Experimental Implementations of Mixed Integer Programming Algorithms

M. D. Grigoriadis, IBM Corp., New York, N. Y.
L. Papayanopoulos and K. Spielberg, IBM Corp., Yorktown Heights, N. Y.

KEY WORDS AND PHRASES: mixed integer programming, mathematical programming, linear programming, integer programming, optimization, branch and bound, algorithmic experimentation
CR CATEGORIES: 4.22,5.40,5.41,5.49

INTRODUCTION

The Mixed Integer Programming Interface Subroutine System (MIPIS)* consists of a set of programs written to facilitate the implementation of most known MIP algorithms in a common system environment. This experimental system [3] makes no claim to overall (running time) efficiency since it utilizes the experimental, Fortran based linear programming (LP) code NYLPS (New York Scientific Center Linear Programming System [7])* known to be less efficient than other commercially available LP codes (e.g. MPS/360, etc.) written in Assembly Language. NYLPS is an OS/360 extension of LPS/360 [8] which operates under DOS/360.

The study and associated development of MIPIS was directed toward examining the manner in which MIP algorithms would be implemented, in a flexible environment, rather than toward implementing another MIP production code. It is hoped that this experiment will motivate designers to provide additional flexibility for building and extending the scope of future mathematical programming systems.

*These experimental systems are for internal use only and are not available from IBM.

ENVIRONMENT

Generally, production type, large MIP codes are not easily modifiable and are limited in algorithmic variety. Though the particular algorithm used by such codes is often mathematically simple, it is always embedded in a much larger LP system with elaborate data management, report generation facilities, etc. In order to obtain the best possible execution performance, these codes usually are written in a low level (e.g. BAL) programming language. More often than not, the result is a large and complex system. Algorithmic modification and extension of such a system and its enrichment with new MIP algorithms, as they may become available, present practical implementation difficulties. These are the considerable development expense and the necessity for excellent systems programming skills complemented

by an adequate knowledge of mathematical programming methods.

A number of experimental codes implementing a single algorithm in a high level language do allow easy modification and extension to other algorithms. However, they are limited in problem solving capacity due to their lack of sophisticated data handling capability.

Nevertheless, experimenters, as well as sophisticated users, find that neither type of system precisely meets their requirements. The former system is too costly and time consuming for exploratory studies, whereas the latter does not provide an environment where experimental MIP techniques, their programming requirements and their computational behavior can be studied for large scale problems. Furthermore, the great differences in hardware and programming systems make realistic comparison of available MIP programs virtually impossible.

DESIGN CONSIDERATIONS

The choice of NYLPS as a background system for MIPIS was based on several factors: 1) Problem solving capacity comparable to all other large commercially available codes, 2) extensive data manipulation and maintenance capabilities, 3) highly modular structure of simple Fortran routines, modified easily, 4) simple but comprehensive problem control language and 5) operation under OS/360 in multiprogramming mode.

The desirability of (5) arises from the basic design of NYLPS. In order to handle large problems, the system operates "out-of-core," that is, it treats segments of the problem which it brings in from disk, one at a time. This requires a considerable amount of disk I/O the cost of which can be excessive if I/O is not completely overlapped. Thus, the NYLPS/MIPIS system is most suitable for a large machine

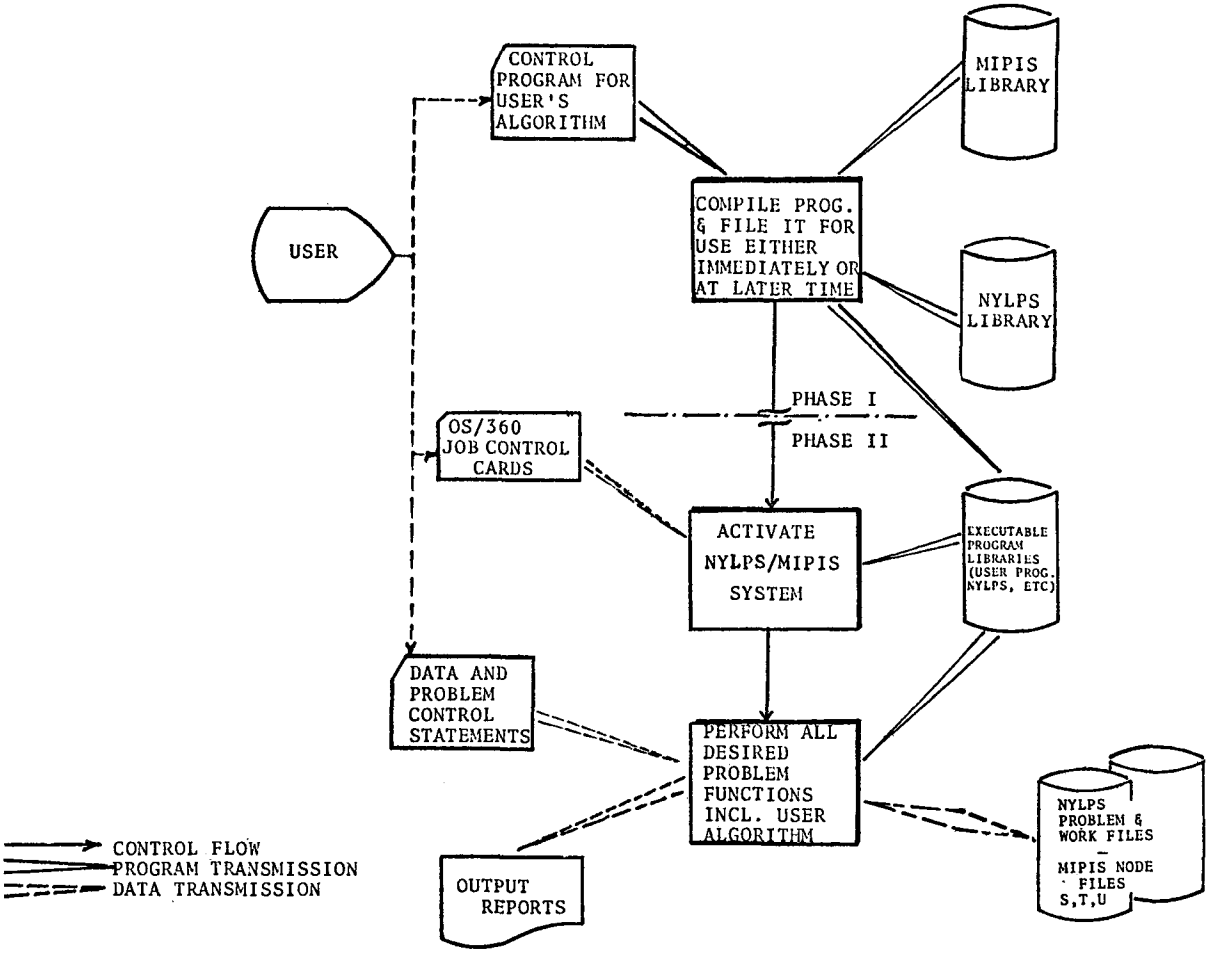

FIGURE 1 COMPILATION AND EXECUTION PHASES OF A USER WRITTEN ALGORITHM

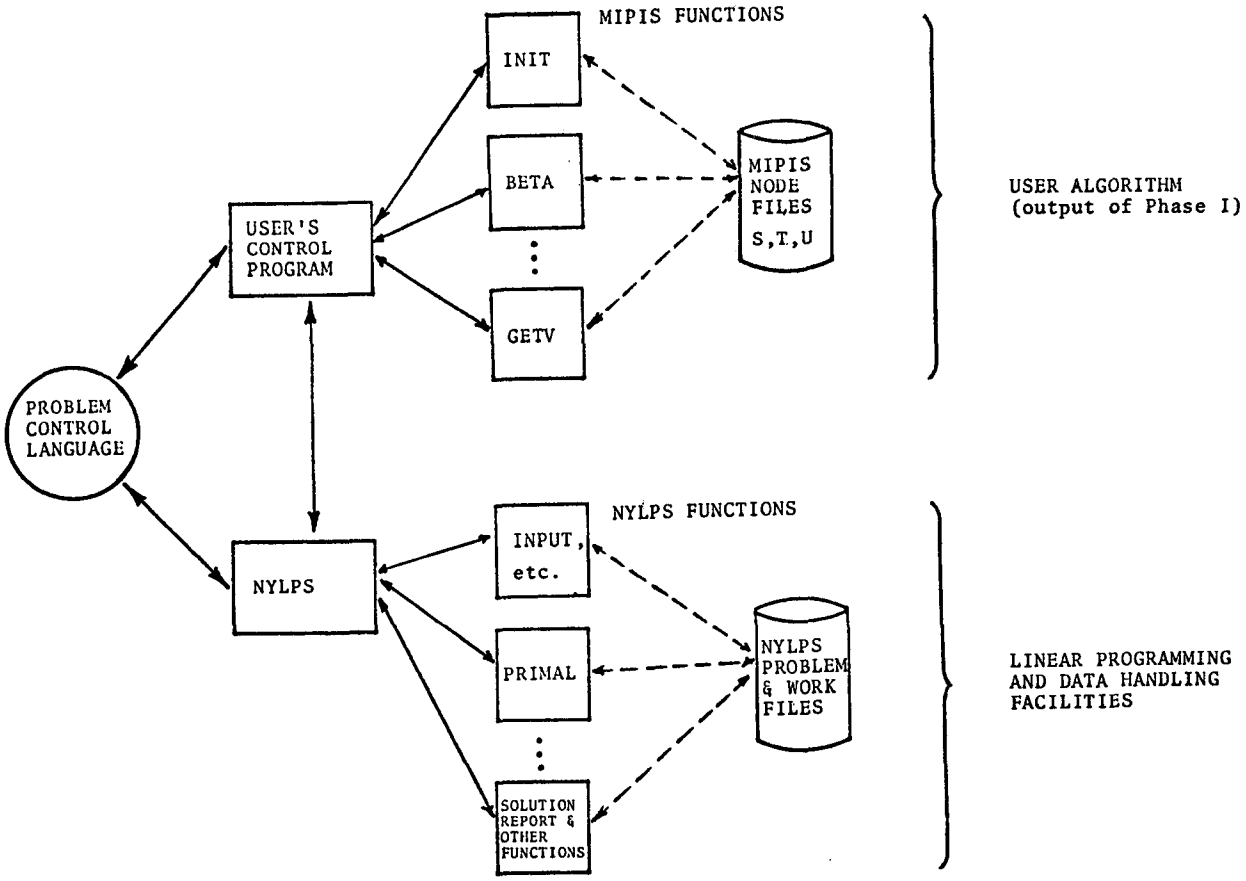

FIGURE 2 THE COMMUNICATION OF A USER ALGORITHM WITH NYLPS DURING EXECUTION

(e.g. a 360/65 with a million bytes) running under multiprogramming (e.g. OS/MVT).

For the purposes of this study, the "user" was assumed to be: a) the designer of new MIP algorithms for large scale problems, b) the experimenter who might compare solution strategies on the same algorithm and/or problem and might synthesize new algorithms for basic ones available in the system, c) the more sophisticated analyst who knows that the performance of an MIP algorithm is almost unpredictable and who wishes to explore various algorithms to select the one best suited to his class of problems.

In spite of the comparable simplicity the user would have a considerable task of learning and programming before he would be ready to implement an MIP algorithm using directly the LP capabilities of NYLPS. The purpose of MIPIS is to furnish an "interface" between the user and the system in order to minimize this training and programming.

MIPIS contains most, if not all, of the facilities needed to implement typical MIP algorithms. These facilities may be used (through simple CALL statements) as "building blocks" which are arranged easily in the construction and modification of experimental techniques. An overview of the User-MIPIS-NYLPS relationship is shown in Figure 1. Phase I depicts the preparation of the "user program" in which the user employs MIPIS facilities by means of Fortran CALL statements. When the program is compiled, it is placed in a library. Subsequently, in Phase II the compiled program is fetched for execution as part of the overall NYLPS overlay structure. Since the compilation output is retained on disk, Phase I need not be performed more than once for each new algorithm introduced into the system.

Figure 2 illustrates the flow of control among the functional units of the system during execution. A user program is entered via the problem control language. Subsequently,

depending on the requirements of the algorithm, the user program requests the services of some or all of the MIPIS functions (e.g. INIT, BETA, GETV). Similarly it accesses NYLPS facilities for input of problem data, linear programming optimization, etc. In each case, use of the appropriate files is made as indicated by the broken lines of the diagram.

MIPIS FACILITIES AND DATA FILES

For the purposes of this study most of the known algorithms of branch-and-bound (e.g. [5],[6]) and enumerative [1] types were examined to establish their computational requirements and degree of similarity [4]. Almost all of these involve the initial solution of the continuous LP and the subsequent consideration of a sequence of solution vectors whose (integer restrained) components are fixed at appropriate integral levels (or constrained in appropriate intervals) as dictated by the particular algorithm. By and large the algorithms generate information stored in successive nodes of a computational tree, compute "penalties" at each node, shift from one branch to another according to various criteria, etc. Single branch, multibranch and combinations of single and multibranch trees may be considered, as well as other data structures, penalty tables, lists, etc.

During the course of the search, the nodes of a tree and the information associated with each node are generated by means of an auxiliary program, e.g. LP, over some selected ("free") set of variables, and subsequently used by other parts of the algorithm. This information, potentially of an extremely large volume, must be stored and retrieved in an efficient manner from auxiliary data storage devices.

In addition to the Problem and Work files used by NYLPS, MIPIS utilizes the following three data files, each assigned to a direct access data set:

S - "Dictionary File," contains the LP objective function value at each node, ordered to facilitate retrieval of the "best" node and (pending/not pending) node indicator.

T - "Node (information) File," contains detailed blocks of information (e.g. the bounds of the integer constrained variables, the next variable to branch on, etc.) associated with each node in the S file.

U - A dictionary file for single branch algorithms. This file is used to store objective function data for an ordered set of nodes (along a single branch of the search tree). It facilitates backtracking and branching by eliminating the need to search the disk for a previous "pending" node.

If a particular algorithm requires the selection of the best branching point among all pending nodes, this is accomplished through an examination of the dictionary files S and U. When the most desirable node is found, the dictionary record specifies the record in the library file T where the complete definition of the node is located.

The facilities required by the vast majority of MIP algorithms are listed below. Each of these is given a mnemonic corresponding to the MIPIS subroutine name which can be called to perform that task.

SETB: Set or change the bounds of a variable.

OUT: Write the information associated with the current node into the dictionary and node library.

IN: Read the data associated with a particular node into core from the library file. Also prune the tree of all nodes with higher objective function values than the best integer solution, and select the next "best node" as dictated by the algorithm.

GENR: Generate rows and compute "penalties" for each integer constrained variable which is currently basic and at a nonintegral level.

GETV: Retrieve the values of the primal and dual variables (e.g. after a LP solution is performed).

GETB: Obtain the bounds of a given variable.

RESB: Restore previous bounds on all variables.

BETA: Determine the status and values of the basic "integer" variables.

SVSL: Save the data (bounds, etc.) at a particular node for subsequent use.

RSSL: Restore a particular node (saved by SVSL at some previous stage).
BEND: Generate a "Benders inequality" — a necessary condition to be satisfied by the integer constrained variables.
INIT: Initialize core and files.
PRINT: Print an integral solution.

IMPLEMENTING AN MIP ALGORITHM VIA MIPIS

The implementation of the branch-and-bound algorithm illustrated here uses the well known Dakin [5] modification of the Land and Doig [6] method of search. Integrality of the integer variables under this technique is attained through the progressive constriction of the bounds imposed on these variables. The flowchart in Figure 3 demonstrates the construction of this algorithm in terms of the MIPIS facilities. The user written program requires only a few Fortran statements.

TEST RESULTS

A number of MIP problems were run to test the algorithm described above. The self-explanatory test results are summarized in Table 1. The comparative figures enclosed in parentheses correspond to test runs of a multibranch version of the same algorithm.

ACKNOWLEDGEMENTS

The authors acknowledge the valuable efforts of their colleagues L. Bodin, J. Colmin, Pat Grapes(Mrs.), K. Harrow, S. Torok and W.W. White who participated in the implementation of MIPIS.

The test problems were contributed by members of the MIP Subcommittee of the SHARE MPS Project and were compiled by Miss Roberta L.A. Heintz.

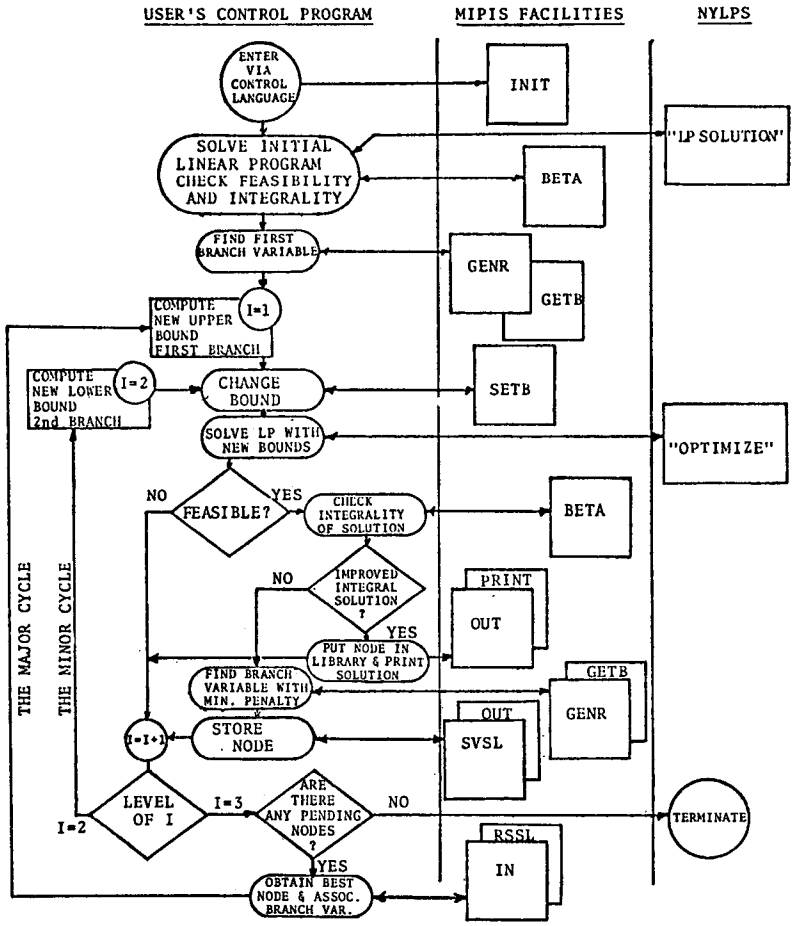

FIGURE 3 AN MIPIS IMPLEMENTATION OF THE DAKIN ALGORITHM

Problem Name	No. of rows	No. of columns	No. of integer variables	No. of integer solutions found	No. of LP's solved	No. of pivots to proven integer solution	Function value at opt. continuous solution	Function value at opt. integer solution
UCAMP085	86	90	15	1 (1)	23 (23)	199 (199)	8,152.20	8,498.60
NY2835	29	35	35	1 (2)	45 (51)	371 (461)	521.05	550.00
NY2889M	29	89	31	2 (6)	103 (105)	872 (968)	834.68	946.70
COBLEND	60	195	15	3 (3)	115 (121)	2768 (3063)	28,442.35	29,565.92
CO.PSL	10	14	12	5 (1)	41 (27)	162 (106)	-742.17	-727.00
ART1ST	25	47	29	2 (2)	129 (139)	626 (718)	717.00	818.75

TABLE 1 - TEST RESULTS

REFERENCES

1. Balinski, M.L. and K. Spielberg, "Methods for Integer Programming: Algebraic, Combinatorial and Enumerative," Progress in Operations Research, Vol. 3 (Julius S. Aronofsky, ed.), John Wiley and Sons, 1969.

2. Benders, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems," Numerische Mathematik, Vol. 4, 1962, pp.238-252.

3. Bodin, L. et alias, "The NYLPS/MIPIS System for Mixed Integer Programming," Doc. No. 71PSC1, Philadelphia Scientific Center, IBM Corporation, 1971 (Restricted Distribution).

4. Colmin, J. and K. Spielberg, "Branch and Bound Schemes for Mixed Integer Programming," Report #320-2972, New York Scientific Center, IBM Corporation, May 1969.

5. Dakin, R.J., "A Tree Search Algorithm for Mixed Integer Programming Problems," The Computer Journal, Vol. 8, 1965, pp.250-255.

6. Land, A.H. and A. Doig, "An Automatic Method of Solving Discrete Programming Problems," Econometrica, Vol. 28, 1960, pp.497-520.

7. Papayanopoulos, L.J., "The New York Scientific Center Linear Programming System for OS/360," Doc. No. 69 NYSC 3, New York Scientific Center, IBM Corporation (Restricted Distribution), 1969.

8. Linear Programming System/360, (LPS/360) Program Description Manual, Publication No. H20-0607, Program Information Department, IBM Corporation.