R. BONCZEK
C. HOLSAPPLE
A. WHINSTON

Mathematical programming within the context of a generalized data base management system

RAIRO. Recherche opérationnelle, tome 12, nº 2 (1978), p. 117-139
<http://www.numdam.org/item?id=RO_1978__12_2_117_0>

R.A.I.R.O. Recherche opérationnelle/Operations Research
(vol. 12, n° 2, mai 1978, p. 117 à 139)

MATHEMATICAL PROGRAMMING WITHIN THE CONTEXT OF A GENERALIZED DATA BASE MANAGEMENT SYSTEM (*) (1)

by R. Bonczek (2), C. Holsapple (3) et A. Whinston (4)

Abstract. — *Aspects of mathematical programming are examined within the context oj a generalized data base management and query system. This system is general in the sense of its ability to support applications other than mathematical programming and its independence from the actual types of data values available. It is shown how data for mathematical programming may be organized into a network data structure which may be interrogated via non-procedural, English-like queries. Three methods are presented for interfacing math programming algorithms with this data base. Enhanced data manipulation facilities, particular to matrices and systems of equations, are also introduced. Finally a method is shown whereby programs may be integrated into a data structure, enhancing a user's ability to build alternative models for data analysis.*

INTRODUCTION

Data base management is a relatively new field which is currently the object of intense investigation. It involves the organization of data into some structure and the fitting of data with models and models with data in order to provide needed analyses. This presentation explores ways in which tools in the field of data management can offer assistance in the solution of mathematical programming problems. Designers of specialized math programming systems will observe a correspondence between some of the data base notions presented here and the ideas used in various math programming-related data facilities. The view adopted here pictures mathematical programming as a problem of data management, where the data relates to constraints and objectives. The models include linear, interger and non-linear application routines. As such we outline a fundamentally new perspective for viewing mathematical programming problems. It must be emphasized that we are not here concerned with mathematical programming algorithms per se, but with an effective tool for implementation and utilization of such algorithms, regardless of their special methods. Moreover, we suggest that development

(*) Manuscrit reçu août 1977.
(1) Research supported in part by National Science Foundation, Division of Computing Research, Grant Number MCS 76-24675.
(2) Assistant Professor of Management and Computer Sciences, Purdue University.
(3) Visiting Assistant Professor of Management, Purdue University.
(4) Professor of Management and Computer Sciences, Purdue University.

and implementation of math programming algorithms may very well profit from future advancements in the data base management field. This would permit development and implementation efforts to be concentrated on theoretical aspects and numerical techniques, removing the often onerous task of data management (e. g. over-laying, manipulation of specialized storage structures, etc.).

Within the context of data base management we introduce three basic methods for effecting the interface between data and mathematical programming routines. The first method consists of extracting appropriate data values from a data base and building them into a file that can be input to the desired application routine. In the second method, application programs are devised such that they utilize commands which enable direct access to the data base. The third method incorporates programs into the data base itself such that they may be executed by submission of non-procedural, English-like queries.

The specifics of these three methods are outlined within the framework of GPLAN (Generalized Planning System) which is under continuing development at Purdue University. The outstanding features of this data base management system may be summarized as follows: utilization of a network data base, selective retrieval of any configuration of data from a given network structure, and user interface with a data base and application routines via a non-procedural, English-like query language. As indicated in the ensuing discussion, GPLAN's extensive data management capabilities also provide a convenient tool for the evaluation of parametric changes and various modifications in problems formulation. Moreover it enables storage, retrieval and manipulation of not only objective and constraint coefficients, but also descriptive information about each coefficient such as its source and currency. During the formulation of large scale problems such information is vital for purposes of resolving conflicting constraints and rectifying the variety of errors and inconsistencies which almost inevitably occur. The GPLAN system provides a single, general mechanism for handling specially structured matrices. Finally, this system allows the data base to be used by other applications (e. g., simulations, statistical packages, etc.). A cursory overview of the GPLAN method of data management is the necessary precursor of a detailed examination of its applicability to problems of mathematical programming.

THE GENERALIZED PLANNING SYSTEM

GPLAN [1, 2] has two primary constituents: a data management system (GPLAN/DMS) and a query system (GPLAN/QS). The former enables a user to access a network data base with a procedural, programming language.

The latter allows retrieval of data and the execution of large application routines as a result of posing non-procedural, English-like queries; this feature permits data base utilization by non-programmers.

Within the scope of this presentation, a data base is considered to be defined by two attributes: a schema and a collection of data values which are logically organized in conformity with that schema. A schema is the specification of a logical structure; it is a blueprint of data base contents. Notice that we do not consider physical storage structures here, since all user requests of the data base are made in terms of its logical organization. The fundamental building blocks of a schema are data item types; for example, VARIABLE-ID, VARIABLE-DESCRIPTION, CONSTRAINT-ID, CONSTRAINT-DESCRIPTION, COEFFICIENT-VALUE, COEFFICIENT-SOURCE refer to *types* of data that we may desire to include in a data base. Each of these data item types represents many *occurrences* of data values of that type within the data base; the data item type VARIABLE-ID may have "X1" through "X100" as data value occurrences. The schema also specifies the nature of the relationships that each data item type has with other data item types.

There are two varieties of relationships among data item types: aggregation and association. Data item types may be aggregated into what are termed record types; for instance VARIABLE-ID and VARIABLE-DESCRIPTION may be aggregated to form the record type VARIABLE. This is illustrated in figure 1 *a*, where the record type is indicated by the rectangle labeled VARIABLE. A sample record occurrence of VARIABLE is "X1" and "AMOUNT OF RESOURCE I TO BE USED". Alternatively, record types (and therefore data item types) may be associated with each other by means of a set relation, as outlined in the CODASYL Data Base Task Group (DBTG) Report of 1971 [3]. The DBTG "set" concept should not be confused with the mathematical notion of a "set", for they are not related. A set is defined in terms of an owner record type and a member record type such that there is a one-to-many relationship between owner and member occurrences. That is, there may be many occurrences of the member record type associated with each occurrence of the owner record type; but for a particular set, a given member occurrence may be associated with no more than one occurrence of the owner record type. Consider the record types VARIABLE and COEFFICIENT, the latter being an aggregation of such data item types as COEFFICIENT-VALUE and COEFFICIENT-SOURCE. If we define the set HAS with VARIABLE as its owner record type and COEFFICIENT as its member record type, then we have indicated that there may be many coefficients associated with each variable; but a given coefficient cannot be associated with more than one variable. Pictorially a set is indicated by an arrow that points from the owner record type to the member record type (see *fig.* 1 *b*). Not only does a set furnish information about the relation among occurrences of owner and member record types, but it also permits the member

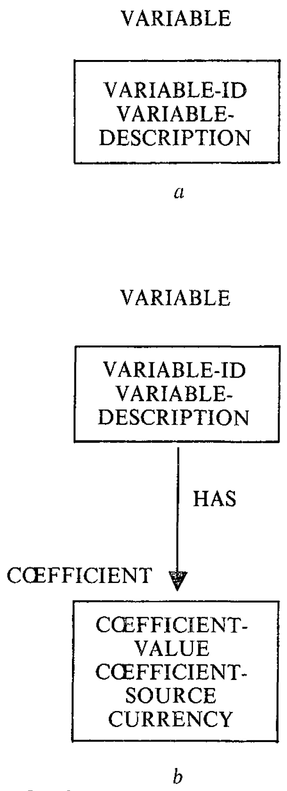

Figure 1. — Example of structural components of a data base.

occurrences associated with an owner occurrence to be logically ordered according to some criterion. For instance, given an owner occurrence of the set HAS, its member occurrences may be ordered in an ascending fashion according to the values of their data item type COEFFICIENT-VALUE.

Data base schemas are formally defined with a Data Description Language (DDL). Depending upon the conventions permitted by the DDL various kinds of data structure may be defined. If the DDL allows a record type to be declared as the owner of no more than one set and the member of at most one set, then the data base has a strictly linear structure. This is analogous to the "array" data structure permitted in a FORTRAN program. If the

DDL allows a record type to be the owner of more than one set, but the member of no more than one set, then we have a tree structure which is analogous to the data structure permitted by the Data Division of a COBOL program. If a record type is allowed to own many sets, as well as be a member of more than one set, then we have a network data structure. In keeping with the GPLAN philosophy of providing generality and flexibility, GPLAN/DMS and GPLAN/QS support network data structures. This provides a compact and powerful tool for representation of the types of data (and their relationships) involved in mathematical programming; a detailed example is provided in a later section.

The GPLAN/DMS furnishes a Data Manipulation Language (DML) which gives the user a means for storage, modification and extraction of data values for a particular data structure (as defined in terms of the DDL). This DML is utilized within the framework of a host language, e. g., FORTRAN, COBOL. Each command in the DML consists of a call to a FORTRAN subroutine which is a part of the data mangement system. Therefore, these DML subroutines essentially extend the FORTRAN language to give it complete data manipulation capability with respect to data organized into network structures. An important feature of this DML implementation is its high degree of machine independence; i. e., subject to a few minor modifications, this DML can be used on any machine that has a FORTRAN compiler and a random access mass storage facility.

Whereas GPLAN/DMS requires that a user write programs in a host language with the utilization of pertinent DML commands, GPLAN/QS does not require one to be a programmer in order to utilize the data base for purposes of display or execution of large application routines. The user needs merely to specify what is to be done; there is no statement of the procedures to be followed in order to accomplish the task. Examples of very simple commands are:

LIST COEFFICIENT-SOURCE FOR VARIABLE-ID = "X1" AND CONSTRAINT-ID = "R3";

LIST VARIABLE-ID AND CONSTRAINT-ID FOR COEFFICIENT-SOURCE = "TEST 1".

Upon receipt of such commands, the query system analyses the request, sets up the necessary DML commands, executes those commands and supplies the requested data values. The system is designed such that it permits the selective (or unconditional) retrieval of any data configuration. Moreover, it permits execution of application routines using any desired (and germane) data from the data base. The fundamental query syntax is

⟨ COMMAND ⟩ ⟨ RETRIEVAL CLAUSE ⟩ ⟨ CONDITIONAL CLAUSE ⟩

The command indicates which application routine is to be executed; in the queries above, LIST indicates that a report generator is to be executed. In the retrieval clause the user specifies what data are to be used for execution; this retrieval is dependent on conditions specified in the conditional clause.

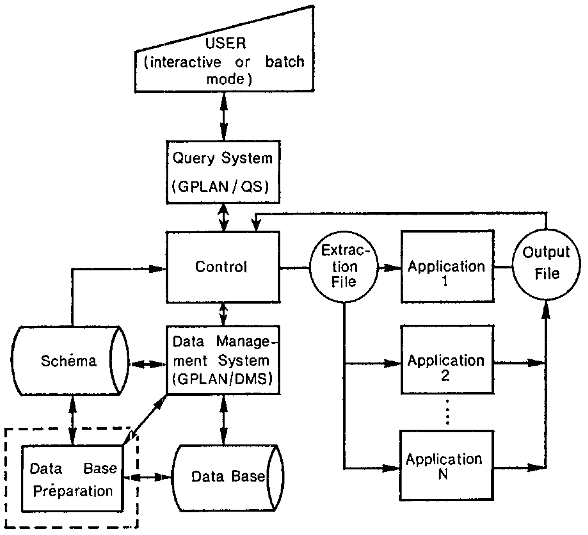

Figure 2. — GPLAN system.

A user of the query language is allowed to present arbitrarily complex retrieval clauses. Not only may this clause contain the names of data items to be retrieved, but arithmetic operations (using literals or data items) and both single and multivariate functions may also be introduced. The conditional clause is composed of a Boolean expression which may contain data item names, literals, arithmetic operators, relational operators, logical operators, single-variable functions and multivariate functions. The query language also permits the use of noise words, synonyms and various other cosmetic features for the convenience of the user.

A conceptual overview of the standard GPLAN system is portrayed in figure 2. The library of application routines is composed of two sections: standard routines and special routines. The standard library of applications consists of routines to generate reports and plots and to perform linear regressions, statistical analyses, and data modification. The library of special applications may include such routines as linear and non-linear optimization programs. In the succeeding sections we elaborate on the issue of interfacing such optimization routines with a network data base by means of the three methods alluded to in the introduction.

MATHEMATICAL PROGRAMMING WITHIN THE GPLAN CONTEXT

We examine this issue from the standpoint of providing flexibility and convenience, both to those who implement mathematical programming algorithms and to those make use of such implementations. In so doing, we utilize the distinctive GPLAN features of the network data base structure, the language for programmer interface with a data base (DML) and the query language that allows non-programmers to effectively use a data base and pertinent application routines. Not only does the GPLAN framework allow for the obvious, i. e., the solution of linear and non-linear optimization problems; it also addresses the following considerations, which may perhaps be more subtle, but are certainly of practical significance.

1. The resolution of erroneous formulations.
2. Treatment of coefficients which are themselves functions.
3. Situations wherein matrices contain common data.
4. Storage of sparse matrices.
5. Utilization of data to produce timely, non-routine reports other than the report furnished by a general mathematical programming routine.
6. Ability of a data base to support other varieties of application routines (e. g., simulations, regressions, etc.) in addition to mathematical programming.

The modus operandi for effective accomodation of each of these attributes is detailed in the course of the sections which follow; however, a brief elaboration of each is presently in order. With respect to the first point, when erroneous coefficients or improper formulation is suspected the ready availability of information with regard to coefficient sources and currency is important. This has obvious implications for the way in which data is organized and retrieved.

Concerning the second attribute, it is not uncommon that coefficients are the results of functions that have been evaluated on the basis of some other data. There may even be alternate functional forms (or alternate data sets for evaluating a function) which suggests the need for a facile method of interfacing desired functions with the math programming routines which depend upon them. Indeed a linear programming routine may be viewed as a function that requires (recurring) evaluation in order to execute a non-linear programming routine [4]. One technique for handling this situation involves an elimination of the distinction between data and function, with respect to data base construction; i. e., functions are treated as data and included in the data base structure. This point will be elucidated in a subsequent section.

Attribute number three is important for cases where there is interest in several matrices, which are not entirely distinct with respect to constraints and variables. For example we may be investigating a problem which has multiple plausible formulations, some pairs of which share constraints (and therefore variables). Care must be taken to assure that updates to constraints in one matrix are reflected in other matrices which share these constraints; this is not a trivial matter where large volumes of data are involved. A technique that can be used within the GPLAN framework allows us to store a contraint only once, while at the same time specifying that it is to be included in an arbitrary number of matrices. This avoidance of redundancy averts the potential for inconsistencies and storage inefficiencies. An extreme example of non-distinct matrices is the case of matricies with in matricies (e. g. linear programming problems solvable by the decomposition principle [12]).

Many real-world applications entail the utilization of sparse matrices. In the effort to avoid storing zeros, many schemes have been devised for packing (and unpacking) non-zero coefficients into arrays. Under the GPLAN concept, all matrices (sparse or otherwise) can be accomodated by a single simple logical structure, which realizes substantial storage savings if a matrix happens to be sparse. Only non-zero coefficients need to be stored; and storage space is neither used nor even allocated for zero coefficients. This provides a single mechanism for storing specially structured matricies of all kinds; thus a special storage and access method is not required for each type of matrix structure.

Presumably managerial decisions are not based solely upon the output of mathematical programming routines. The fifth consideration indicates the need for a facility to generate other reports from a data base and frequently these are non-standard in terms of the types and configurations of data that are retrieved. The GPLAN query system allows the selective retrieval of any configuration of data as a result of typing an English-like, non-procedural query at a computer terminal. This obviates the crude necessity of writing a program every time a new type of report is needed.

This concept can be extended to include not only retrieval, but also the execution of large application routines that are not of the mathematical programming variety. Futhermore such executions can be accomplished through the query language. The result is a situation wherein a network data base can support a broad spectrum of analyses for both programmers and non-programming users.

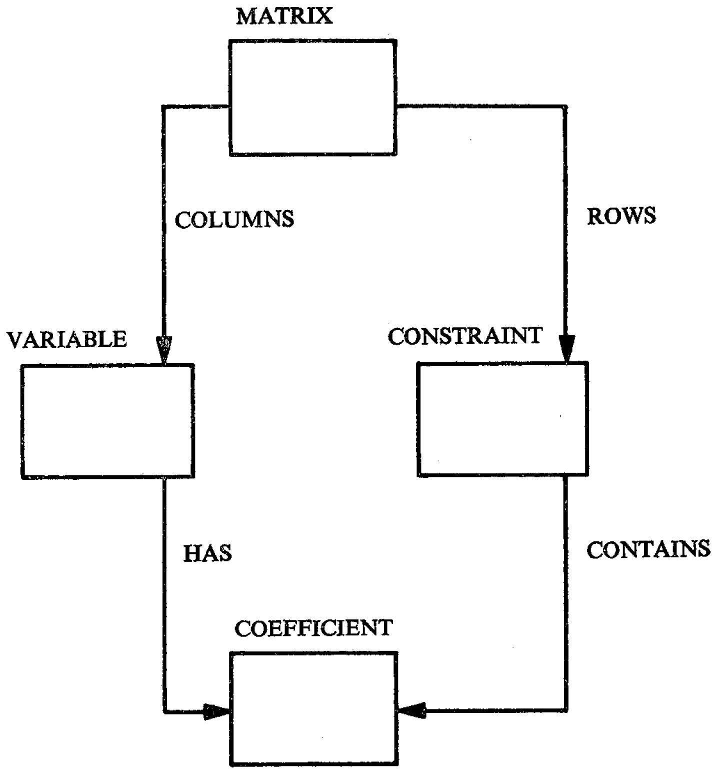

Figure 3. — Fundamental network structure of matrices.

A NETWORK DATA STRUCTURE FOR LINEAR PROGRAMMING PROBLEMS

In order to illustrate the distinctive features of data storage based on a network structure, we outline a particular structure for storage of data relating to linear programming problems. Aspects deserving special emphasis are the generality, compactness, convenience, and organized nature of this means for specifying and communicating about information needs. Figure 3 depicts the fundamental logical structure of information contained within matrices. Recall that each rectangle represents a record type and each arrow denotes a set. The record type MATRIX is the owner of two set types: COLUMNS and ROWS. Since a set defines a one-to-many relationship between an occurrence of the owner record type and possibly many occurrences of the member record type, the set COLUMNS associates occurrences of the VARIABLE record type with an occurrence of MATRIX. Similarly, ROWS indicates that there are many constraints associated with a MATRIX.

Within a matrix there is a many-to-many relation between variables and constraints. That is, a variable is contained in many constraints, but each constraint contains many variables. This kind of relation, by definition, cannot be represented by a single set. However, it can be represented by two sets (HAS and CONTAINS) which have a common record type COEFFICIENT as member. So an occurrence of the VARIABLE record type HAS many COEFFICIENTS associated with it. Similarly each constraint CONTAINS many coefficients. Recall that in the definition of a set, it was stated that a member occurrence of a particular set can be associated with no more than one occurrence of that set's owner record type. Thus a given occurrence of COEFFICIENT cannot be associated with more than one variable (via the HAS set) and one constraint (via the CONTAINS set). In other words, if we are given an occurrence of COEFFICIENT we immediately know the variable and constraint with which it is associated. Conversely, if we are given an occurrence of VARIABLE and an occurrence of CONSTRAINT then we can immediately determine the coefficient which they have in common. (Notice that this structure allows them to have more than one occurrence of COEFFICIENT in common; this handles the contingency where there is uncertainty as to the correct value of a coefficient, so that alternative values may be stored for purposes of further analysis.) When a coefficient is zero, no occurrence of COEFFICIENT is created so that the corresponding variable and constraint can have no coefficient in common.

Observe that the logical structure of figure 3 does not permit two matrices to have a variable or constraint in common. By the definition of a set, COLUMNS does not allow an occurrence of VARIABLE to be owned by more than one occurrence of MATRIX; the same holds for ROWS. One way to treat this problem is to allow redundant occurrences of the record types VARIABLE and CONSTRAINT; but this poses problems for the maintenance of data base integrity. However, we can extend the present logical structure to handle the case of many-to-many relationships between

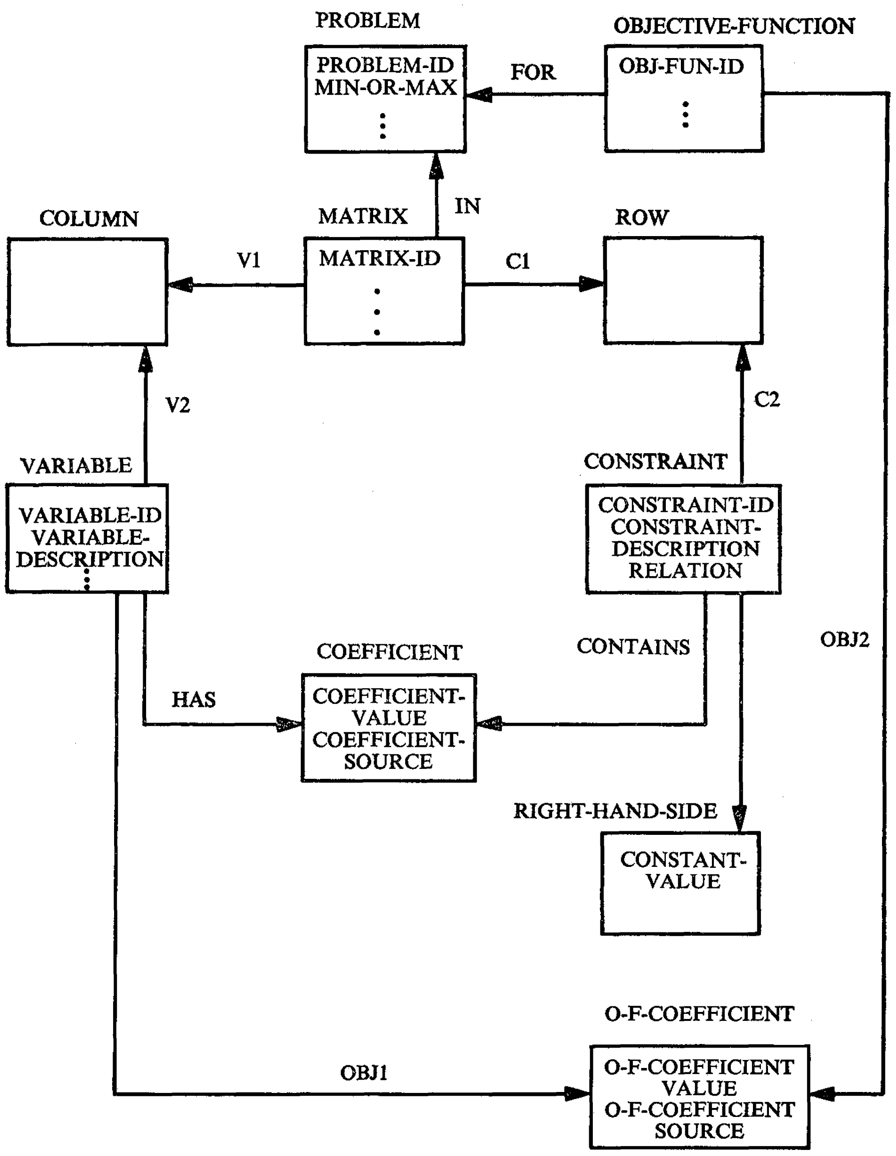

Figure 4. — Extended logical data structure

MATRIX and VARIABLE and between MATRIX and CONSTRAINT without introducing redundancy. The structure is shown in figure 4 and is analogous to that discussed for the many-to-many relation between VARIABLE and COEFFICIENT. Figure 4 also portrays the means for structuring information about the objective function and shows how matricies and objective functions can be associated with one another to form a PROBLEM. Figure 5 shows an example of this at the record occurrence level. Each circle depicts an occurrence of the record type indicated in the

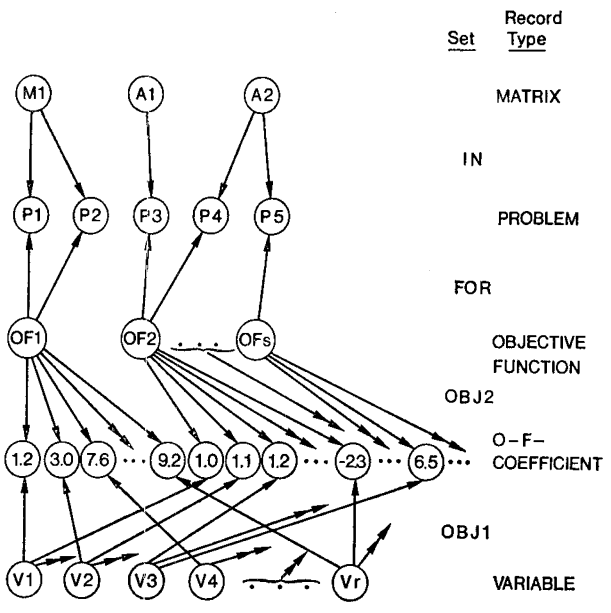

Figure 5.
Problem formulat on using the logical data structure of figure 4.

right margin. Arrows emanating from an occurrence point to other occurrences that it owns by virtue of the set specified opposite the arrows in the right margin. For instance, "M1", "A1" and "A2" are occurrences of the record type MATRIX. Recalling that a set defines a one-to-many relationship between an occurrence of its owner record type and occurrences of its member record type, we observe that "M1" is associated with two occurrences of the record type PROBLEM (i. e. "P1" and "P2") via the set named IN. It can also be seen how the sets IN and FOR allow us to denote which matrix and which objective function constitute a problem. The problem identified by "P4" consists of the objective function "OF2" and the matrix "A2", but "A2" also participates in problem "P5" and "OF2" participates in problem "P3". In the lower portion of figure 5, it can be seen how the sets OBJ1 and OBJ2 are used to show which variables are associated with each objective function. Note that variable "V2" is found in both "OF1" and

"OF2"; its coefficient value is 3.0 for the former and 1.1 for the latter. For diagrammatic clarity, not all record occurrences are shown; double headed arrows are used to indicate ownership of several members which are not displayed.

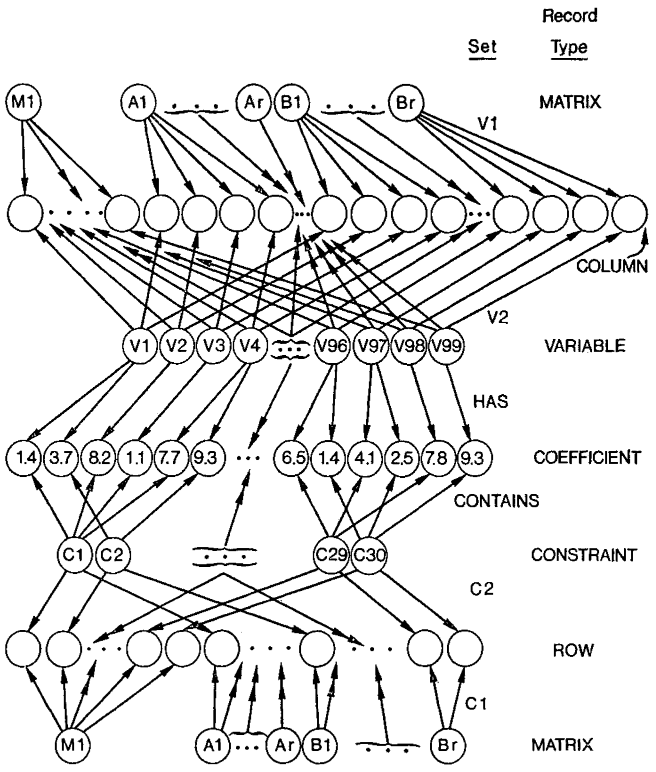

Figure 6. — Record occurrence representation non-distinct matrices.

Once again utilizing the logical data structure of figure 4, a record occurrence representation of non-distinct matricies is presented in figure 6. In particular the example is that of matricies (A1, ..., Ar, B1, ..., Br) within another matrix (M1). Thus the logical structure of figure 4 provides the means for representing matricies solvable by the decomposition algorithm. The reader should observe that the example is easily extendable, within the same structural framework, to accomodate the situation wherein the matricies B1, ..., Br may themselves be of the decomposition form, and so forth. Referring to figure 6 we see that the master matrix "M1" includes all variables, whereas the other matricies include only a few. Observe that constraint "C1" contains the variables "V1", "V2", "V3" and "V4" with coefficients "1.4", "8.2", "1.1" and "7.7" respectively; all other variables have zero coefficients (for which no storage is allocated) with respect to this constraint. Constraint "C2" has only two non-zero variables (i. e. "V1" and "V4" with coefficients "3.7" and "9.3"). Finally notice that constraints are associated with appropriate matricies via the sets "C2" and "C1". For pictorial convenience the occurrences of MATRIX shown at the bottom of the figure are repeats of those shown at the top, but such occurrences are not repeated in the actual data base. Note that "M1" encompasses all constraints, whereas the other matricies include only some of the constraints.

THE INTERFACE OF PROGRAMS AND DATA BASE

The first two methods described here are presently fully available to GPLAN users; the third method is the object of current extensions to GPLAN. The first method examined here makes use of existing programs that have been devised independently of data base management considerations. Each such program requires that input data be in the particular format that it can use. Thus if the data that a linear program (LP) requires is stored according to a data base structure like that of figure 4, then that data must be extracted from the data base and written onto a file in the format amenable to the LP routine. An obvious way to accomplish this is to write a program that uses DML commands to find and extract pertinent data from the data base. Output statements of the host language are used to transfer extracted data to a sequential file that is formatted for use by the LP routine. This DML extraction routine can serve to interface the LP routine with any LP problem that resides in the data base.

Incorporation of the extraction and LP routines into the query system's library of special applications allows us to submit queries such as

```
RUN LP FOR MATRIX-ID = "M27" AND OBJ-FUN-ID = "OF1",
            RUN LP FOR PROBLEM ID = "P7".
```

As previously mentioned, the system also furnishes flexible and broad retrieval capabilities, execution of statistical and regression packages from the standard library and the ability to support a special library that may contain routines ranging from special report generators to large scale simulations.

A second method of application-data base interface makes use of DML commands within the mathematical programming routine proper. Thus, the entire formulation of the problem need not reside in the program's arrays, nor is there any need for overlays; instead, DML is used to withdraw and return particular coefficients (or groups of coefficients) to the data base as needed by the algorithm. The standard DML can do little more than store, find and extract data values. But what is a pivot operation if not a type of data manipulation. It is therefore as a logical extension to the DML, that we introduce a group of extended DML commands applicable to mathematical programming. In so doing, this second method of application-data base interface is greatly enhanced, for the programmer can issue DML commands to perform such tasks as pivoting, finding inverses and determinants, and even the solution of a linear system. Such commands are predicated upon the natural network data structure for LP problems as presented in figure 3 and 4.

For instance, the pivot command has arguments indicating the matrix, column and row for which the pivot operation is to be executed. This command is implemented by invoking the appropriate DML routines and performing appropriate arithmetic operations. The DML routines traverse the network structure (implemented as a doubly linked list [1]), access coefficients to be operated upon, and store the modified coefficients back into the data structure. No overlaying is required in the implementation of the pivot command, since it is handled automatically by the DML thereby providing a virtual system. The pivot command may be utilized in the development of linear programming code, leading to a single command which solves a linear system. This LP command could be based on any of a variety of LP algorithms. Indeed it may be desirable to devise several LP commands. For example, LP1 may be based on the simplex or revised simplex method and LP2 could be an implementation of the decomposition algorithm [12]. This LP2 would repeatedly utilize the LP1 command, which repeatedly utilizes the pivot command. In any case it should be noted that programs using this second interface method are, of course, amenable to integration into the special application library of the query system.

The third method for interfacing applications with the data base consists of treating applications as part of the data base, i. e., application routines are accounted for in the logical structure of the data base. In order to elucidate the concepts involved, we make use of the following example drawn from the field of water pollution control. Previous work employing GPLAN in the pollution control context is described in [5, 10]. This work has been largely predicated upon continuing efforts to develop tools to assist planners in their attempts to comply with the Federal Water Pollution Control Act Amendments of 1972.

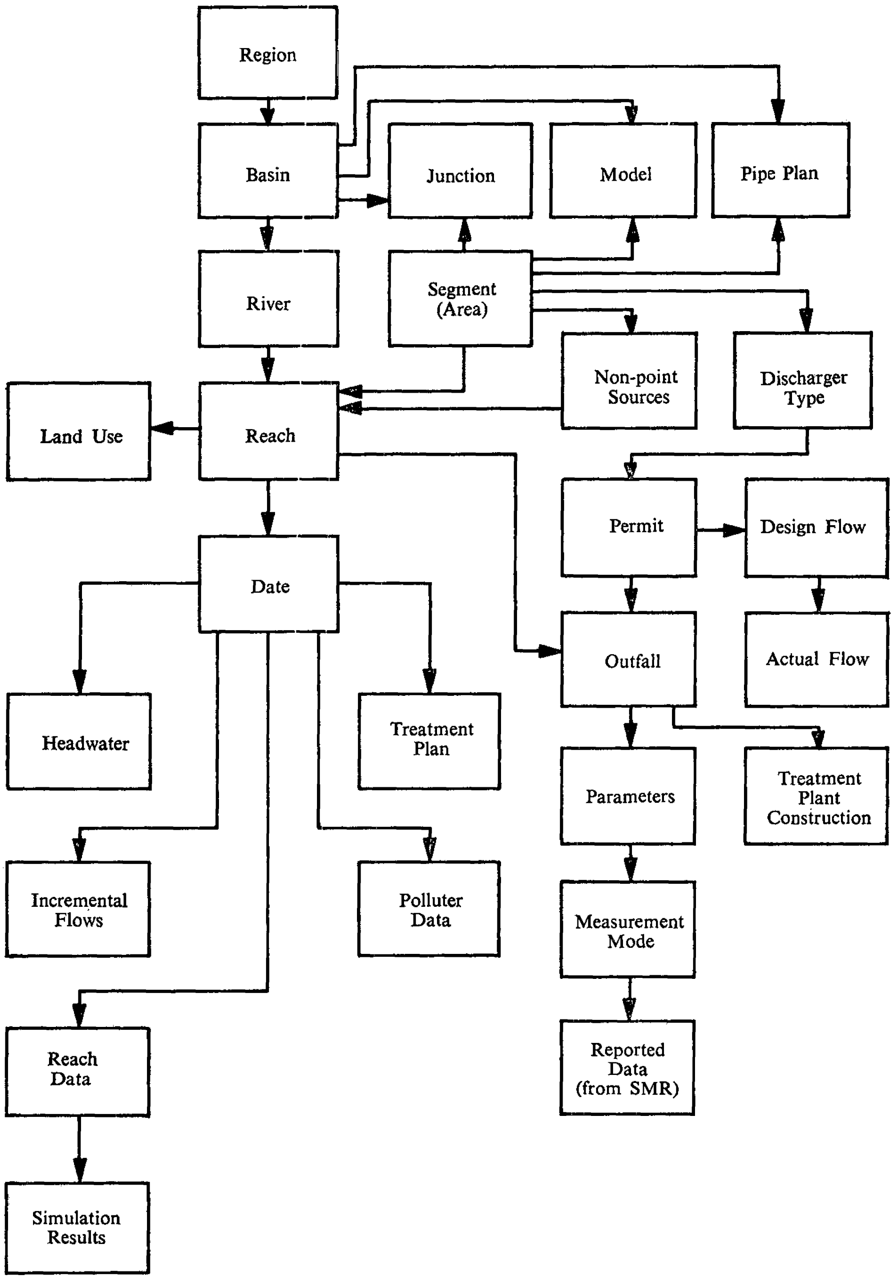

Figure 7. — Overwiew of logical structure.

Briefly, each river within a basin is decomposed into reaches, each of which describes a portion of the river in which certain state characteristics are relatively invariant and in which there is at most one pollution source (outfall) or incremental flow. A group of contiguous reaches which exhibit water quality impairment may be declared to be a designated area for local planning purposes. Water quality within a given reach depends not only upon that reach's state characteristics, polluter activities and pollution treatments; but also upon the state characteristics, polluter activities and pollution treatments of all upstream reaches. One objective, then, is to ascertain a basin-wide treatment plan which: *a*) satisfies legal water quality standards; *b*) is amenable to implementation from political, managerial and technical standpoints, and *c*) minimizes the basin-wide cost of treatment. Other considerations involve the monitoring of polluter compliance with the conditions of their discharge permits, the management of treatment facilities construction, and area-wide coordination of water pollution control programs with other local activities such as land use planning.

Figure 7 displays a simplified version of a logical data base structure for area-wide water quality planning; the data item types are too numerous to include here. As can be seen, this data base includes information about basin-wide piping plans, state characteristics within each reach, flow characteristics of the basin, temperature and effluent data for the reaches, water quality goals and existing treatment plans. Provision is also made for permit compliance monitoring and other planning data. More detailed descriptions of these record types and their data item types may be found in [9].

Suppose that the cost C_g^T of controlling thermal effluent by means of a cooling tower in reach g is known and stored as part of the HEADWATER record type. Then the overall cost of controlling temperature in the river can be represented as

$$C_T = \sum_g (C_g^T)(s_g)(T_g^* - T_E)(F_{g4}),$$

where T_g^* is the temperature of the thermal effluent in reach g (stored in the record type POLLUTER DATA); T_E is the equilibrium temperature of the basin (stored in MODEL record type); F_{g4} is the thermal effluent entering reach G from upstream (part of the REACH DATA record type); and s_g is the percent of thermal effluent removed at the cooling tower in reach g. The decision variable is s_g. In order to represent the function C_T in the data base, we declare a record type for C_T (see *fig.* 8) which is associated with the record types MODEL, REACH DATA, HEADWATER and POLLUTER DATA via set relationships named for the data item values to be extracted from these record types. Thus we store programs (C_T is a very simple program) as occurrences of record types; e. g., if there were several ways to compute C_T,

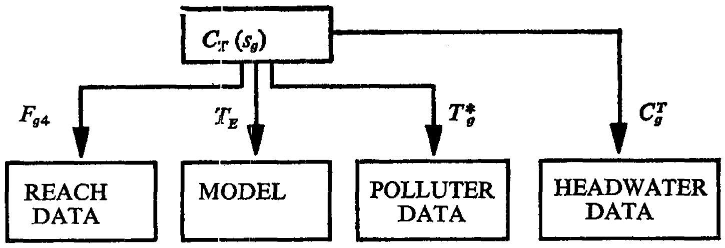

Figure 8. — Cooling tower cost.

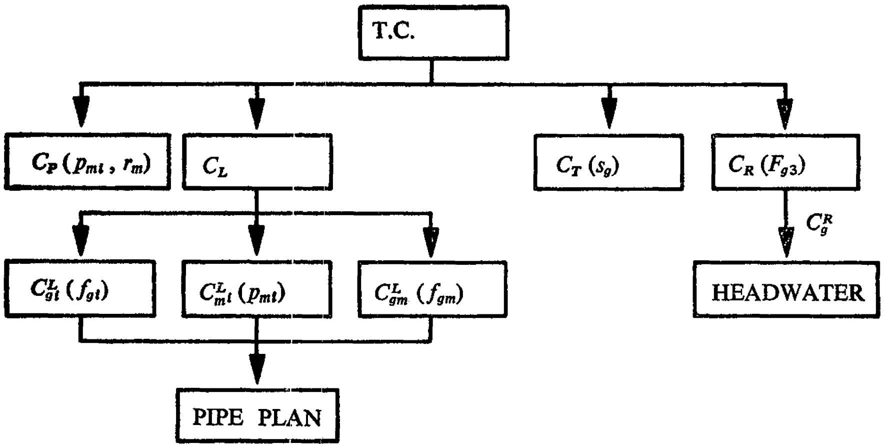

Figure 9. — Total cost function.

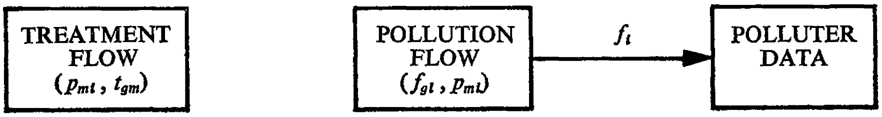
TREATMENT FLOW (p_{mt}, t_{gm})
POLLUTION FLOW (f_{gt}, p_{mt})
f_t
POLLUTER DATA

Figure 10. — Flow balance equations.

we would have one occurrence of the record type C_T for each functional form.

Similarly, piping costs (C_L), treatment plant costs (C_P), and flow augmentation and reservoir costs (C_R) can be computed from stored data base values and dependent variables, and the corresponding record types can be constructed. These three, in addition to C_T, are combined to compute a total pollution control cost:

$$TC = C_L + C_P + C_R + C_T.$$

The decision variables for TC are simply those for each of the component costs. Observe that the component costs are effectively subroutines of the total cost computation. Figure 9 uses program record types portraying this relationship. Thus we have an example of the kind of structural information that can be indicated by the use of program record types. The data structure reflects the subroutine flow and can be queried to furnish documentation about the operation of stored programs.

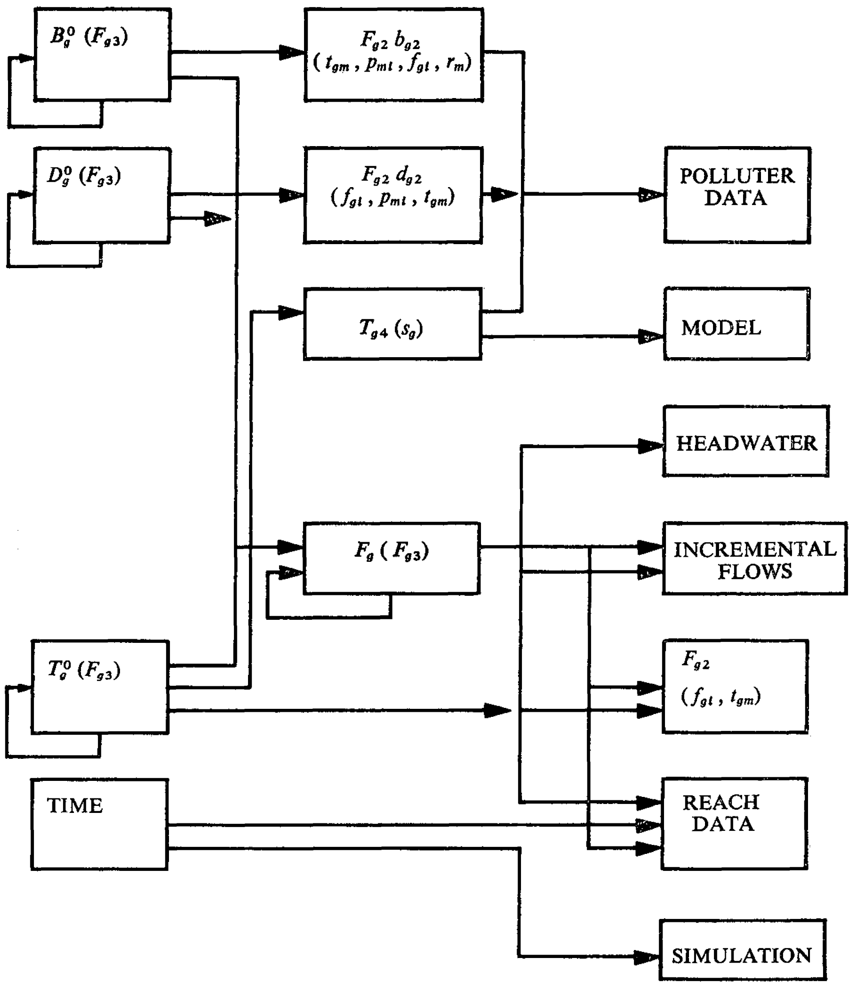

Figure 11. — Calculation of B_g^0, D_g^0, T_g^0, and TIME.

The user of a data base system incorporating such program record types can compute the cost of treatment for a given plan of treatment by specifying

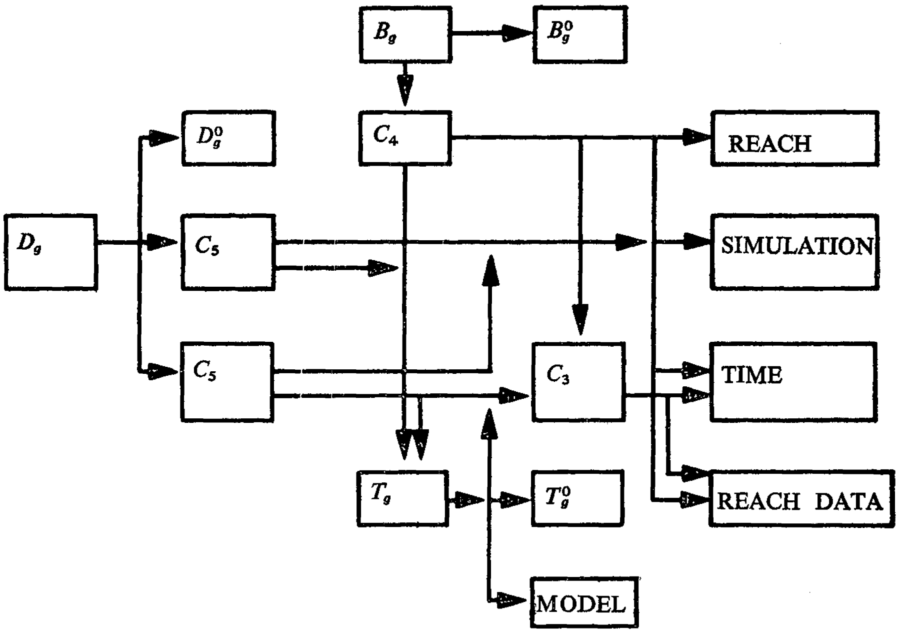

Figure 12. — Calculation of Bg, Dg, and Tg.

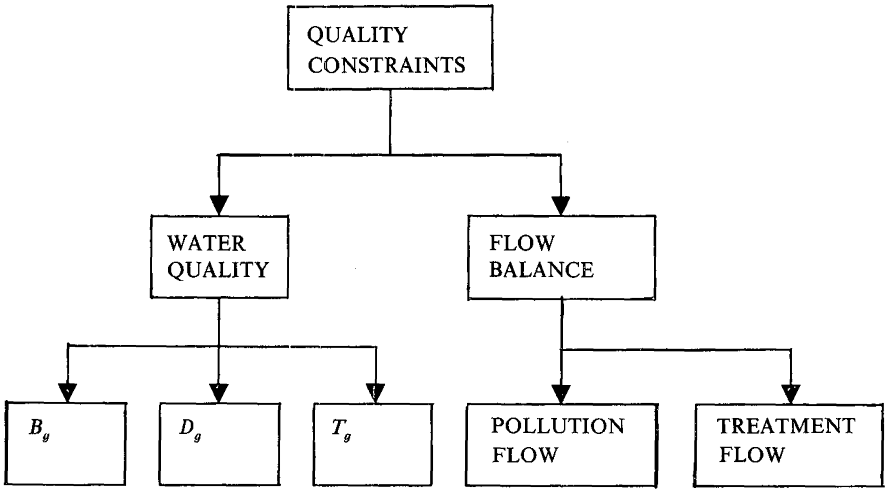

Figure 13. — Water quality constraints

values for the decision variables. Furthermore, we now examine how programs can be used to determine a treatment plan and therefore a total cost. A method for solving the problem:

Minimize: treatment costs;

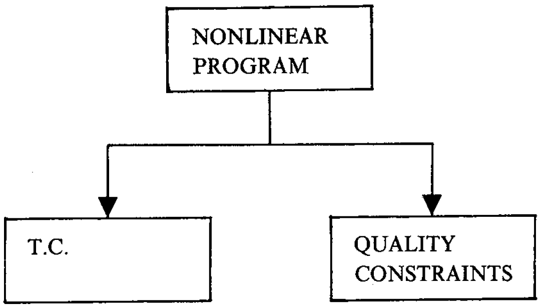

Figure 14. — Optimization model

Subject to: quality goals satisfied;
is outlined in [10]. The total cost function described above becomes the objective function of a non-linear programming model. The decision variables of the total cost function are the primal variables in the minimization. Representation of the flow balance equations for pollution and treatments is shown in figure 10. The water quality goals, although complicated can also be represented through the use of program record types; see figures 11 through 13. Notice that results of the computation of B_g^0, D_g^0, T_g^0 and TIME (*fig.* 11) are used in the calculation of B_g (biochemical oxygen demand), D_g (dissolved oxygen deficit) and T_g (thermal effluent temperature) as shown in figure 12. Flow balance constraints are combined with water quality constraints in figure 13. Finally, the entire optimization package can be represented as a program record type (*fig.* 14). This optimization program can be executed to determine an optimal treatment plan and can be dissected to indicate the assumptions behind the execution of the model. Observe that the quality constraints can be evaluated individually with a particular set of dependent variables, in order to determine the effect of a particular treatment plan.

The non-linear programming subroutine is available for use on two levels. It can be used as a stand-alone program, where the objective function and constraints are specified by the user at execution. As evidenced by the example just presented, it can also function as a participant in a predefined package. In the instance where a non-linear algorithm utilizes a linear programming algorithm, then the linear routine can either be used in a stand-alone capacity or in conjunction with the non-linear routine. Perhaps the most outstanding feature of this treatment of programs, beyond the facile query capability, is the flexibility that it permits with respect to combining programs into models. The issue of automatic interface of programs with a data base is examined in [11].

CONCLUSION

In this paper, we have examined numerous aspects of mathematical programming within the context of a generalized data base management and query system (GPLAN). Special attention is devoted to the incorporation of mathematical programming into a data base system that is general, in the sense of its ability to support other applications (e. g., simulations, statistical analyses, etc.) and its independence from the actual kinds of data values available. Organization of data values according to a network logical structure was delineated. Three methods of data base-application interface were investigated. Enhanced data manipulation facilities (particular to matrices and systems of equations) were introduced. Also introduced was the concept of treating programs as data, which has broad implications for the user's ability to build models. This was illustrated with an application drawn from the field of water quality planning. The objective has been to demonstrate contributions which the data management field can make to the development and implementation of mathematical programming algorithms. It is suggested that future advances in data management technology may further enhance these contributions. A topic of future investigations is that of the execution of math programming routines within a distributed data base environment. Especially important in this regard is the decomposition form of linear programming. As a final note, it is important to observe that the concepts presented are not specific to water quality planning, but are applicable to any situation where data is analyzed via mathematical, statistical or simulation programs.

REFERENCES

1. HASEMAN, W. D. and A. B. WHINSTON, *Introduction to Data Management*, Richard D. Irwin, 1977.
2. R.H. BONCZEK, C. W. HOLSAPPLE and A. B. WHINSTON, *Extensions and Corrections for the CODASYL Approach to Data Base Management*, International Journal of Information Systems, Vol. 2, 1976.
3. CODASYL: Data Base Task Group Report, A.C.M., April 1971.
4. G. GRAVES, D. PINGRY and A. WHINSTON, *Water Quality Control: Nonlinear Programming Algorithm*, Revue Française d'Automatique, Informatique et Recherche Operationnelle, October 1972.
5. W. D. HASEMAN, C. W. HOLSAPPLE and A. B. WHINSTON, *Implementation of a Large Scale Water Quality Data Management System*, Socio Economic Planning Sciences, vol. 10, March 1976.
6. W. D. HASEMAN, C. W. HOLSAPPLE and A. B. WHINSTON, *O. R. Data Base Interface — An Application to Pollution Control*, Computers and Operations Research, January 1976.

7. W. D. Haseman, A. Z. Lieberman and A. B. Whinston, *Water Quality Management and Information Systems*, Journal of Hydraulics Div. A.S.C.E., March 1975
8. W. D. Haseman and A. B. Whinston, *A Data Base for Nonprogrammers*, Datamation, May 1975.
9. C. W. Holsapple and A. B. Whinston, *A Decision Support System for Area-wide Water Quality Planning*, Socio Economic Planning Sciences Vol, 10, 1976.
10. D. E. Pingry and A. B. Whinston, *A Multigoal Water Quality Planning Model*, Journal of Environmental Engineering, Div. A.S.C.E., December 1973.
11. W. D. Haseman and A. B. Whinston, *Automatic Program Interface*, The Computer Journal November 1977.
12. G. B. Dantzig and P. Wolfe, *Decomposition Principle for Linear Programs*, Operations Research, Vol. 8, 1960.