Some large-scale linear and especially mixed-integer programming problems, and the underlying practical decision-making situations, have so far been solved with only limited success. A new control language for IBM's system MPSX-MIP/370 permits recursive use of the basic system and easy access to its elements, and therefore appears to offer great potential for new advances.

The paper first describes the facilities of the language, called the Extended Control Language, and the interfaces to the system and gives a number of representative illustrative uses. It then considers a number of basic applications of the system and possible heuristic and algorithmic approaches to difficult problems, often very large problems with structure, which may now become more easily solvable or tractable for the first time.

The Extended Control Language of MPSX/370 and possible applications

by L. Slate and K. Spielberg

The application of mathematical programming to *large-scale practical problems* has been hampered by the absence of reliable tools other than linear programming (with the possible exception of such techniques as simulation, project management, econometric forecasting, etc.). There exists a wealth of know-how, algorithms, and heuristics, but their realization in practice has been lagging, being largely limited to small-scale or highly specialized problems.

The Extended Control Language of IBM's MPSX/370 (Mathematical Programming System Extended/370)[1–5] appears to offer potential for substantial progress beyond what has so far been attainable. It is a PL/I-based language that permits access to all of the individual procedures of the system as well as to most of the working arrays of a particular linear programming problem.

A user can write PL/I programs that access the system repeatedly (e.g., iteratively), or that modify data and procedures after preset interruptions, followed by resumption of execution. This ability should permit substantial improvement in the solution of selected problems in nonlinear and mixed-integer programming.

Practical problems that lead naturally to sequences of linear programs (such as dynamic management science problems) can be set up and solved in succession. At the same time, well-known algorithms or heuristics can be applied to decompose very large models into smaller fragments, to be solved in an orderly and tractable fashion.

In this paper, we give an overview of the systems features and capabilities, followed by a sketch of possible applications. Our intent is to give an idea of some of the possibilities. We cannot hope to give a complete survey, nor can we attempt to give a complete list of references. If we overstress our own past work and that of close associates, it is only because we are most familiar with it.

Facilities of the Extended Control Language

MPSX/370 can be invoked by means of a PL/I program, so that strictly speaking, the Extended Control Language[4, 5] is PL/I. Certain MPSX/370 facilities have been designed to aid in this process, and more broadly speaking, PL/I augmented by these facilities is referred to as the Extended Control Language.

interface to the communication region

Much of the control of the linear programming and mixed-integer programming runs is provided by the communication region of MPSX/370. It consists of a collection of PL/I variables, referred to by privileged names (all starting with the character X), e.g.,

Datanames:

XDATA	name of data
XPBNAME	name of problem

Parameters:

XTOLV	feasibility tolerance
XFREQINV	frequency of inversion
XDOINV	address for inversion demand (see the discussion on setting demands)
XMXDROP	bound on IP (integer programming) objective function
XMXJ	number of integer variables

The communication region cells are set by assignment statements, e.g.,

```
XDATA='ANDELU'
XPBNAME='PB15'
XMXDROP=5000
```

The communication region cells have default values assigned (declared and initialized) by the system procedure DPLINIT, invoked by the PL/I statement %INCLUDE DPLINIT.

The procedures of MPSX/370, which have retained their names for the most part, can be invoked by PL/I call statements, e.g.:

```
CALL CONVERT;
CALL SETUP;
CALL RESTORE('NAME', 'BASI');
CALL OPTIMIZE; etc.
```

Arguments are passed in argument lists and through the communication region variables.

setting of demands

"Demands" are activated by the system when certain conditions arise during a run. The action to be taken when the condition occurs is defined through the PL/I "ON CONDITION" facility. A typical definition follows:

```
ON CONDITION(XDONFS) BEGIN;
CALL STATUS;
CALL SOLUTION;
STOP;
END;
```

The first statement names a condition and begins the block that defines the actions to be taken when the condition occurs. In this example those actions are execution of the STATUS and SOLUTION procedures, followed by termination of the PL/I program. XDONFS is a communication region variable associated with the system's recognition of problem infeasibility.

Some conditions, such as problem infeasibility or major error, are defined by the system; others, such as attainment of a particular iteration count, are defined by the user.

Names and descriptions of some of the demands are as follows:

Name	*Description*
XDONFS	problem has no feasible solution
XDOUNB	solution is unbounded
XMAJERR	major error is detected
XDOFREQ1	iteration count is a multiple of XFREQ1
XMXDOFRN	node number is a multiple of XMXFRN

The last two demands are examples of conditions for which the user specifies the controlling parameter (e.g., XFREQ1). Default implementation of demands is provided by DPLINIT.

high-level interface with data and results

In PL/I, data are stored in generalized arrays called structures. A structure is created by a DECLARE (DCL) statement. It can have several levels of detail, which can be explicity declared and then referenced. For our purposes, we need not go into a careful description of structures; the reader can get the flavor from the examples.

generation of data

One basic use of PL/I structures is the creation of input or revised data in a user (PL/I) program. For example, the following structure:

```
DCL 1 STCOL(10),
    2 IND     CHAR(2),
    2 NAME1 CHAR(8),
    2 NAME2 CHAR(8),
    2 VALUE DEC FLOAT(6);
```

can be used to represent 10 coefficients of a COL section input deck for CONVERT. The structure STCOL needs to be filled with data by insertion (from a PL/I program) of the appropriate indicators, names, and values.

Also, the entire input deck for an MPSX/370 run can be placed into one overall structure (see pages 197 and 198 of Reference 5 for a complete example). What is important for us is the potential usefulness of a main-storage resident PL/I structure. Many problems exist in which the data for the linear programming model, especially in the case of cost coefficients, are complicated functions of real-life situations (e.g., they may have to be computed from rate tables, subject to other constraints such as union rules, etc.). In such situations, one may have a need for the computation of the coefficients (from parameters in case studies, for instance) in PL/I procedures, and subsequent insertion in the appropriate slots of the overall input structure.

Once the structure is established, say in STRCT1, it can be invoked from certain MPSX/370 procedures via the argument list by means of the keyword 'STRUCTURE', e.g., CALL CONVERT ('STRUCTURE', STRCT1).

access to system data

A number of MPSX/370 procedures, such as SOLUTION, RANGE, TRANROW, and TRANCOL, have output that the user can request to be stored in PL/I structures.

SOLUTION and RANGE output can be examined and analyzed, and possibly altered, and the subsequent algorithmic procedure can then be modified as a result.

TRANROW and TRANCOL permit the retrieval of portions of the *initial* or *current* (i.e., *updated*) simplex tableau. This can, for example, be used to observe the effect on the current solution of

introducing unit changes for the nonbasic variables, thus permitting sensitivity analysis involving the entire updated tableau (or sections thereof). Algorithmic uses of such facilities might be the generation of "surrogate constraints" or "Benders inequalities" or "Gomory cuts" in integer programming (e.g., see References 6 and 7).

Whenever the entire matrix is involved, there must be the possibility of selective output. MPSX/370 uses "selection lists" and "code parameters" to limit the output in various ways.

Selection lists

Selection lists are used as arguments of systems procedures such as the ones listed previously. They are used to define a subset of the rows and/or columns of the linear programming matrix which are to be used currently. A selection list consists of keywords such as 'RMASKS', which precede sets of masks (each being eight characters in length) to identify those names (identifiers of rows, for instance) that are of interest (are to be selected). For example, the list:

'RMASKS', 'MASK1', 'X*******', 'CMASKS', 'YY******', leads to the selection of:

- All rows specified by MASK1.
- All rows whose names start with the character X.
- All columns whose names start with the characters YY.

In the above, the asterisks represent any character.

The Extended Control Language also provides a special procedure, SELIST, to aid in the automatic construction of selection lists that are to correspond to a current algorithmic situation. In the following example, SELIST is used to construct the masks of selection lists that correspond to the basis.

```
DCL     BNAME (XM) CHAR (8);
DCL     ISIZE BIN FIXED (31);
DCL     C(3) CHAR (8) INITIAL ('RMASKS', 'MASK1', ' ');
DCL     TAB(XM,XJ);
DCL     1 STR4,
          2 BRHS(1) DEC FLOAT (6),
          2 TABLEAU(1,XJ) DEC FLOAT(6),
  CALL SELIST ('BASIS', 'NAME', BNAME, 'SIZE', ISIZE);
PUT     DATA(BNAME) SKIP;
ILP:    DO I=1 TO XM;
        C((2)=BNAME (I)
        CALL TRANROW('STRUCTURE', STR4, 'NONAME',
                          'SELIST', C);
        PUT LIST ('VALUE OF I IS', I);
        TAB(I,*)=TABLEAU(1,*);
        END;
```

The example should be reasonably clear even without PL/I expertise on the reader's part.

- The call to SELIST via keywords 'BASIS', 'NAME', 'SIZE', causes the program to place the names of the basic variables into the character array BNAME and the size of the basis index set (which we know to be equal to XM, the number of rows of the linear programming (LP) model) into ISIZE.
- At a typical iteration of the loop ILP (which is executed XM times) the name of the *i*th basic variable is placed into the second position of the character vector C (i.e., in place of MASK1 . . .).
- As a consequence, the call to TRANROW, via keywords 'STRUCTURE', 'NONAME', and 'SELIST', causes the coefficient values of row *i* (and *not* their names) to be placed into array TABLEAU(1,XJ).
- If space were tight, one would not accumulate the coefficient matrix in an array TAB(I,*), but might write it out, row per row, on a disk.
- The overall intent of the example might be to store the initial matrix in row form, or to work on the rows of the updated tableau during execution of a linear or a mixed-integer program.
- Note: The *program* SELIST should not be confused with the *keyword* SELIST.

Code parameters

Just as selection lists can be used to limit the portion of the matrix to be examined, code parameters ('RSECTION', 'CSECTION') can be employed to limit the types of information to be placed into the SOLUTION or RANGE report by the system. For details see the program reference manual.[5]

Algorithmic tools and internal data

Most of the facilities described at the beginning of the previous section deal with accessing the system on a fairly high level (control language level). The use of SELIST was somewhat of an exception. Here we first discuss the use of *internal system subroutines* and the possibly necessary access to *internal computational data*.

internal system subroutines

The following internal subroutines of MPSX/370 are callable from a user subroutine in the Extended Control Language:[3,4]

SELIST	PRICED1
INVALUE	FTRANL1
GETVEC1	FTRANU1

FIXVEC1	BTRANL1
POSTMUL	BTRANU1
PREMUL	ELIMN1
PRICEP1	CHUZR1

The reader who is familiar with the various steps of the simplex method will have no difficulty in identifying these subroutines with functional transformations of the algorithm. What one has to learn from the manuals (see Reference 5, pages 277 to 279, for a detailed example) is the initialization of structures and computation of addresses (see also the following discussion) which are required for the implementation of an algorithm. For any particular use, the difficulties are not great, but a general discussion would lead too far afield here.

access of work regions and internal computational data

One gains access to internal computational data by means of a systems macro DPALG, i.e., by the initialization:

```
%INCLUDE DPLALG;
```

The system permits the addressing of relevant data in symbolic fashion. These data reside in three possible areas:

- The H-region (keys and internal numbers of vectors in the basis; updated by ELIMN1 after each pivot step).
- The column byte map (logical information about the status of the variables).
- The work regions (a variable number of regions, each of XM double words).

The addressing scheme makes use of the PL/I operator ADDR and the capability of providing a "pointer" to a structure (by means of BASED) in a declaration statement of a structure.

For example, the PL/I statements

```
PCOL=ADDR(BMAP(J));
IF XBASIC THEN...;
```

permit the testing of the basic or nonbasic nature of variable j, because DPLALG contains a structure

```
DECLARE 1 BCOL BASED (PCOL),
        2 (XBASIC,XUB,XOBSUT,...),
        2 (...          );
```

which effectively functions as a "window" over the totality of data, picking out the wanted data item automatically.

In similar fashion, one is able to access the work regions by virtue of the DPLALG declaration statement:

```
DECLARE 1 WREGION BASED(PW).
        2 (WSIGN,MW) BIN FIXED(15),
```

```
2 JINC BIN FIXED(31),
2 W(XM REFER(MW)) DEC FLOAT(16);
```

To access the kth work region one must set

PW=XW(k);

and may then obtain the rth double word in that region by the statement:

a=W(r).

Important fundamental applications

There are a number of fundamental uses of the system that must be mentioned, even though they are fairly straightforward and need little exposition. The fact is that they may be most important from a day-to-day applications point of view. The Extended Control Language has much to offer in that it makes the implementation of such features simpler and potentially more powerful than was previously possible.

access to data bases

Future linear programming models will require increased utilization of realistic data from maintained data bases. An attractive feature of the Extended Control Language is that it permits the *direct* use of DL/I, the PL/I data-handling language of IMS (see Reference 8), a widely used data base system.

interactive computing

The use of interactivity is important from a data-managing and model-building point of view. Being called from a PL/I program, subject to standard PL/I compilation, MPSX/370 with the Extended Control Language feature can easily be put into the interactive mode, e.g., via the system TSO (time sharing option[9]).

There has also been a recent, albeit slow, trend towards interactivity in algorithmic work. Some of our previous work in References 10 and 11 may be useful in illustrating this concept.

interaction with other programs

Programs and program products (systems) in other languages can be accessed, as long as they can be accessed from PL/I. This capability opens possibilities for generating results through an MPSX/370 run and feeding them to other programs, or conversely using other programs to provide input data for MPSX/370. A good candidate for such interaction may be a *simulation* program, such as the system GPSS V.[12]

complicated data generation

Usually, the input to a system such as MPSX/370 is one of variable names and constraint coefficients, even if done with one of the matrix generation packages. It appears that the Extended Control Language will facilitate the generation of coefficients according to complicated rules.

Examples come to mind readily in the case of cost coefficients. The cost coefficients can be dependent on many parameters in nonlinear or discontinuous form. They might depend, typically, on rate tables, transportation constraints, union rules, etc. One can conceive of case studies that would require repetitive calculation of costs followed by optimization, possibly aided by user intervention in the interactive mode.

output generation

Report generation as such is not likely to require Extended Control Language features. However, especially in the case of a complicated model (as might be generated according to the above discussion), the useful output for management might require output generation within PL/I programs. For example, the output of the optimization run may be for a problem that was set up in an intermediate step of an overall procedure and would then require analysis, interpretation, and translation via programs written in PL/I for that purpose.

repeated, iterative use of MPSX/370

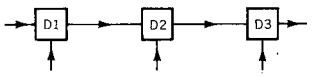
The repeated solution of a linear (integer) program is important in many ways. One instance derives from the fact that many linear programming models do not tell the entire story about the real situation. As a result, many solutions are unsatisfactory in practice. One may then have to write additional PL/I programs for checking the overall acceptability of the results and for rejecting the results (by modification of input data or by addition of new constraints) so that more acceptable solutions might be found in a repeated run. Interactivity (inspection of solution by management at a terminal) might play a big role in this mode of operation.

A particular instance of the above would be the case of problem constraints that are too weak or too tight. The reentrant mode would be used after a tightening or relaxation of the constraints (e.g., by changes in the right-hand sides) in a systematic or interactive (possibly unsystematic) fashion. One may also conduct a variety of parametric studies more general in nature than those permitted directly by procedures of MPSX/370 itself.

large structures

As shown schematically in Figure 1, many large-scale linear programming problems are for *interlinked* or *dynamic models*. Their main objective is to tie together a number of diverse operations (say, Department 1, Department 2, Department 3 of a corporation) or a number of dynamically repeated operations for the same department, with inputs passed on from stage to stage plus external inputs that are known or predicted.

Figure 1 Schematic of an interlinked model



Such a situation calls for the repeated solution of the same, or slightly modified, linear programming model with varying inputs. The linking of the various stages and preparation of the inputs can be executed readily within the Extended Control Language.

Many large-scale problems exhibit more complicated structures. Much of what has been said above would also apply to them. Certain ideas of valid heuristic approaches can perhaps be gained from discussions of heuristics or decomposition and partitioning in this paper, or from such sources as Reference 13.

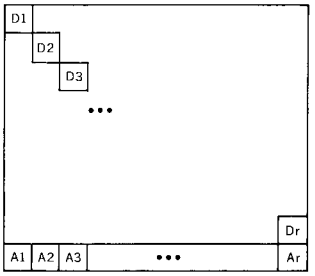
stagewise solution of problems

With many large problems, one may be content to use schemes that do not guarantee an optimum but may yet yield satisfactory answers (better than those attainable by other means). What is "large" may depend on the type of problem and on the machine on which one wants to solve the problem. Large may refer to a 3,000-row, 20,000-variable linear program, or to a 200-variable integer program, and for special structures the situation may again be quite different. Two possible approaches are briefly outlined here, even though they already lead into the next section on heuristic problem solving. In both instances, one solves approximate (relaxed) problems, that is, problems in which some of the constraints of the original problem have been left out.

decomposition procedures

Consider a problem of the structure depicted in Figure 2. Solving the subproblems D1, D2, · · ·, Dr in sequence, with the linking constraints below neglected, is in some sense an approximation to the overall problem. Quite often the large problem is generated by aggregation of models that solve the problems of individual departments D1, D2, · · ·, as if they were independent. Setting up the overall problem is then an attempt at recapturing the consequences of interlinkages among the departments.

Figure 2 Subproblems of a problem structure



The Extended Control Language provides the ability of collecting the solutions to the various subproblems and substituting them into the linking problem. One can then investigate the infeasibilities in the "linking" problem, and possibly *alter* the subproblems D1, D2, · · · as a consequence, in order to obtain an improved overall solution in another attempt. (The literature is full of schemes for such approaches, but the user is well-advised to be guided as much as possible by his *practical insights* into the problem.)

partitioning procedures

For very large integer problems without structure, the following approach can be quite feasible. Partition the integer variables into smaller sets, e.g., partition a 1,000-variable set into the 10 sets of variables (1 to 100), (101 to 200), · · ·, (901 to 1,000). Solve the problem initially as a mixed-integer problem with the first set of variables (1 to 100) *designated* as integer variables, the others as continuous variables. Given a solution to this problem, *fix* the integer-valued variables at their optimal values and solve the problem again with the first variables substituted and the second set of variables (101 to 200) designated as integer, etc. Of course, the success of such a scheme cannot be guaranteed. But many alternatives are possible, and a carefully monitored approach of

such a nature cannot fail to produce better insight, at least, into the nature of the overall problem than was initially available.

A similar approach in a structure situation was taken by one of the authors.[14] The problem was the Fixed Charge Transportation Problem, i.e., a transportation problem with shipment $x(i, j)$ on the routes (i, j), *and the additional* imposition of a fixed charge $f(i, j)$ if *any shipment* occurs on route (i, j). The problem is a mixed-integer problem with $m \cdot n$ (number of sources times number of destinations) integer variables $y(i, j)$.

The stagewise approach, which converges but can be interrupted whenever one is content with one of the obtained solutions, is that of deleting all integer variables and solving the transportation problem at the outset. At the next step, one includes as first set those $y(i, j)$ integer variables for which there were shipments on route (i, j). This approach leads back to a Fixed Charge Transportation Problem, but with hopefully *only few* (initially no more than $m + n - 1$) integer variables. In general, one adjoins to the current set of integer variables those $y(i, j)$ that have been newly introduced in the last solution obtained. For most data, such a solution procedure converges well before too many (namely close to $m \cdot n$) integer variables have to be taken into the mixed-integer problem. One good feature of such a method is that there are always many feasible solutions to the problem, a property that is by no means assured in general mixed-integer programming.

Heuristic problem solving

Heuristic approaches to mathematical programming problems are important in at least two ways. First, they may either afford the only means of getting solutions to otherwise intractable problems or may, at any rate, curtail the work in arriving at useful solutions. Second, they may be indispensable *within* algorithms designed for the efficient computation of truly optimal solutions.

The latter argument for heuristics has recently been bolstered by important new work in computational complexity,[15] which appears to make it quite clear that a very broad class of operations research (mathematical programming) problems are essentially of *the same degree of* complexity. All of these problems, and they appear to include most mixed-integer problems, are just about as difficult as the traveling salesman problem, and that problem is difficult indeed.

Such a "complexity" result does not mean that there is no hope of getting reasonable solutions to practical problems. What it suggests is that one should utilize all insights one can muster and

implement all heuristics one can think of, since *all brute force methods are of no avail by themselves*.

In terms of this paper then, the above constitutes a powerful argument for the use of the Extended Control Language as a tool towards implementing heuristics.

a general heuristic scheme

We are concerned with a problem P which is in a sense intractable. As a consequence, we replace it by a sequence of problems which *are tractable*:

1. Replace P by a relaxation P′.
2. Solve P′.
3. Test the solution of P′. Is it a good solution to P? If so, record it.
4. "Improve" the solution to P′. That is, use all ingenuity and insight at your disposal to arrive at a solution of P, taking as a starting point the solution to P′ just obtained. Record any solutions found.
5. Go to 1.

Several elements in the above scheme have been left undefined. For example: How does one obtain the relaxation P′? How does one avoid looping? When does one terminate? Answers to such questions are difficult to give in general, but relatively easy for a well-known special case.

rendering infeasible solutions feasible

It is often easy to arrive at solutions that are infeasible but, in some sense, close to feasibility for the overall problem. Computer programs are *in general* quite bad at "recognizing" such situations. For special problems, it is important that the user has means of *interrupting* the machine solution process from time to time, or under certain conditions, and to "look" at the solution in some detail. This condition implies the desirability of programmed interrupts in the Extended Control Language, via a suitable demand definition, with possible uses of interactivity, and subsequent alterations and reentry of system execution.

rounding

In integer programming, one of the natural ways of arriving at a potentially feasible integer solution is rounding. Most simply, one may use one input parameter r and round fractional variables:

- $x(j)$ *down* if $x(j) \leq [x(j)] + r$.
- $x(j)$ *up* if $x(j) > [x(j)] + r$.

While rounding ensures integrality, it does not ensure feasibility. One may therefore also have to pay attention to the constraints of the problems that are likely to be violated by the rounding process.

With the Extended Control Language one may interrupt the processing of MPSX-MIP/370 (MIP—mixed-integer programming) after generation of a node in the branch and bound procedure (using the XMXDOFRN demand with XMXFRN=1), and then try rounding (possibly in a fashion guided by one's knowledge of the problem) to generate feasible solutions.

problems with special structure

For certain mixed-integer problems, the very difficult task of finding *feasible* solutions can be made easy by exploitation of the structure. We give two examples, of many. The reader may wish to consult References 6, 7, and 16 for other examples.

The fixed-charge problem

For the fixed-charge problem:

$$\min \sum c(j) \cdot x(j) + \sum f(j) \cdot y(j) \tag{1}$$

$$A \cdot x \leq b \qquad \text{(normal constraints)}$$

$$x(j) \leq u(j) \cdot y(j) \qquad \text{(linking constraints, for all or some } j\text{)}$$

$$x(j) \geq 0,\ y(j) \in \{0, 1\}$$

in which the possibilities $y(j) = 0$ serve the purpose of suppressing activity $x(j)$ (and with it the fixed charge $f(j)$), corresponding to every fractional solution $0 < y(j) < 1$, there is always an integer feasible solution y ($y(j) = 1$ for $y(j) > 0$, $x(j)$ unchanged). References 17 and 18 describe a very important special case, the *plant location problem*.

Set covering

The problem is:

$$\min c \cdot y \tag{2}$$

$$E \cdot y \geq e$$

$$y(j) \in \{0, 1\} \quad \text{for all} \quad j$$

where E is an m by n matrix of zeros and ones and e is a unit vector of m ones. The objective is to pick columns j so as to cover the right-hand side at minimal cost. Applications are numerous and important (also for related problems, e.g., the problems with more general right-hand side "requirements").

For every fractional solution, one can again find a *feasible* integer solution by rounding the fractional variables up. More importantly, Reference 19 gives a procedure by which the *rounded* solution can be improved by being transformed to a vertex of the underlying polyhedron. Such a transformation, as well as other heuristics, can be carried out within the Extended Control Language at every node of the branch and bound algorithm of MPSX-MIP/370.

Multiple choice problems

Many resource allocation and scheduling problems have the additional constraints

$$\sum_{j \in J(k)} y(j) = 1, \qquad k = 1, 2, \cdots, p$$

where the $J(k)$ are often a *partitioning* of the original index set J for the (0, 1) integer (decision) variables $y(j)$; i.e.

$$J = J(1) + J(2) + \cdots + J(p)$$

Such problems are usually not very well-behaved, in the sense that the linear programming solutions invariably have fractional values for many of the $y(j), j \in J(k)$ (adding up to one, but otherwise not very meaningful). MPSX-MIP/370 has special features (SOS—specially ordered sets) for treating multiple choice problems. Even so, the possibility of judicious rounding to multiple choice feasible solutions should by no means be disregarded.

local exploration

It has long been known that many large-scale integer programming problems, especially in the difficult area of scheduling, are reasonably well-behaved under local exploration schemes. In one case, some lattice point (integer point, usually a vertex of the hyper-cube, for example, a point with coefficients all 0 or 1) is generated and then enumerated over "neighbors" of the point in some limited neighborhood. Good results for such an approach were probably obtained first for the traveling salesman problem.[20]

An airline scheduling program[21] used by some airlines was essentially based on such an approach. Relatively small subsets of columns (about 1,000 columns) were selected out of an enormous set-covering problem (millions of columns) and optimized over the subsets by the algorithm of Reference 19.

Many other important practical problems are likely to yield to such a mode of attack, and the Extended Control Language could be used whenever the frequent solution of linear or integer problems would be required in the overall process.

Algorithmic uses

In the previous section, we dealt with the more straightforward solution approaches, which would come to mind fairly readily. Yet, there is also an enormous literature on more sophisticated techniques. Out of the vast literature, the user might initially wish to consult books such as those in References 6 and 7 on integer programming, in Reference 13 on large-scale linear programming, and perhaps in Reference 22 on general operations research approaches.

Relatively little of practical advantage has as yet accrued from this body of fine work, and there are surely many opportunities for coming up with applications of great usefulness.

In the following discussion, we merely scratch the surface to give some idea of the potential. The emphasis is again on mixed-integer programming, not only because that is where most of our own experience lies, but also because a very large percentage of currently unsolved practical problems fall into this field. A detailed exposition would surely be too lengthy, so we present a mere outline, with one or two references to papers or books in which more references can be found.

nonlinear programming

There are many ways in which linear programming subproblems on the one hand, or possibly modifications of the simplex method on the other hand, can be important in nonlinear programming.

- Programs with quadratic objective function and linear constraints are much easier than the general nonlinear case. There are possibilities of modifying the simplex method itself for a direct approach to the problem. For older methods the reader may consult Reference 22. Fundamental new ideas stem from the work of Lemke.[23] They have given rise to a body of literature on fixed point and complementary pivot algorithms that is clearly important,[24] but for which it is difficult to foresee what role, if any, the simplex method will play.
- Many general problems lend themselves to approximate treatment via piecewise linearization of the constraints and/or the objective function.
- For problems with structure, there are partitioning methods that require the iterative solution of linear programs.[25]
- Among a variety of search methods that lead to repeated solution of linear programs, one may consider the "methods of feasible direction,"[26] or "sequential search methods,"[27] etc.

enumerative schemes for pure integer programming

The standard commercial mixed-integer programming codes use branch and bound programming, primarily because of its basic simplicity and reliability. However, there is some evidence that this technique does not function at its best for pure integer programs, especially pure (0, 1) programs. Enumerative techniques appear to be better in many instances, in particular when the problems are such that certain constraints can be taken care of implicitly rather than explicitly in the search scheme.

The fundamental work on branch and bound schemes goes back to Little, Sweeney, Murty, Karel, Land, Doig, and Dakin; that on enumeration to Balas and others.[6,7] For some new approaches, which may be of interest in connection with the Extended Control Language, see References 10, 11, and 28.

automatic invocation of decomposition techniques

Decomposition, partitioning, and column generation techniques have been around for a while (based on the work of Dantzig, Wolfe, Benders, Gomory, and Gilmore[13]) but have found application primarily for special problems only. It appears that the Extended Control Language opens up possibilities for generalized implementation of such techniques, always coupled with periodic checks that are necessary to avoid continuation of the algorithm when it begins to become ineffective (the major drawback of all such methods, which should *not* prevent their judicious use).

It is clear that branch and bound methods and enumeration methods can be combined in many ways, but there has been hardly any computational work of such a nature, undoubtedly on account of the considerable programming difficulties. Recent work suggests that there is some potential for such approaches, especially in the interactive mode. Linear programming subproblems are important, and for large problems, the use of a robust linear programming code, such as MPSX/370, would be essential.[11]

Lagrangian methods

For specially structured problems, there has been substantial interest in Lagrangian methods that place linear combinations of constraints into the objective function (see Reference 29 for a fundamental paper and Reference 30 for an excellent summary). The method is easy to implement and can be quite effective for the right kind of problem.

cutting-plane methods

An entire body of literature is devoted to the addition of cutting planes to linear programs in order to remove unwanted vertices of the underlying polyhedron. For integer programming, these methods are primarily associated with Gomory. Other important contributions were made by Johnson, Balas, Glover, etc. in integer programming and by Kelley for convex programming.[6,7,13] As to practical applications, much the same can be said as for Lagrangian methods, and a case can be made for augmenting commercial codes so as to accommodate the addition of cutting planes, as inconvenient as that may be in codes that are organized around a matrix with a fixed number of rows (the addition of new rows requires the invoking of the system procedure SETUP in the case of MPSX/370). This difficulty may be overcome by a sufficient number of empty rows added to the original model.

exploitation of special structures

Finally, it may be redundant, but it is probably still worthwhile to stress once more the importance of looking for and heeding the existence of structure in mathematical programming. Few reliable statistics exist. But it is probably fair to say that a high percentage, perhaps even an overwhelming percentage, of large-scale and difficult mathematical programming problems have special structure. It is, of course, tempting to resolve them by means of a general purpose code. However, when that does *not* lead to satisfactory results, and when the problem is sufficiently impor-

tant to warrant the work that is required, then there is little choice other than to examine and exploit the structure, whether by algorithmic or by heuristic means.

In many instances, the Extended Control Language will provide the user with a tool for implementing whatever techniques are required to take advantage of the special nature of the problem.

ACKNOWLEDGMENT

We would like to acknowledge the helpfulness of our colleagues M. Guignard, C. Scotti, and H. V. Smith. Most importantly, however, we want to point out that our colleagues of the Paris Program Product Center of IBM, M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, had the essential role in the development of the Extended Control Language.

CITED REFERENCES AND FOOTNOTE

1. For the Extended Control Language of MPSX/370, the IBM Program Product is 5740-XM3 for OS and 5746-XM2 for DOS.
2. M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent, "Experiments in mixed-integer linear programming." *Mathematical Programming* **1**, 76–94 (1971).
3. M. Benichou, J. M. Gauthier, G. Hentges, and G. Ribiere, "The efficient solution of large linear programming problems: Some algorithmic techniques and computational results," *IXth International Symposium on Mathematical Programming*, Budapest (1976).
4. *IBM Mathematical Programming System Extended/370 (MPSX/370), Control Languages*, SH19-1094, IBM Corporation, Paris, France (April 1975).
5. *IBM Mathematical Programming System Extended/370 (MPSX/370), Program Reference Manual*, SH19-1095, IBM Corporation, Paris, France (January 1976).
6. R. S. Garfinkel and G. L. Namhauser, *Integer Programming*, Wiley Interscience, New York (1972).
7. H. M. Salkin, *Integer Programming*, Addison-Wesley, Reading, MA (1975).
8. *IBM Information Management System/Virtual Storage (IMS/VS)*, GH20-4106 S (Program Product 5734-XX6), IBM Corporation, Data Processing Division, White Plains, NY.
9. *IBM TSO Assembler Prompter*, GC28-6698 (Program Product 5734-CP2), IBM Corporation, Data Processing Division, White Plains, NY.
10. M. Guignard and K. Spielberg, "Propagation, penalty improvement and use of logical inequalities in interactive (0, 1) programming," *Symposium ueber Operations Research*, University of Heidelberg, Germany (1976).
11. M. Guignard and K. Spielberg, "An experimental interactive system for integer programming," Working Paper (1976).
12. *IBM General Purpose Simulation System V, Introductory User's Manual*, SH20-0866 (Program Product 5734-XS2), IBM Corporation, Data Processing Division, White Plains, NY (1971).
13. L. Lasdon, *Optimization Theory for Large Systems*, The MacMillan Co., New York (1970).
14. K. Spielberg, "On the fixed charge transportation problem," *Proceedings of the 19th National ACM Conference*, A1.1-1 to A1.1-13 (1964).
15. R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Problems*, Editors: R. E. Miller and J. W. Thatcher, 85–104, Plenum Press, New York (1972).
16. M. L. Balinski and K. Spielberg, "Methods for integer programming: Algebraic combinatorial, and enumerative," in *Progress in Operations Research, Vol. III*, Editor: J. Aronofsky, John Wiley & Sons, New York (1969).

17. K. Spielberg, "Algorithms for the simple plant location problem with some side conditions," *Journal of the Operations Research Society of America* **17,** 85–111 (1969).
18. M. Guignard and K. Spielberg, "Algorithms for exploiting the structure of the simple plant location problem," *Annals of Discrete Mathematics* **1,** 247–271, North-Holland Publishing Co. (1977).
19. C. E. Lemke, H. M. Salkin, and K. Spielberg, "Set covering by single branch enumeration with linear programming subproblems," *Journal of the Operations Research Society of America* **9,** 998–1022 (1971).
20. S. Lin, "Computer solutions of the traveling salesman problem," *The Bell System Technical Journal* **44,** 2245–2269 (1965).
21. J. Rubin, "A technique for the solution of massive set covering problems, with application to airline crew scheduling." *Transportation Science* **7,** 34–48 (1973).
22. H. M. Wagner, *Principles of Operations Research*, Prentice-Hall, Englewood Cliffs, NJ (1969).
23. C. E. Lemke, "Bimatrix equilibrium points and mathematical programming," *Management Science* **7,** 681–689 (1965).
24. B. C. Eaves and H. Scarf, "The solution of systems of piecewise linear equations," *Mathematics of Operations Research* **1,** 1–27 (1976).
25. J. B. Rosen, "Convex partitioning programming," in *Recent Advances in Mathematical Programming*, Editors: R. L. Graves and P. H. Wolfe, McGraw-Hill Book Co., New York (1963).
26. G. Zoutendijk, *Methods of Feasible Directions*, American Elsevier, New York (1960).
27. H. Glass and L. Cooper, "Sequential search: A method for solving constraint optimization problems," *Journal of the ACM* (1965).
28. M. Guignard and K. Spielberg, "Reduction methods for state enumeration integer programming," *Annals of Discrete Mathematics* **1,** 273–285, North-Holland Publishing Co. (1977).
29. M. Held and R. M. Karp, "The traveling salesman problem and minimum spanning trees: Part II," *Mathematical Programming* **1,** 6–25 (1971).
30. A. M. Geoffrion, "Lagrangian relaxation for integer programming," *Mathematical Programming Study* **2,** 82–114 (1974).

Reprint Order No. G321-5064