in the first-cut model are fitted to the information in the database. In case of a test element, each model elements are registered as a set of input nodes, and the feasibility is registered as output nodes. The neural network performs learning by training and testing data that are randomly generated. Then, the system acquires interrogating data from the error rates. If the error rates from training data and interrogating data are statistically indifferent and also if the average error rates are scored under a threshold level, then the model element will be regarded as semantically meaningful. Thus, the "semantically" verified set of SM elements is stored as a model base.

4. First-Cut Model Discovery from Data Dictionary

The first-cut model discovery is to build syntactically a set of well-formed modules from a data dictionary using the GPS algorithm. The GPS algorithm is well suited for the problem domain, which satisfies the following conditions: first, the domain states should be finite; second, the operators that move a state to another state exist. And finally, the problem should be represented as a state. Semantics on database is embedded in a data dictionary.

In general, a data dictionary contains names, aliases, classifications, descriptions, attributes with data types and units, standard representations and entity class memberships. An attribute has its own data type, index set, and units [10]. We define the specification of the data type, index, and units of an attribute as a in a set of all possibles A is represented as the following 3-tuple .

$$S=\langle DT_s, I_s, U_s\rangle$$

where DT_s : data type of $S, DT_S \in DT: = \{DT = \{\forall S \in S, DT_S\}$

I_s : index set of $S, I_S \in I_S: = \{I = \{\forall S \in S, I_S\}$

U_s : data unit of(1) $S, U_S \in I_S: = \{U = \{\forall S \in S, U_S\}$ (1)

Besides, let an attribute A(A∈A, A is a set of attributes of selected problem domain) exist, then a many-to-one mapping function from an attribute to its state as ρ which satisfies:

$$\rho(A) = S \quad (2)$$

can be derived. Then, from equations (1) and (2):

$$\rho(A) = S = \langle DT_{\rho(A)}, I_{\rho(A)}, U_{\rho(A)} \rangle \quad (3)$$

is satisfied.

In LP model formulation, it is observed that each left hand side (LHS) and right hand side (RHS) in every valid objective function and constraints can be identified as a state, and it satisfies : (1) the data type (2) the index set and (3) the identical unit of LHS and RHS. For example, suppose a constraint, $\sum_i A_{ij} * X_{ij} \leq B_j$, is valid. Then the data type, index set, and unit of $\sum_i A_{ij} * X_{ij}$ and B_j are the same. On the other hand, $\sum_i A_{ij} * X_{ij} \leq D_i$ is not valid since the index set of LHS ({j}) is different from that of RHS({i}). Murphy addressed that the formulation of the model is valid if the index set of each LHS matches that of RHS [28]. In general, if at least one of them is not equivalent, the constraint is no more valid and additional operations to perish the differences are required. Thus, the necessary condition of model validation is defined as:

[Definition] *Necessary condition of Model Vaildation*: If an LP model is valid, then LHS and RHS of the model have the same data type, index sets, and units.

The condition implies that if a modeller elicits a set of model constructs to formulate an objective function or constraints, then at least the data type, index sets, and units of the LHS and RHS should be identical. Meanwhile the activities of model formulation usually begin with the identification of decision variables. If RHS is given, then the model formulation effort simplifies to an activity to determine the relationships among decision variables and RHS including the identification of the differences between the data type, index set and units of selected decision variables and RHS constants. Suppose a problem, P, is defined as follows [16]:

$$P = \langle I(s), G(s), M, S \rangle \quad (4)$$

where
$I(s)$: initial state (a state of decision variable),
$G(s)$: goal state (a state of RHS),
M : set of operators,
S : set of finite states.

Then, solving for P can be interpreted as reducing the gaps between $I(s)$ and $G(s)$ by applying appropriate M. When applying operators, the state changes are added to S. In LP formulation problems, the initial and goal states correspond to decision variables and RHS constants, respectively.

4.1 Initial State and Goal State

In formulating models, the problem can be identified as a discrepancy between present and desired states of a system, for which the initial and goal states must be identified and represented as follows :

$$I(s) = \langle DT_{I(S)}, I_{I(S)}, U_{I(S)} \rangle, \quad (5)$$

$$G(s) = \langle DT_{G(S)}, I_{G(S)}, U_{G(S)} \rangle. \quad (6)$$

4.2 Differences

There are three types of differences: namely, characteristic, structural, and unit differences.

4.4.1 Characteristic Differences

Characteristic differences are the differences of data types between any two states. Data types can be classified as primitive data types and derived data types. The primitive data types are the fundamental dimensions: e.g., length, mass, time, currency, etc. The derived data types are

the functions of primitive data types: e.g., volume (=length^3), acceleration (=length / time^2), unit cost (=currency/mass), etc. Besides, the attributes of whichever data type is null can also exist. In general, if two variables have the same data type, then it is always possible to calculate if in fact their values are equivalent [7]. The data type (DT) has the following characteristics :

For all state X, Y $\in S$, if $DT_X = DT_Y = a$, then
$DT_{X+Y} = DT_{Y+X} = a$(Commutative),
$DT_{X-Y} = DT_{Y-X} = a$(Commutative),
$DT_{X/Y} = DT_{Y/X} =$ null(Simplification) and
$DT_{X*Y} = DT_{Y*X}$ is undefined.

For all X, Y and Z $\in S$, is $DT_Y = DT_Z = a$, then
$DT_{(X/Y)*Z} = a$,
$DT_{X/Y} =$ null,
$DT_{C*X} = DT_X$, where $DT_C =$ null and
$DT_{X/(Y*Z)}$ is undefined.

For more information on the laws of structural arithmetics, one can refer to Bhargava's dimensional analysis[8].

[Example 1] For example, suppose any two attributes A_{ij}, C_{ij} exist and $\rho(A_{ij})$= <quantity/time, {I,j},_>, $\rho(C_{ij})$ =<quantity/time, {I,j}, thousand>, respectively. Then $DT_{\rho(C_{ij}+A_{ij})} = DT_{\rho(A_{ij}+C_{ij})} = DT_{\rho(A_{ij}+C_{ij})}$ = '*quantity/time*', that is, the data type of $A_{ij}+C_{ij}$ and $C_{ij}+A_{ij}$ is identic a : 'quantity/time'.

The characteristic differences can be defined as:

[Definition] *Characteristic Differences :* For any X, Y $\in S$, if DT_X and DT_Y are the data types of X and Y, then there are characteristic differences between X and Y, if $DT_X \neq DT_Y$.

This implies that since the relationships between LHS and RHS in LP model are linked in a form of equality (=) or inequality (<=, >=), constraints are valid only if there are no characteristic differences.

4.4.2 Structural Differences

Structural differences are related to the differences among index sets of attributes. They are essential for model composition in model formulation [28].

[Definition] *Structural Differences* . For any X, Y $\in S$, if I_X and I_Y are the index set of X and Y, then there is a structural difference between X and Y, if and only if $I_X \neq I_Y$.

[Example 2] Suppose that two states $X=\rho(X_{ij})$ and $Y=\rho(B_i)$ exist, then $I_X=\{i, j\}$, $I_Y=\{i\}$. If it is necessary to sum X_{ij} by j then they have no structural differences since $X' = \rho(\Sigma_j X_{ij})$ and $I_{X'} = \{i\} = I_Y$. In mathematical programming, 'Σ' is used to remove structural differences.

4.3.3 Unit Differences

Each quantity has a base unit of measurement along with several other units. In the International Metric System, there are seven fundamental quantities (or data types) and base units[8] : length(meter. kilometer, centimeter), mass(kilogram, gram, milligram), time(second, hour, minute), electric current(ampere, millamps), temperature(Kelvin, Celsius, Fahrenheit), luminou (candala, candle-power) and amount of substance (mole, kilomole). In addition, currency (dollars. wons, etc.) is important in mathematical programmeming There may be a difference between

attributes that have the same data type, and hence, the unit difference is meaningful only when they have the same data type.

[Definition] *Unit Differences*: For any X and Y $\in S$, which satisfy $DT_X = DT_Y$, where U_X and U_Y are the units of X and Y, respectively. Then, there is a unit difference between X and Y if $U_X \neq U_Y$.

[Example 3] For example, suppose a state X that represents LHS piece ('piece' means a set of coefficient and decision variable) and another state Y that represents RHS constant have the same data type named 'length', and the unit of X is 'kilo-meters' and that of Y is 'meters', there is a unit difference between X and Y since U_X = 'kilo-meters' and U_Y = 'meters'. To remove the difference we may divide X by 1,000: X' = X/1000. Then, $U_{X'} = U_{(X/1000)}$ = 'meters' = U_Y.

4.3 Operators

Operators are used to remove the three types of difference. There are a few feasible operators: e.g. '+', '−', '*', '/', and 'Σ'. Formally, the operator M($M \in \boldsymbol{M}$, $\boldsymbol{M}$ is a set of operator) is represented as :

$M: X \rightarrow Y$
where
X, Y : a state in $\boldsymbol{S}$
$\boldsymbol{S}$: a set of all possible states.

[Example 4] In LP, **M** = {'+', '−', '*', '/', 'Σ'}. Suppose that X = <quantity, {i,j}, million> and Y = <quantity, {i}, million> exist. Since there is a structural difference (I_X={i,j}≠{i} = I_Y), an operator may be applied to X to remove the difference. The operator will change X into X'= <quantity, {j}, million>.

4.4 Modeling Engine : Metaknowledge for Removing Differences

The modeling engine guides the navigation within the domain space. It contains metaknowledge for removing differences. The basic concept of metaknowledge is compound of analyzing differences, searching proper operations and applying them. The metaknowledge is used for removing characteristic differences, removing structural differences and removing unit differences.

4.3.1 Metaknowledge for Removing Characteristic Differences

Assume that there are two states, X and Y, and the data types of the states are DT_X and DT_Y, respectively. If DT_X does not equal DT_Y, then a characteristic difference exists and a set of new attributes to which the data types must be multiplied to DT_X. The modeling engine procedure for characteristic differences is as follows:

[Step 1.1] Analyze the data type of the initial state X($DT_X \in DT$), and the goal state, Y(($DT_Y \in DT$).

[Step 1.2] Compare the differences.
- If there is no difference ($DT_X = DT_Y$), then stop.
- Otherwise, go to [Step 1.3]

[Step 1.3] Seek an attribute of which the data type is DT_Y/DT_X that makes DT_X into DT_X* (DT_Y/DT_X) = DT_Y.
- If found, then multiply the attribute to X and stop.
- Otherwise, go to [Step 1.4]

[Step 1.4] Seek attributes of which the data types are DT_Y/a and a/DT_X, respectively, that

makes DT_X into $DT_X*(DT_Y/a)*(a/DT_X)=DT_Y$.

- If both of them are found. then multiply them to X and stop.
- Otherwise, go to [Step 1.5]

[Step 1.5] Seek attributes of which the data types are DT_Y/a and b/DT_X, respectively.

- If both of them are found, then multiply them to X and set it as X and go to [Step 1.1]
- Otherwise, stop. Insoluble.

4.3.2 Metaknowledge for Removing Structural Differences

Let the index set of state X and Y, I_X and I_Y, exist, and a candidate attribute $C(I_C)$, which can resolve the characteristic difference, is entered. Then the modeling engine procedure for structural difference is as follows:

[Step 2.1] Analyze the index set of the initial state $X(I_X \in I)$ and goal state $Y(I_Y \in I)$

[Step 2.2] Compare the index set of them.

- If $I_X \supseteq I_Y$, then apply $\Sigma_{(I_x - I_y)}$ to I_X where $\Sigma \in M$, then stop.
- If $I_X, I_y \neq \emptyset$ and $I_X \cap I_y \neq \emptyset$, then go to [Step 2.3].
- Otherwise, Stop. Insoluble.

[Step 2.3] Search for any attribute C of which the index set I_C satisfies $I_C \supseteq (I_x \cup I_Y)$.

- If found, then multiply C to X and apply $\Sigma_{(I_x)}$ to I_X. Stop
- Otherwise, stop. Insoluble.

4.3.3 Metaknowledge for Removing Unit Differences

The transformation of units is required in the model solution and model integration [8]. The algorithm is as follows:

[Step 3.1] Analyze the units of the initial X (U_X) state and goal state Y(U_Y).

[Step 3.2] Compare U_X and U_Y.

- If $U_X = U_1$, then stop.
- Otherwise, multiply U_Y/U_X to U_X, and then stop.

4.5 First-cut Model Discovery Process

The first-cut model discovery process involves problem definition, sub-problem generation and difference resolution shown as follows:

[Step 1] Problem Definition

[Step 1.1] Determine a problem domain and identify it as $\boldsymbol{A}$ (a set of attributes defined in the data dictionary) and corresponding state $\rho(A) = S$.

[Step 2] Sub-problem Generation

[Step 2.1] Elicit a decision variable in $\boldsymbol{A}$ and identify it as an initial state $I(S) = \langle DT_{I(S)}, I_{I(S)}, U_{I(S)} \rangle$.

[Step 2.2] Elicit candidates for RHS in $\boldsymbol{A}$.

[Step 2.3] For each candidate, decompose the original problem into sub-problems. Each of the sub-problems are represented as a goal state (G(S)): $G(S) = \langle DT_{G(S)}, I_{G(S)}, U_{G(S)} \rangle$.

[Step 3] Difference Resolution. For all sub-problems,

[Step 3.1] Apply relevent modelling engine.

[Step 3.2] Repeat step 3.1 until either of the following conditions is satisfied.

(1) If no differences are found, then stop. The sub-problem is solved.

(2) If any differences are found and no relevant modelling engines can be applied,

then stop. The sub-problem is insoluble.

[Step 4] Determine a model that is made up of solved subproblems.

5. Second-Cut Model Discovery

5.1 Model Variation

The modules generated in the previous phase may be varied since the model formulation by removing three types of differences between LHS and RHS requires only a necessary condition for validating models. Hence, the model variation is required in order to validate them. Four considerations for model variation are proposed : intervention of attributes that have a null data type, interaction within RHSs, intersection of other decision variables and intervention of new decision variables.

Intervention of attributes that have a null data type

It is necessary to refine models by adding coefficients to LHS or RHS. If the data type of an attribute is null (e.g. constant), then it cannot contribute to model formulation since it is thought that no characteristic differences can be detected. It searches for the following possibilities:

If $A=\langle DT_A, I_A, U_A\rangle$ and
$B=\langle DT_B, I_B, U_B\rangle$
where $DT_A=DT_B$, $I_A=I_B$, and
$U_A=U_B$ exists,

then check if a state C that satisfies:

$$DT_{A*C}=DT_A(DC_C=null) \text{ exists.}$$

[Example 5] Suppose that $\sum_j X_{ij}\le B_i(\rho(X_{ij})=X, \rho(B_i)=B)$ is formulated as a constraint that satisfies $DT_X=DT_B$, $I_X=I_B$ and $U_X=U_B$, that is, there is no difference. However, if a constant $c(\rho(c)=C$ that satisfies $DT_{C*X}=DT_B$, $I_{C*X}=I_B$ and $U_{C*X}=U_B$ exists, and to append, it is semantically right, then the constraint should be updated to $c*\sum_j X_{ij}\le B_i$.

Interaction within RHSs

It implies that if some RHSs are interrelated and can be compressed, then they can be congruent in one constraint. It searches for the following possibilities:

If $A=\langle DT_A, I_A, U_A\rangle$, $B=\langle DT_B, I_B, U_B\rangle$, and
$C=\langle DT_C, I_C, U_C\rangle$, where
$DT_A=DT_B=DT_C$, $I_A=I_B=I_C$, and
$U_A=U_B=U_C$ exists,

then check if a function f satisfies:

$$DT_A=DT_{f(B,C)}, I_A=I_{f(B,C)},$$
$$U_A=U_{f(B,C)} \text{ exists.}$$

[Example 6] Suppose that two constraints $\sum_j A_{ij}X_{ij}\le B_i$ and $\sum_j A_{ij}X_{ij}\le C_i$ are generated in the previous phase. It is possible that $\sum_j A_{ij}X_{ij}\ge 2*B_i+C_i$ is semantically right.

Intersection of other decision variables

It is possible that other decision variables may affect RHS. It searches for the following possibilities:

If $A=\langle DT_A, I_A, U_A\rangle$, $B=\langle DT_B, I_B, U_B\rangle$, and
$C=\langle DT_C, I_C, U_C\rangle$
where $DT_A=DT_B=DT_C$, $I_A=I_B=I_C$, and
$U_A=U_B=U_C$ exist,

respectively, then check if a function f that satisfies:

$DT_{f(A,B)} = DT_C$, $I_{f(A,B)} = I_C$, and $U_{f(A,B)} = U_C$ exists.

[Example 7] If two constraints, $\sum_j A_{ij}X_{ij} \geq B_i$ and $\sum_j C_{ij}Y_{ij} \geq B_i$ are formulated as candidates, then a varied constraint, $\sum_j A_{ij}X_{ij} - \sum_j C_{ij}Y_{ij} \geq B_i$, is also syntactically right.

Intervention of new decision variables

It specifies the existence of intervening (indirect) effects between an antecedent variable and its consequent variable. It searches for the following possibilities:

If $A = \langle DT_A, I_A, U_A \rangle$, $C = \langle DT_C, I_C, U_C \rangle$, where $DT_A = DT_C$, $I_A = I_C$, *and* $U_A = U_C$ exist,

then check if a state $B = \langle DT_B, I_B, U_B \rangle$ that satisfies:

$DT_A = DT_B = DT_C$, $I_A = I_B = I_C$, *and* $U_A = U_B = U_C$ exists.

[Example 8] Suppose a constraint $\sum_i A_{ij}X_{ij} \geq C_j$ exists. It is possible that B_{kj} and Y_{kj} exist and (i) $\sum_i A_{ij}X_{ij} \geq \sum_k B_{kj}/y_{kj}$ and (ii) $\sum_k B_{kj}Y_{kj} \geq C_j$ are also syntactically right.

5.2 Second-Cut Model Discovery Process

To construct a valid model, the first-cut modules and the derived modules by model variation activities are to be tested. Actually, it is not easy for generic modeling systems, as proposed in this paper, to warrant model validity [25]. One of the typical methods for model validity is to test the feasibility by using past data [2], and specially test by causal modelling or path analysis [1]. We combine the two methods using the back propagation method, which is one of the Artificial Neural Network techniques for knowledge discovery. The rationale of applying neural network techniques to model validation is that if a module is right, then learned patterns from the real data set and those from artificial data set will be similar. In other words, if a module is valid, then the similarity of real feasbilility set and estimated feasibility set through the module and randomly generated data set will be signicantly high. For this kind test, firstly, an artificial data set of decision variables is prepared by the random number generation. Then, the feasibility values for each data records are computed as a set of binary values by applying to the module. For the nest step, a neural network model is designed. In designing the neural network model, the input nodes represent decision variables in a first-cut module, and the output nodes indicate the feasibility of the module. For example, if there is a module such as:

$$\sum_i USING\_RATE * PROD\_QUANTITY \leq MAXIMUM\_DAILY\_CAPACITY,$$

then the decision variables, PROD_QUANTITY, represent the input nodes.

At this time, it is our assumption that if the network model learns with training and testing data which are artificially prepared, then the correctness level of the network model from the artificially prepared data set and that from the real data would be most similar when compared with other modules in the same variation group. As a matter of course, if the correctness levels of all modules in a same variation group do not

compare to the correctness level from the artificially prepared data set, then the modules are judged as invalid.

There are two useful measures: similarity level and usefulness level. Similarity level measures how the correctness of learning rule interrogated by real data is similar to that of learning rule interrogated by training and testing data which are artificially and randomly prepared.

Similarity Level :

$S(i, \alpha_1, \alpha_2, \ldots, \alpha_n) = 100 - |2C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 3) - C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 1) + C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 2))|'$

where $S(i, \alpha_1, \alpha_2, \ldots, \alpha_n)$ designates the similarity level of module i, and $C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 1)$ $C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 2)$ and $C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 3)$ designate the correctness level for module i by training, testing and real data, respectively.

The usefulness level indicates shows how well the learning rule from a neural network model explains the real situation.

Usefulness Level :

$U(i) = (C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 1) + C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 2) + C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 3))/3,$

where U(i) indicates the useful level of module i.

For example, let us assume that the following module exists in the same model variation group.

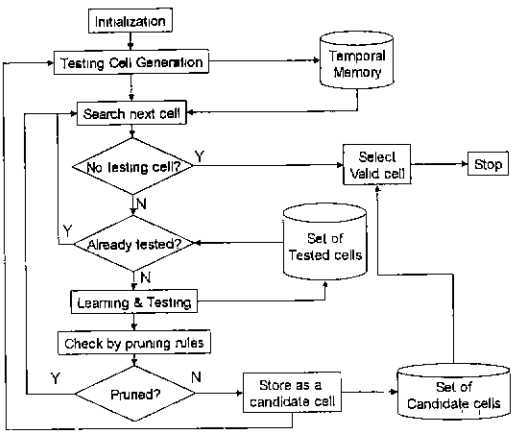
$$\sum_i USAGE\_RATE * PROD\_QUANTITY \le \alpha_1 MAXIMUM\_DAILY\_CAPACITY + \alpha_2 WEIGHT$$

where α_1, α_2 are real numbers.

A procedure for the second-cut model discovery shown in [Figure 2] (e.g.: $\alpha_1 b_1 + \alpha_2 b_2$) is listed as follows :

[Step 0] Initialization

Initialize $\alpha_1 = 1, \alpha_2 = 0$. Run the cell. Then acquire $S(i, \alpha_1, \alpha_2)$ and insert into the set of candidates.



[Figure 2] A Proceudure for second-cut model discovery

[Step 1] Testing cell generation

Generate eight neighboring testing cells where $(\delta_1 = \delta, \delta_2 = 0), (\delta_1 = \delta, \delta_2 = -\delta), (\delta_1 = \delta, \delta_2 = \delta), (\delta_1 = 0, \delta_2 = -\delta), (\delta_1 = 0, \delta_2 = \delta), (\delta_1 = -\delta, \delta_2 = 0), (\delta_1 = -\delta, \delta_2 = -\delta)$ and $(\delta_1 = -\delta, \delta_2 = \delta)$ Then, store them in the temporary memory.

[Step 2] Termination check

Check if there is no testing cell in the temporary memory, then go to step 6. Otherwise, go to step 3.

[Step 3] Running

For a generated cell in the temporal memory, test if the cell has already been tested. If so, prune it and go to step 2. Otherwise, run the cell and then determine $S(i, \alpha_1 + \delta_1, \alpha_2 + \delta_2) - S(i, \alpha_1, \alpha_2)$.

[Step 4] Check by prunung rules

Check if the cell ($\alpha_1+\delta_1, \alpha_2+\delta_2$) satisfies at least one of the following pruning rules, then, prune it.

Rule1 : If $U(i)<\lambda$, then prune it.

Rule2 : If $C(i, \alpha_1, \alpha_2, \ldots, \alpha_n, 3) \leq \eta$ then prune it, where η designates the minimum allowable correctness level.

Rule3 : If $S(i, \alpha_1+\delta_1, \alpha_2+\delta_2) - S(i, \alpha_1, \alpha_2) \leq \mu$, then prune it, where μ designates the maximum allowable declining amount of similarity level.

Otherwise, go to step 5.

[Step 5] Cell insertion

Insert the cell ($\alpha_1+\delta_1, \alpha_2+\delta_2$) into a set of candidate cells. Then set $\alpha_1=\alpha_1+\delta_1$, $\alpha_2=\alpha_2+\delta_2$, and then go to Step 2.

[Step 6] Valid module determination

Select a cell that has the maximum similarity level in a set of candidates cells as a valid module.

6. An Illustrative Example

To show the feasibility of the idea described in this paper, let us show an illustrative example. The <Table 1>, <Table 2> and <Table 3> show data dictionaries and some records of the example. In each table, first row designates the data field name. The underlines are the candidate keys. The second row shows the data type of fields. The third row contains the unit of corresponding fields, if any. Finally, the fourth row indicates the example data.

<Table 1> Raw_Material(i)

RAW_ ID	RAW_NAME	MIN_INV_ REQ	UNIT_ COST	MAXIMUM_ DAILY_ CAPACITY
String(4)	String(20)	Numeric	$/Numeric	Numeric
-	-	Kg	1000	Tons
M1	Raw Material 1	5,000	1.2	24
M2	Raw Material 2	1,500	0.9	6

<Table 2> Product(j)

PROD_ ID	PROD_ NAME	PROD_ DATE	PROD_ QUANTITY	UNIT_ PROFIT
String(3)	String(20)	Date	Numeric	$/ Numeric
-	-	-	1000	1000
X1	Exterior Paint	1998/10/8	45.2	5
X1	Exterior Paint	1998/10/9	37.5	5
X1	Exterior Paint	1998/10/10	40.2	5
X1	Exterior Paint	1998/10/11	41.3	5
X1	Exterior Paint	1998/10/12	47.6	5
.		..	...	..
X2	Interior Paint	1998/10/8	35.3	4
X2	Interior Paint	1998/10/9	24.5	4
X2	Interior Paint	1998/10/10	26.7	4
X2	Interior Paint	1998/10/11	19.0	4
X2	Interior Paint	1998/10/12	21.7	4
.	...	.	...	...

<Table 3> Involves(i,j)

RAW_ID	PROD_ID	Equipments	USAGE_ RATE
String(2)	String(2)	String(10)	Numeric / Numeric
-	-	-	Tons
M1	X1	Mixer1	6
M1	X2	Mixer1	4
M2	X1	Mixer2	1
M2	X2	Mixer2	2

The data dictionary on the above example can be described as follows:

$\rho(RAW\_ID)$ =<string(4), {i}, _>

$\rho(RAW\_NAME)$ =<string(20), {i}, _>

$\rho(MIN\_INV\_REQ)$ =<numeric, {i}, kg>

$\rho(UNIT\_COST)$ =<$/Numeric, {i}, 1000>

$\rho(MAXIMUM\_DAILY\_CAPACITY)$ =<numeric, {i}, tons>

$\rho(PROD\_ID)$ =<string(3), {j}, _>

$\rho(PROD\_NAME)$ =<string(20), {j}, _>

$\rho(PROD\_DATE)$ =<date, {j}, _>

$\rho(PROD\_QUANTITY)$ =<numeric, {j}, 1000>

$\rho(UNIT\_PROFIT)$ =<$/numeric, {j}, 1000>
$\rho(EQUIPMENTS)$ =<string(10), {i,j}, _>
$\rho(USAGE\_RATE)$ =<numeric/numeric, {i,j}, tons>

<Table 4> shows the first-cut model discovery process for the given example.

As a result, the following two modules are deduced.

$$1000 * \sum_j USAGE\_RATE * PROD\_QUANTITY \leq MAXIMUM\_DAILY\_CAPACITY$$

$$1000000 * \sum_j USAGE\_RATE * PROD\_QUANTITY \geq MIN\_INV\_REQ$$

In order to construct a second-cut model, applying the model variation rules tests the first-cut modules. According to the interaction among RHSs rules, the following module is generated:

$$1000 * \sum_i USAGE\_RATE * PROD\_QUANTITY \leq \alpha_1 \; MAXIMUM\_DAILY\_CAPACITY - 0.001 * \alpha_2 \; MIN\_INV\_REQ$$

Where α_1, α_2 *is real*

For the back propagation, Neural Planner Ver.4.52 is adopted. Neural Planner is a neural network system for Microsoft Windows. Neural Planner can learn from training files, self-test using testing files and are interrogated by interrogating files. It can produce spreadsheet or hierarchical output files interactively. The [Figure 3] shows an example learning mode interface.

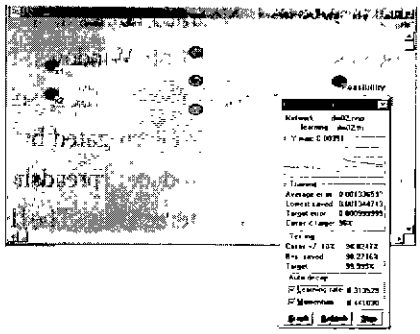
$$1000 * \sum_i USAGE\_RATE * PROD\_QUANTITY \leq 1.0 * MAXIMUM\_DAILY\_CAPACITY - 0.001 * 10 * MIN\_INV\_REQ$$

Then to make a valid model, a neural network model is designed as follows :

Input Notes: PROD_QUANTITY (X1, X2)
Hidden Nodes: three hidden nodes
Output Nodes: Feasibility
Training and testing data set: 1,000 records of PROD_QUANTITY (X1, X2) are artificially acquired by random number generation.
Interrogating data set: 600 records of PROD_QUANTITY (X1, X2) are prepared from real data file.
Initial condition (parameters):
α_1 = 1.0
α_2 = 0.0
δ = 0.2
η = 0.6
μ = 0.05

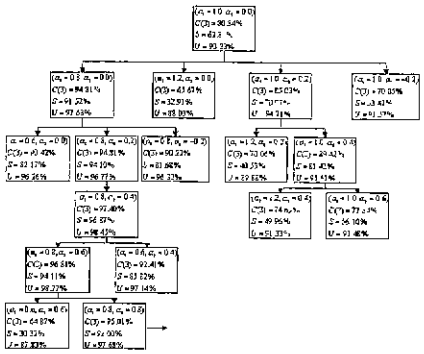
〈Table 4〉 Process of first-cut model discovery process for the given example

Step	Current State	Differences	Operators	Goal State
1	ρ (*PROD_QUANTITY*)			ρ (MAXIMUM_DAILY_CAPACITY)
1-1	ρ (*PROD_QUANTITY*) = <numeric, {j}, 1000>	Structural Differences	Find {i,j} $\sum_j USAGE\_RATE$	ρ (MAXIMUM_DAILY_CAPACITY) = <numeric, {i}, tons>
1-2	<numeric, {j}, 1000*tons>	Unit Differences	Multiply 1000 to LHS	<numeric, {i}, tons>
1-3	<numeric, {j}, tons>			<numeric, {i}, tons>
2	ρ (*PROD_QUANTITY*)			ρ(*MIN_INV_REQ*)
2-1	ρ (*PROD_QUANTITY*) = <numeric, {j}, 1000>	Structural Differences	Find {i,j} $\sum_j USAGE\_RATE$	ρ (*MIN_INV_REQ*) = <numeric, {i}, kg>
2-2	<numeric, {i}, 1000*tons>	Unit Differences	Multiply 1000*1000 to LHS	<numeric, {i}, kg>
2-3	<numeric, {i}, kg>			<numeric, {i}, kg>

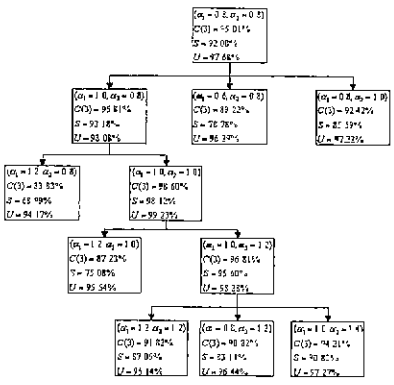


[Figure 3] Sample screen of Neural Planner

[Figure 4(a)] and [Figure 4(b)] show the search tree for the example. The blue-colored cells designate pruned cells. Finally, cell 13 is selected as the optimal solution, and therefore. the following module is selected as a second-cut module.



[Figure 4(a)] Search Tree



[Figure 4(b)] Search Tree (continued)

Finally, the executable SML program generated from the second-cut model is as shown in [Figure 5].

&PROD PRODUCTION SECTOR

RAW_MATERIALi /pe/ There is a list of RAW MATERIAL MAXIMUM_DAILY_CAPACITY (RAW_MATERIALi) /a/ RAW_MATERIAL . Real+ Every RAW_MATERIAL has a MAXIMUM_DAILY_CAPACITY measured in tons.

MIN_INV_REQ (RAW_MATERIALi) /a/ RAW_MATERIAL : Real+ Every RAW_MATERIAL has a MIN_INV_REQ measured in kilograms.

PROCUTj /pe/ There is a list of PRODUCT PROD_QUANTITY (PRODUCTj) /va/ PRODUCT : Real+ Every PRODUCT has a nonnegative PRODUC_QUANTITY measured in thousands

UNIT_PROFIT (PRODUCTj) /a/ PRODUCT : Real+ Every PRODUCT has a nonnegative UNIT_PROFIT measured in thousands

INVOLVES (RAW_MATERIALi, PRODUCTj) /cc/ Select RAW_MATERIAL x PRODUCT where i covers RAW_MATERIAL, j covers PRODUCT A PRODUCT INVOLVES RAW_MATERIAL There must be at least one INVOLVES incident to each RAW_MATERIAL, and at least one INVOLVES incident to each PRODUCT

USAGE_RATE (INVOLVESij) /a/ INVOLVES : Real+ There can be a nonnegative USAGE RATE over each INVOLVES

$ (PRODUCT) /f/ 1 ; @ SUMj (PROD_QUANTITYj * UNIT_PROFITj) There is a TOTAL PROFIT associated with all PRODUCT

T:SUP (RAW_MATERIALi., RAW_MATERIALi) /t/ RAW_MATERIAL ; 1000 * @SUMj (USAGE_RATEij * PROD_QUANTITYj) <= MAXIMUM_DAILY_CAPACITYi 0 001 * MIN_INV_REQi Is the total RAW_MATERIAL used to product PRODUCT to PRO_QUANTITY level less than or equal to MAXIMUMDAILY_CAPACITY minus MIN_INV_REQ? There is called the SUPPLY TEST

[Figure 5] The example output SML program

7. Conclusion

Comparing with data mining techniques, the methodology produces additional kind of knowl-

edge. Finding useful knowledge from database has become more critical in the recent competitive and enterprising environment. The knowledge has been represented as association rules, classification, clustering, regression, etc. However, in databases, there is something more than what the current knowledge discovery techniques can provide. We drew attention to the mining optimization knowledge and investigate why the linear programming models would be one of the most useful forms of knowledge buried in databases. As a result, linear programming model discovery from database is proposed in this paper.

Furthermore, comparing with legacy knowledge-based model formulation support systems, our methodology provides decision makers with more general model formulation support through separating generic formulation knowledge and domain-specific formulation knowledge. We show how the GPS and neural network technique can be combined to discover models from databases. The methodology is more beneficial to the data-oriented decision support systems like data warehousing and OLAP because it is adaptable to domain changes. The neural network can dynamically accommodate domain changes and produces newly modified domain-specific formulation knowledge. Hence, the capability may overcome the limitations of mesa effect that legacy knowledge-based formulation support systems possess.

REFERENCES

[1] Asher, H.B., *Causal Modeling, Beverly Hills*, CA: Sage Publication Inc., 1983.

[2] Ata, N., Courtney, J.F., and D.B. Paradice, "A Prototype DSS for Structuring and Diagnosing Managerial Problems", *IEEE Transactions on System. Man, and Cybernetics*, Vol.18, No.6(1988), pp.899-907.

[3] Becker, K. and Bodart, F, "Methodological and Software Environment for Model Formulation Based on Reusable Object Frameworks", *Proceedings of the International Conference on Decision Support System*, Vol.15(1995), pp.611-626.

[4] Berry, M., and G. Linoff, *Data Mining Techniques : For Marketing. Sales, and Customer Support*, John Wiley & Sons, Inc., 1997.

[5] Bhargava, H.K., "A Simple and Fast Numerical Method for Dimensional Arithmetic", Naval Postgraduate School, *Working Paper* No.90-01(1990).

[6] H.K. Bhargava, and R. Krishinan, "A Formal Approach for Model Formulation In a Model Management System", *The 24th Hawaii International Conference on System Sciences*(1990), pp.453-462.

[7] H.K. Bhargava, and S.T. Kimbrough, "On Embedded Languages for Model Management", *The 24th Hawaii International Conference on System Sciences*(1990), pp.443-452.

[8] Bhargava, H.K., "Dimensional Analysis in Mathematical Modeling Systems : A Simple Numerical Method," *ORSA Journal on Computing*, Vol.5, No.1(1993), pp.33-39.

[9] Binbasioglu, M., "Role of Structural Domain Knowledge in Problem Understanding and Model Formulation", *Proceedings of the Annual Meeting Decision Sciences Institute*(1990), pp.341-344.

[10] Brathwaite, K.S., Analysis, *Design, and Implementation of Data Dictionaries*, McGraw-

Hill Book Company, CA, 1988.

[11] Choobineh, J., "SQLMP : A Data Sublanguage for Representation and Formulation of Linear Mathematical Models", *ORSA Journal on Computing*, Vol.3, No.4(1991), pp.358-375.

[12] Fayyad, U.M., Piatetsky-Shapiro, G., and P. Smyth, "From Data Mining to Knowledge Discovery", In *Advances in Knowledge Discovery and Data Mining*(1996), pp.1-34.

[13] Frawley, W.J., Piatetsky-Shapiro, G., and C.J. Matheus, "Knowledge Discovery in Databases : An Overview", In *Knowledge Discovery in Databases*, ed. G. Piatetsky-Shapiro and B. Frawley, Cambridge, Mass : AAAI/MIT Press(1991), pp.1-27.

[14] Geoffrion, A.M., "Reusing Structured Models via Model Integration", *Proceedings of the Twenty Second Hawaii International Conference on System Sciences*(1989), pp. 601-611.

[15] H.J. Greenberg, *A Primer for MODLER: Modeling by Object-Driven Linear Elemental Relations*, University of Colorado at Denver, Denvor, Feb., 1991.

[16] Guvenir, H.A. and G.W. Ernst, "Learning Problem Solving Strategies Using Refinement and Macro Generation", *Artificial Intelligence*, Vol.36, No.2(1990), pp.209-243.

[17] Jiawei Han, "Mining Knowledge at Multiple Concept Levels", *Proc 4th Int'l Conf. on Information and Knowledge Management (CIKM'95)*, Baltimore,s Maryland, Nov. (1995), pp. 19-24.

[18] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases", *Proceedings of 1995 Int'l Conf. on Very Large Data Bases (VLDB'95)*, Zrich, Switzerland, September 1995, pp.420-431.

[19] Jiawei Han, Yongjian Fu, Wei Wang, Jenny Chiang, Wan Gong, Krzysztof Koperski, Deyi Li, Yijun Lu, Amynmohamed Rajan, Nebojsa Stefanovic, Betty Xia, Osmar R. Zaiane, "DBMiner : A System for Mining Knowledge in Large Relational Databases", *Proceedings of 1996 Int'l Conference on Data Mining and Knowledge Discovery (KDD'96)*, Portland, Oregon, August 1996, pp.250-255.

[20] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, K. Koperski, G. Liu, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. R. Zaiane, S. Zhang, H. Zhu, "DBMiner: A System for Data Mining in Relational Databases and Data Warehouses", *Proc. CASCON'97: Meeting of Minds*, Toronto, Canada, November 1997.

[21] Krishnan, R , Li, X. and D. Steier, "A knowledge-based mathematical model formulation system", *Communications of the ACM*, Vol. 35, No.9(1992), pp.138-146.

[22] Kwon, O.B., and S.J. Park, "RMT : A Modeling Support System for Model Reuse", *Decision Support System*, Vol.16, No.1 (1996), pp.131-153.

[23] Laird, J.E., Newell, A.,and P.S. Resenbloom, "SOAR : An Architecture for General Intelligence", *Artificial Intelligence*, Vol.33, No.1 (1987), pp.1-64.

[24] Li, X., Jobling, C.P., and P.W. Grant, "A Knowledge Representation Scheme for Automatic Model Formulation", *The 7th Symposium on Computer aided control systems design*(1997), pp.365-379.

[25] Liang, T.P., "Modeling by Analogy : A Case-Based Approach to Automated Linear

Program Formulation", *Proceedings of the Twenty Fifth Annual Hawaii International Conference on System Sciences*(1991), pp. 276-283.

[26] T.P. Liang, "Analogical Reasoning and Case-Based Learning in Model Management Systems", *Decision Support Systems*, Vol.10(1993), pp.137-160.

[27] P.C. Ma, F.H. Murphy, and A. Stohr, "A Graphics Interface for Linear Programming", *Communications of the ACM*, Vol.32, No.8(Aug. 1989), pp.996-1012.

[28] Murphy, F.H., Stohr, E.A., and P.C. Ma, "Composition Rules for Building Linear Programming Models from Component Models", *Management Sciences*, Vol.38, No.7 (1992), pp.948-963

[29] F.H. Murphy, and E.A. Stohr, "An Intelligent System for Formulating Linear Programs", *Decision Support Systems*, Vol.2(1986), pp. 39-47.

[30] Newell, A., and H. Simon, *Human Problem Solving*, Englewood Cliffs, NJ: Prentice-Hall Inc., 1982.

[31] Tseng, S. F., "Diverse reasoning in automated model formulation", *Decision Support Systems*, Vol.20(1997), pp.357-83.

[32] Vellore, R. C., Vinze, A. S., and A. Sen, "Understanding the Case-Based Approaches to Model Formulation", *Proceedings of the Annual Meeting Decision Sciences Institute*(1990), pp.323-325.

[33] A. Vinze, A. Sen, and S.T. Liou, "AEROBA : A Blackboard Approach to Model Formulation", *Journal of Management Information Systems*, Vol.9, No.3(1993), pp.123-143.