IBM

Application Program

H20-0238-0

1130 Linear Programming

Mathematical Optimization Subroutine System

Application Description

The 1130 Linear Programming - Mathematical Optimization Subroutine System provides the 1130 disk user with a simple, efficient means of solving linear programming problems, and with a powerful tool for implementing other mathematical optimization applications.

The system contains all the routines necessary to solve a linear programming problem and to perform an extensive postoptimal analysis of the problem. The system provides extensive data generation and maintenance facilities.

To solve a linear programming problem, these routines are called into core storage by procedure control statements that define the processing sequence. The control statements may originate on cards or on paper tape.

The LP-MOSS routines can also be used like subroutines by a program written in FORTRAN. This is accomplished by a special command that is translated into valid FORTRAN statements by a system translator. The command calls a program into core storage to be executed before returning control to the next statement in the calling program.

This manual contains a description of the system, the machine configuration required, and limits on problem size.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

CONTENTS

INTRODUCTION

GENERAL DESCRIPTION OF LP-MOSS/1130

The 1130 Linear Programming — Mathematical Optimization Subroutine System (LP-MOSS) provides the 1130 disk user with a simple, efficient means of solving linear programming problems, and with a powerful tool for implementing other mathematical optimization applications.

Mathematical optimization may be defined as any mathematical technique for determining the optimum use of various resources (for example, capital, raw materials, manpower, plant, or other facilities) to attain a particular objective (for example, minimum cost or maximum profit) when there are alternate uses for the resources. Linear programming is the most widely used of these techniques, and has been used to allocate, assign, schedule, select, or evaluate the uses of limited resources for various jobs, such as blending, mixing, bidding, cutting, trimming, pricing, purchasing, planning, and the transportation and distribution of raw materials and finished products.

LP-MOSS consists of a comprehensive set of programs and subroutines that perform the data processing and computational tasks necessary for the solution and analysis of linear programming problems. These routines constitute the Mathematical Optimization Subroutine System (MOSS).

Since a large number of business optimization problems can be solved by a straightforward application of linear programming, LP-MOSS also provides the Linear Programming System (LPS). LPS uses the MOSS routines to form an easily used, yet powerful and efficient system for solving linear programming problems.

LINEAR PROGRAMMING

LP-MOSS contains a linear programming system that provides the user with a very easy means of solving linear programming problems. For the specified 1130 configuration, the system has a logical processing capacity of 700 rows, including all objective rows. The number of columns is limited only by the disk space available.

The LPS uses, automatically, sophisticated scaling and inversion techniques to improve solution speed and accuracy. However, some numerical difficulty may be encountered with large problems in the upper half of the row capacity range.

The input data (for example, the chemical analysis of raw materials) originates on cards or on paper tape and is stored on disk for later processing. A disk may contain one or more problems, which can be updated and rerun periodically. Disk-stored data can be referenced to simplify new-problem preparation. A problem may contain alternate objectives (for example, different sets of raw material costs) and may include alternate sets of problem bounds (for example, different sets of alloy specifications). Master problems can be defined by conbining subproblems (for example, corporate plan from division plans).

Output reports are available on cards, paper tape, typewriter, or printer. Output options include a full solution report, a comprehensive solution analysis, and parametric reports.

MATHEMATICAL OPTIMIZATION

MOSS is designed for optimization applications that can use LP, or a variation of LP, as part of the optimization procedure. MOSS is suited especially to applications that require:

1. Use of linear programming optimization as a subroutine
2. Programmed data generation to cut costs or time in preparing input data
3. Specialized report preparation
4. An optimization procedure that is a modified form of the linear programming solution procedure

The system contains a comprehensive set of linear programming routines and a program translator. The MOSS routines are written primarily in FORTRAN and are highly segmented into subroutines to simplify user adaptation. The LP optimization routine has a logical processing capacity of 700 rows. The optimization technique used is a two-phase (composite), bounded variable, product form of the inverse revised simplex algorithm, with multiple pricing for problems with up to 350 rows. The program translator allows the user to write FORTRAN programs that call (EXECUTE) other programs as subroutines.

The MOSS internal data organization is designed for flexibility and simplicity of use in FORTRAN programs. Several facilities, including SORT and MERGE, are provided for use in generating and maintaining data.

FEATURES

Linear Programming System

Large problem capacity

Program logic provides for up to 700 rows

Simple, flexible, processing control

Optional conditional control of processing sequence

Special solution of simultaneous equations procedure

Simple problem definition

Easy-to-use format

Extensive data maintenance functions

Automatic slack generation

Specification of a starting solution (basis)

Definition of new problem by reference to subsets of old data

Combination of problems to form master problems

Advanced mathematical methods

Automatic, iterative, input scaling for accuracy

Revised simplex method (product form of inverse) for rapid processing

Bounded variable feature for range ($\leq$ and $\geq$) constraints and bounded variables to simplify problem description and to increase problem capacity and solution speed

Multiple pricing for increased solution speed

Efficient triangularization inversion method for speed and accuracy

Extensive postoptimal analysis options

Discrete parametric analysis for all problem data

Activity-cost-bound relationship for all variables

Extensive checking

Input check for duplicate entries

Solution processing check to test for need of early inversion and/or rescaling

Output reports include solution check

Output flexibility

Complete or partial output reports

Specification of output device

Mathematical Optimization Subroutine System

FORTRAN language routines

Routines highly segmented into subroutines

Execution of a program as a subroutine

FORTRAN internal data files

Easy use in FORTRAN programs

Formats can be extended for special requirements

SORT and MERGE routines

Output data files to simplify preparation of special reports

USING THE SYSTEM

Use of the linear programming system requires absolutely no knowledge of MOSS. However, MOSS can often be used to extend the use of linear programming in an application. So that the user can consider such extensions, a brief discussion of MOSS and its relationship with LPS is given below. The user who wishes to use only LPS can read the LPS section of the manual and then skip straight to the appendix. The user whose problems require the use of MOSS should read both the LPS and MOSS sections of the manual.

LP-MOSS Structure

MOSS consists of a set of routines that perform the various linear programming functions, together with a means of using these routines like subroutines in a program written in FORTRAN, using special MOSS control statements.

LPS is a set of FORTRAN programs that call the MOSS routines to carry out the various phases of a straightforward linear programming application. These programs are controlled, in turn, by the LPS control program, which reads and interprets control records to determine the sequence in which the programs are called. Thus, LPS can be regarded as a standard control program written to simplify the use of MOSS for the straightforward solution and analysis of linear programming problems.

MOSS can also be used in many other ways. If he so desires, the user can very easily write further programs to be called by control records via the LPS control program, or even via his own control record system, making use of both the LPS and MOSS routines to perform the special functions required by his application.

THE LINEAR PROGRAMMING SYSTEM

USING LPS

LPS is composed of sets of programs, called procedures, which are stored on the disk. The user controls the use of these programs by procedure control statements, which are read by the LPS control program. Each control statement causes a procedure to be called into core storage and executed. When the procedure is completed, the control program reads the next statement.

Solving a Problem

The basic control procedures can specify completely the solution sequence for input, linear programming optimization, and solution reporting.

INPUT	Reads input (matrix and limits) data and stores the problem on disk. The matrix describes the linear relationship of variables to each other and to the objective (a matrix may contain alternate objectives). The problem limits are usually specified by a bound set. A bound set specifies the permissible solution activity range for each variable (a problem may contain alternate bound sets).
MOVE	Reads processing specifications (for example, name of the BOUND set and OBJECTive).
MINIMIZE (MAXIMIZE)	Computes the minimum (maximum) value of the objective that will meet the problem requirements.
LPSOLUTION	Prepares a solution report with the name, objective value (for example, cost), and problem limits of each variable of the input problem data, as well as the computed solution value and the reduced cost (Dj) or marginal value (PI value).
ENDJOB	Ends the linear programming run.

Illustration of Basic Control

Figure 1 shows how the user would set up a run to obtain a minimum cost solution to an alloy blending problem.

Postoptimal Analysis of a Problem

These procedures are used to determine the effects of changing a problem or of departing from its optimum solution.

For example, a manufacturing process may have a bottleneck at one stage of the process and overcapacity at another. One additional machine at the bottleneck stage would considerably improve production, and hence profits; an extra machine at the other stage would merely increase the overcapacity. Similarly, a machine breakdown at the bottleneck stage would seriously impair production, while a breakdown at the overcapacity stage would have a minute effect.

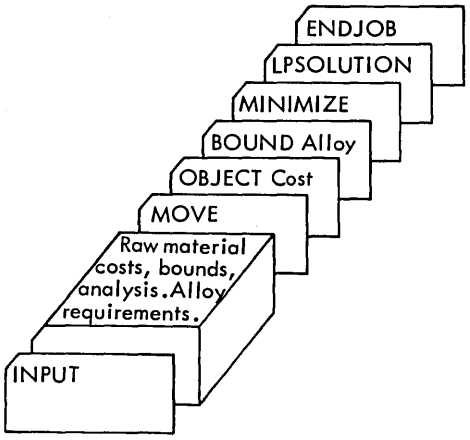

Figure 1

The postoptimal procedures are used to pinpoint such bottlenecks, material shortages, or vital contracts, so that steps can be taken to improve profitability or, at least, so that extra precautions can be taken at crucial points.

The description of the procedures below is given in terms of costs or profits, but the procedures are applicable equally, whether the objective is cost, profit, machine time, throughput, or any other quantity.

LPANALYSIS — Determines the effects of changes in cost, bounds, or activity of a single variable. The report can be used, for example, to investigate the effects of discounts, material shortages, or overtime working, and to pinpoint bottlenecks or overcapacity. Included are:

1. The cost per unit of changing the activity of a variable, and the activity range over which this cost is valid

2. The profit per unit of changing the bounds of a variable, and the range of bounds over which this is valid

3. The range of costs over which the activity of a variable would not change, and the activity levels to which it would change if the cost left this range

LPPARAMETRIC Investigates the effects of changes to the entire problem. The procedure provides an analysis of the changes in profitability and use of resources as the problem data is modified systematically over a specified range. The report can be used, for example, to consider the effects of technological changes, statistical variations in materials or processes, and widespread shortages or price increases. The procedure accepts a set of changes to be made to any of the problem data. The change data is added to the original data, and the problem is resolved to find the new optimal solution. The process is repeated as many times as specified by the user, with solution reports printed at specified intervals. A MOVE statement specifies the number of REPORTS and the INTERVAL at which they are printed (see Figure 2).

Illustration of Postoptimal Analysis

Figure 2 shows how the user would set up a run to obtain a minimum cost solution, the postoptimal analysis report, and five solutions with variations of the input data. LPSOLUTION reports will occur at A+ .1B, A+ .2B, A+ .3B, A+ .4B, A+ .5B, where A is the original problem data and B the change data.

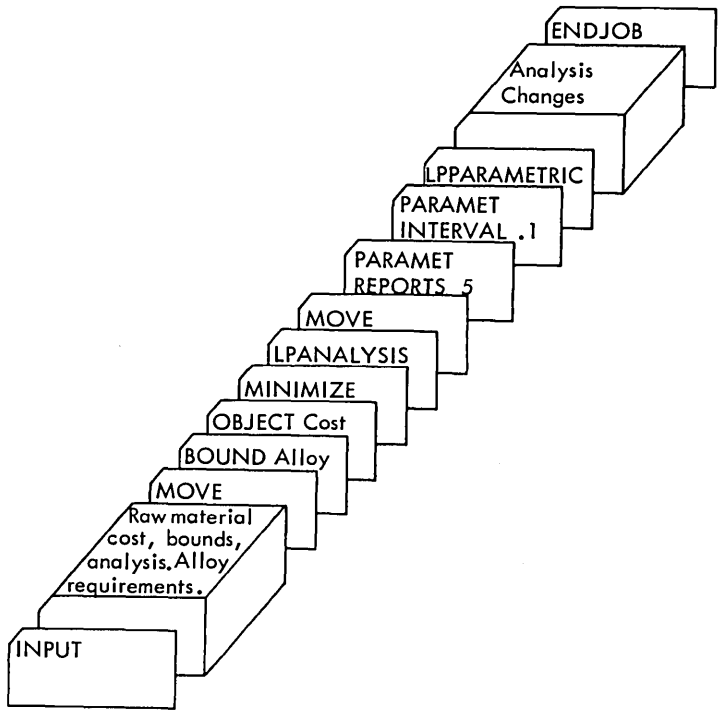

Figure 2

Maintaining Problem Data

These procedures are provided to simplify problem data maintenance, preparation, and verification. They provide facilities to store and maintain problem data on the disk, and to allow disk-stored problems to be combined to form master problems.

SAVEDATA — Preserves the problem data input previously for later reference and/or reprocessing.

REVISE — Reads input data from cards or paper tape to alter the current problem data. The formulator may change, delete, and/or add data to the problem to alter or correct the problem formulation.

MERGE — Forms a new problem by combining disk-stored problems. The subproblems can be input, processed, and tested for validity before the formation of the master problem. Problems formed by MERGE can be preserved for maintenance by SAVEDATA and can be modified by REVISE.

DELETE — Deletes an obsolete problem. A problem that is not saved (SAVEDATA) during the formation of the problem by INPUT or by MERGE is deleted automatically by ENDJOB or by the formation of a new problem.

Example of Data Maintenance

Figure 3 shows how a corporate problem (model) is to be formed using the data from two plants. Each plant model is first to be solved independently and an analysis obtained.

Selecting Problem Data

These features are provided to simplify the selection of data from disk-stored input files and output reports, and to expedite solution processing.

Processing Selection

SAVESOLUTION — Saves the current problem solution for use as an advanced starting position for a later optimization. The user can save a number of solutions, with each given an appropriate name, to provide alternate starting solutions. A solution is named by MOVEing the "name" to SOLUTION before SAVESOLUTION. When the solution is not named, it becomes the normal starting solution for this problem.

RESTORE — Restores a previously saved solution for use as an advanced basis for the current optimization. The name of the restored solution must be MOVEd to SOLUTION before RESTORE.

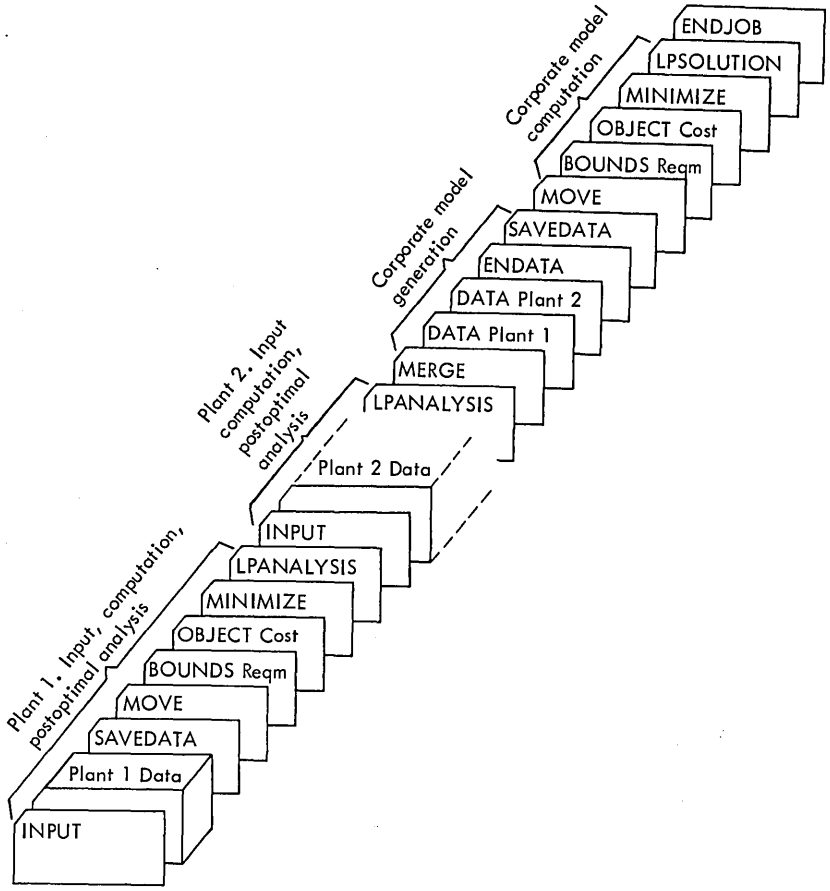

Figure 3

RESET — Deletes a previously saved solution, and RESTOREs the normal starting solution for this problem. The normal starting solution is an all logical (generated) variable solution unless a solution has been input or SAVESOLUTIONed.

Input Selection and Reference

Data from a previous problem can be referenced. The user can use all, or can select a part, of the data to form a new problem.

For many applications it is useful to maintain a file of variables to be used in problem description. This file becomes an inventory of variables, from which problems can be defined by a selection of a number of the columns, usually process variables.

The inventory of variables is the matrix that describes the interrelationships of variables. A list or file of rows, and a list or file of columns, can define the variables of a new problem as a subset of a master problem.

Indicator cards can be used to reference previously input and saved data. Data can be selected from a previous input by a COLS file and/or a ROWS file.

Illustration of Data Selection

The formulator has a new problem that uses some of the data (MATRIX) of a previous problem. This new problem is to be run periodically. The two bound sets of this problem produce very dissimilar solutions; hence the formulator would like to reprocess the problem constrained by each of the bound sets, beginning from the previous solution to the bound set (see Figure 4).

Conditional Control of the Solution

The user may desire to alter the processing sequence, depending upon circumstances that arise during the run. LPS enables the user to test processing conditions and to alter the processing sequence, if desired.

IF	Enables the user to test one or more conditions (for example, INFEAS-infeasible LP problem) and to bypass portions of the solution sequence (for example, LPANALYSIS) IF the condition has occurred.
IFNOT	Enables the user to test and to bypass portions of the solution sequence IF the condition has NOT occurred.

Conditions:

OPTIMUM	An OPTIMUM (and feasible) solution to an LP problem has been obtained.
NORMAL	The previous procedure has been completed NORMALly.
MAJOR	A MAJOR error, usually due to faulty input data, has been detected during the preceding procedure.
MINOR	A MINOR error, indicating possible input data error, has been detected during the preceding procedure.
UNBOUND	The current LP problem has no finite solution; an UNBOUNDed objective.
INFEAS	The current LP problem has no FEASible solution; an INFEASible problem.
ANY	Always. This condition is always true; it allows an unconditional break or interruption to bypass a portion of the solution sequence.

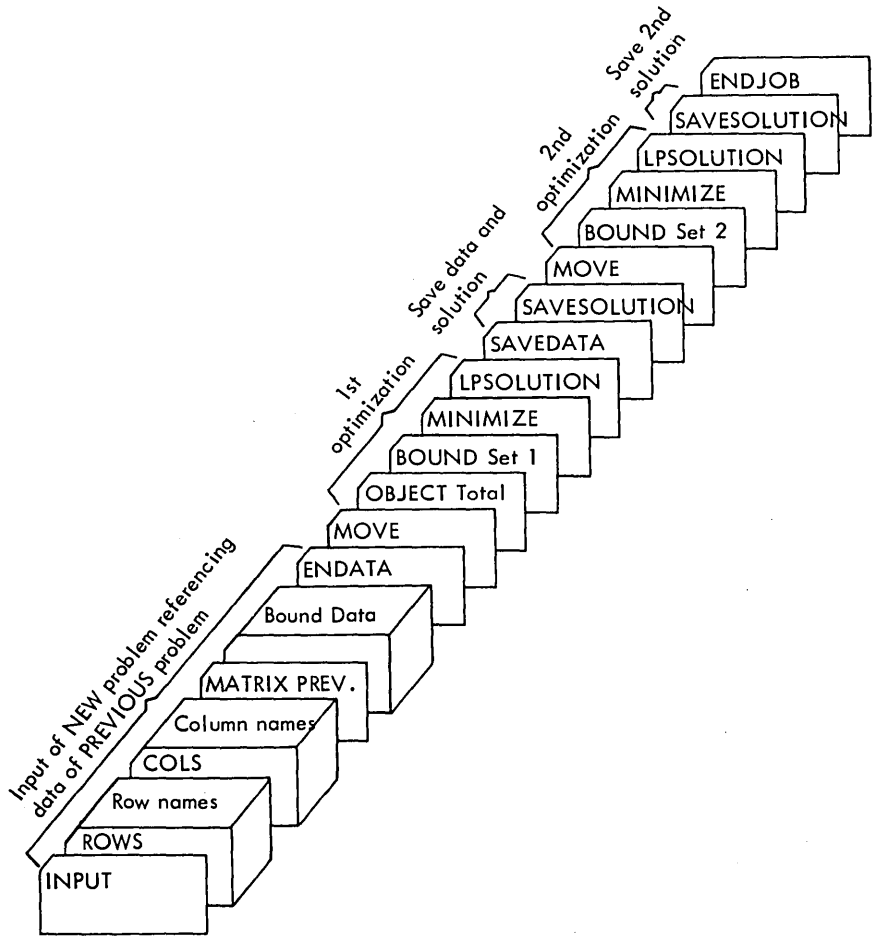

Figure 4.

Example of Logic and Conditional Control

Shown below is the sequence of statements necessary to obtain the solution to an LP problem. If the solution is OPTIMUM, take an LPANALYSIS and a solution using a secondary objective. If the first solution is INFEASible, change bound sets and obtain the new solution. If the first solution is neither OPTIMUM nor INFEASible, take an LPSOLUTION report; if the solution is UNBOUNDed, terminate the run.

```
INPUT
     Data                    Standard input through optimization
MOVE
     BOUNDS     SET1
     OBJECT     PROFIT1
MAXIMIZE

IF
     MAJOR      3RUN         Bypass until 3RUN if a MAJOR error occurred.
     OPTIMUM    2RUN         Bypass until 2RUN if LP solution is OPTIMUM.
     INFEAS     1RUN         Bypass until 1RUN if LP solution is INFEASible.

LPSOLUTION                  If not OPTIMUM, prepare LPSOLUTION report.

IF
     UNBOUND    3RUN         Bypass until 3RUN if solution is UNBOUNDed.

1RUN
MOVE
     BOUNDS     SET2         Solution INFEASible (not OPTIMUM and not
MAXIMIZE                    UNBOUNDed); obtain LP optimum with alternate
LPSOLUTION                  BOUNDS, and obtain solution report.
IF                          Bypass remainder of procedure until 3RUN.
     ANY        3RUN

Label  2RUN                 This sequence is executed only if the first solution is
       LPANALYSIS           OPTIMUM.
       MOVE
         OBJECT  PROFIT2
       MAXIMIZE

       IF
         MAJOR   3RUN       Test for MAJOR error on second optimization.  Prepare
       LPSOLUTION           solution report.

Label  3RUN
       ENDJOB
```

Restart and Off-Site Data Preparation

LPS provides facilities for interrupting the solution in process during the MINIMIZE (or MAXIMIZE) procedure and for restarting the solution from the point of interruption. The RESTART procedure can also be used to recover from computer malfunction, provided that the problem data on the disk has not been overwritten.

```
RESTART                     Allows the solution process to be restarted from the
                            point of interruption.  If machine malfunction occurs, the
                            procedure recovers close to the point of malfunction.
```

Solution of Simultaneous Equations

A special procedure, SOLVE, can be used to solve (invert) a system of simultaneous equations and to prepare a solution report. The inverse is maintained in a factored, or product, form of inverse for efficient inversion of sparse (mostly zero coefficients) matrices.

SOLVE — Performs an inversion and obtains a solution report for the specified right-hand-side (RHS) column. For problems that contain multiple right-hand-side columns, reports are written only for the specified right-hand sides. The solution report contains the input name and the computed solution value for each variable.

Figure 5 shows how the user would set up a run to obtain solutions for the right-hand sides STANDARD and TEST.

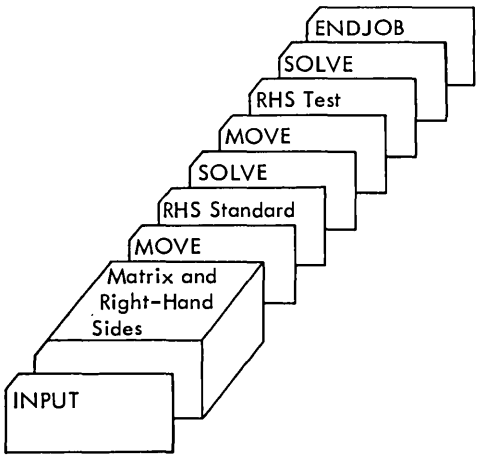

Figure 5

SYSTEMS CHART AND DESCRIPTION

LPS is composed of an LPS control program and procedures that are stored on the disk. The 1130 Monitor calls the mathematical optimization subroutine system monitor. The MOSS monitor then calls (by a control record) the LPS control program (see Figure 6).

User control is exercised by control statements. Procedure control statement records cause a particular procedure to be called into core storage and executed. When the procedure is completed, the next control record is read. Logic (IF and IFNOT) control records cause condition cards to be read and tested for condition match; a label search is executed when a condition is satisfied. Specification (MOVE) control records cause specification records to be read to alter current processing parameters (name of objective, tolerances, frequencies, for example).

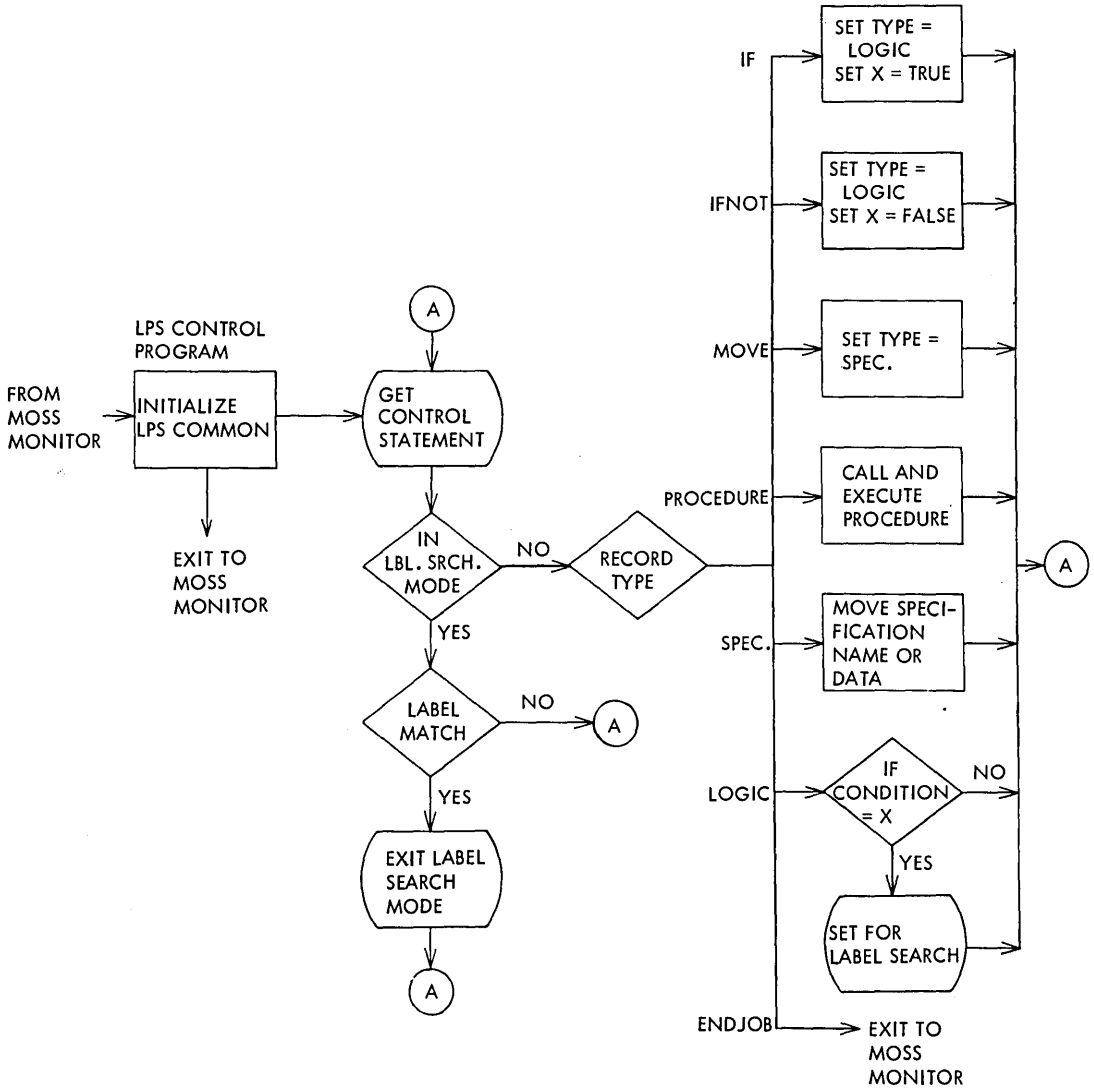

Figure 6.

RESTRICTIONS AND RANGE

The following formulas can be utilized to find the approximate number of disk sectors used in storing and processing a problem:

If a problem is to be input, processed, and the solution output taken, the approximate number of disk sectors used is (R+C)(1/5+N/25).

If the problem is to be input only, the approximate number of disk sectors used is (R+C)/25+CN/50.

Where: R = the number of rows in the problem

C = the number of columns (including RHS, range, and bound sets)

N = the number of nonzero elements per column

Examples:

Rows	Columns	Nonzero Elements Per Column	Sectors Used Input Only	Total
100	200	6	36	132
100	200	12	60	204
250	500	10	130	550
700	1000	5	168	680

THE MATHEMATICAL OPTIMIZATION SUBROUTINE SYSTEM

USING MOSS

Linear Programming Applications

To implement a linear programming application, the user writes a FORTRAN control program that can use any of the MOSS routines like a subroutine (using the EXECUTE statement). Any FORTRAN statement can be used in such a program, providing an extremely powerful control language. The user control program can also EXECUTE programs written by the user. Such programs can, for example, be written to generate data for the MOSS LP routines, or to write management reports from the solution files formed on the disk by the MOSS routines.

Control programs can be written many levels deep (that is, a program can EXECUTE a second program, which EXECUTES a third program, etc.), so that complete systems can be built around the MOSS and user routines. An example is the IBM Linear Programming System/1130, which uses a FORTRAN control program to read control cards. The control program calls other programs in the sequence specified by the cards, which, in turn, use the MOSS routines to solve linear programming problems and systems of simultaneous equations.

This feature of MOSS enables the user to construct complete procedures that use linear programming "inline". For instance, it is possible for the user to write a production control and stock control system for feed manufacturing that:

1. Accepts as input the production requirements for a period

2. Builds a linear programming model of the manufacturing requirements for the period, taking into account availability of stocks, plant, etc.

3. Determines the optimum production program for the period by solving the linear programming model

4. Accepts changes to the program necessitated by management policy decisions

5. Generates replenishment orders for stocks that fall below the safety level as a result of the final program

To implement such a system, the user first generates an overall model of his production operations. The model is generated by the CONVERT routine from unit record input and is maintained, as necessary, by REVISE. Then, to carry out the period-by-period calculation of the optimum production program, the user writes procedures to:

1. Use the CONVERT programs and subroutines to generate a set of problem bounds from the production requirements for the period

2. SETUP the complete model for the period, using the overall model and generated bounds set

3. Call PRIMAL to find the optimum production program, and FILEANALYSIS to save the program on a disk file (this file is used to print a management report describing the optimum program, and the effect on profits of changing the program)

4. REVISE the generated bounds set as necessitated by management decisions, find the new optimum program, and print worksheets, management summaries, etc.

5. Use the final FILESOLUTION file to update inventory and print replenishment orders

Mathematical Optimization Applications

MOSS is designed to be a very powerful tool for implementing mathematical optimization techniques, especially those which use linear programming or which are variations of linear programming.

The internal data formats used by MOSS are highly flexible and are easily handled in FORTRAN programs. The records of a file need not be in physical sequence on the disk, allowing records to be added to or deleted from a file at will. In particular, rows and columns can easily be added to the LP processing matrix.

MOSS provides SORT and MERGE facilities for use in generating and maintaining data. The LP input data (bounds and matrix elements) can be generated in any sequence and sorted later into the correct sequence by the MOSS data conversion routines. This feature is valuable especially in techniques that generate and solve a series of linear programming problems.

Techniques that are variations of linear programming, such as separable programming or the trim program, can easily be implemented by modifying the MOSS LP routines. These routines are highly segmented, and segments can easily be replaced or added. For example, to implement a trim program, the LP pricing routine can be replaced by dynamic programming pricing and column generation routines.

THE APPLICATION MONITOR

The MOSS application monitor reads and interprets control records that call user programs or the MOSS translator. Any program that is compiled from statements generated by the translator can be called in the application monitor.

The application monitor serves two functions:

1. When first called by an 1130 Monitor XEQ statement, the application monitor initializes core storage for use by MOSS programs.

2. The application monitor reads a control record and inserts the name of the program to be called into a dummy EXECUTE statement.

The monitor is written entirely in FORTRAN, except for the subroutine that sets the name of the program to be called in the dummy EXECUTE statement.

THE TRANSLATOR

One of the main features of MOSS is the ability to write programs that can be used like subroutines by other programs. To accomplish this, programs are written in FORTRAN, augmented by four system statements — COPY, START, EXECUTE, and RETURN. The MOSS translator is used to generate a valid FORTRAN program by translating the system statements into valid FORTRAN statements. This program can then be compiled by the 1130 FORTRAN Compiler.

The use of each of the system statements is described below (see also the systems chart).

COPY — A set of valid FORTRAN statements can be stored on the disk in card image form and generated later in a program by a COPY statement. This statement can be used, for instance, to generate the set of statements necessary to define a COMMON area that is used by several programs and subroutines.

START — This must be the first executable statement of any program that is to be used in MOSS.

EXECUTE — This statement allows one program to use another like a subroutine. The program to be EXECUTEd is read in to core storage, overlaying the calling program. When the program has been EXECUTEd (as defined by a RETURN statement), the calling program is read back in to core storage, and control is passed to the statement after the EXECUTE statement. Program switches in the calling program are not saved unless they are in the COMMON area of both the EXECUTEd and calling programs. A program being EXECUTEd may, in turn, EXECUTE a third program.

RETURN — This statement returns control to the statement after the EXECUTE statement that called the program.

SYSTEMS CHART AND DESCRIPTION

The MOSS application monitor is called by an XEQ card to the IBM 1130 Monitor. User programs and the MOSS translator are called by control records to the MOSS application monitor (see Figure 7).

The user program called can EXECUTE MOSS routines or other user programs, which can, in turn, EXECUTE MOSS routines and user programs. Each heavily outlined box in Figure 7 defines a single program. As each MOSS routine or user program is completed, control returns to the calling program until the highest level program (the one called by a MOSS application monitor control record) is completed. Control then returns to the MOSS application monitor, which reads another control record to determine the next program to be called. When all MOSS jobs are completed, as determined by an END control record, the MOSS application monitor returns control to the 1130 Monitor.

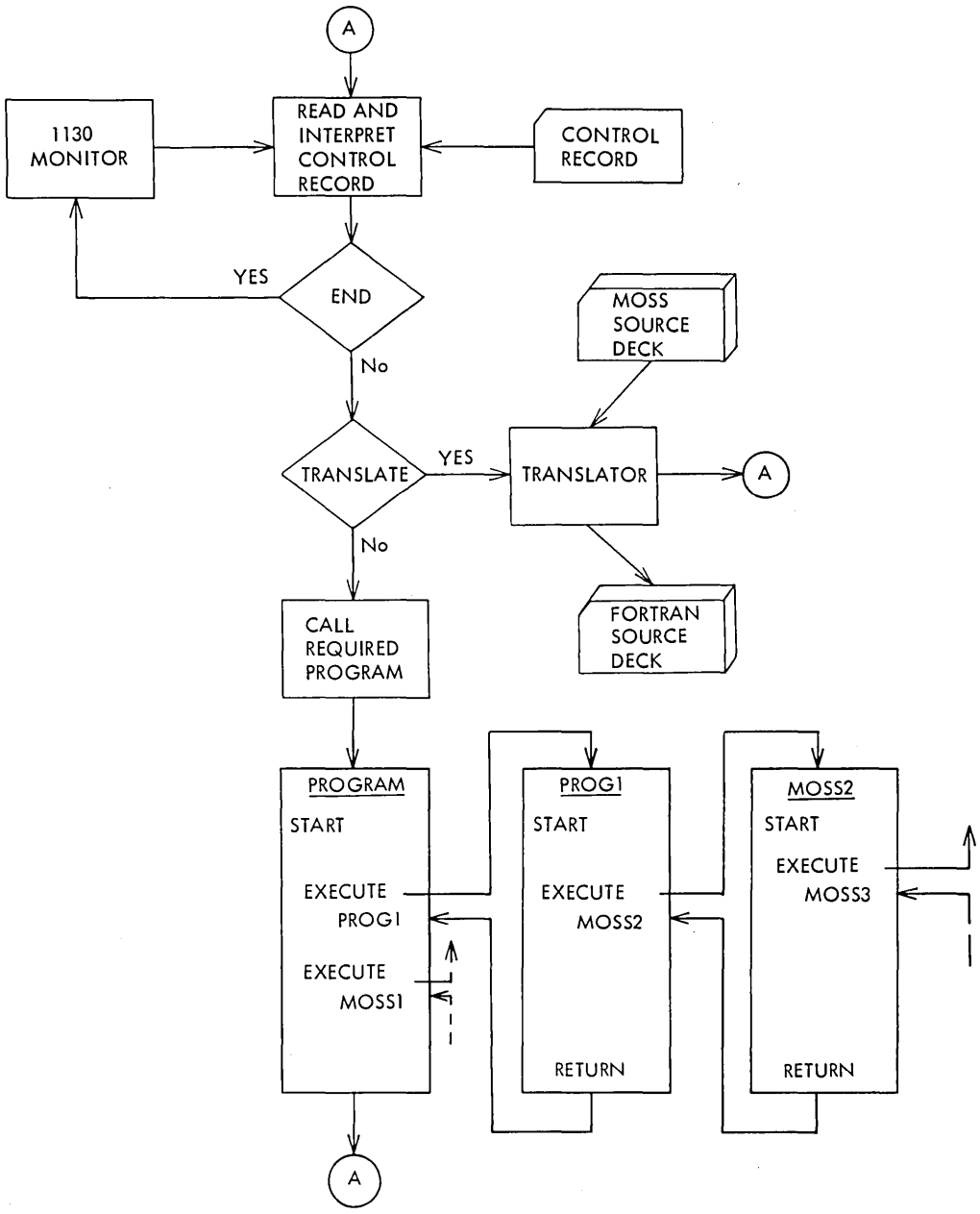
A
1130
MONITOR
READ AND
INTERPRET
CONTROL
RECORD
CONTROL
RECORD
YES
END
No
MOSS
SOURCE
DECK
TRANSLATE
YES
TRANSLATOR
A
No
CALL
REQUIRED
PROGRAM
FORTRAN
SOURCE
DECK
PROGRAM
START
EXECUTE
PROG1
EXECUTE
MOSS1
PROG1
START
EXECUTE
MOSS2
RETURN
MOSS2
START
EXECUTE
MOSS3
RETURN
A

Figure 7.

Figure 8 shows the sequence of cards necessary to call the MOSS application monitor and perform two jobs. The first job is to call and execute a program, PROGA, which also requires some data. When this is completed, the MOSS translator is to be called to prepare a FORTRAN source deck from a program written using the system pseudo-FORTRAN statements. The program source deck is followed by enough blank cards to contain the FORTRAN deck generated by the translator. Extra blank cards are ignored.

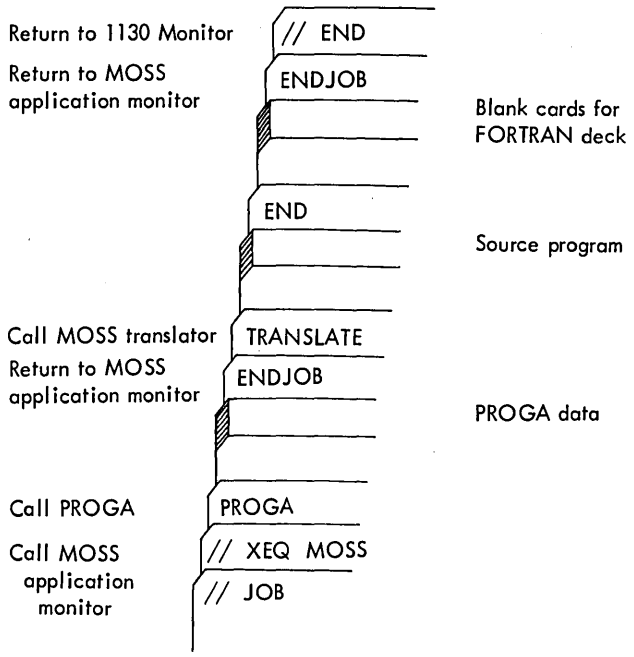

Figure 8.

MOSS FILES

Internal (disk) files are used to store problem information. The data files are designed for problem data generation and maintenance, and to provide processing and output flexibility. The processing files are designed for computation efficiency.

An internal file is composed of one or more elements (groups of information). The elements are stored in system-defined, fixed-length records. The records may be in any physical order in an external file, and may be split or divided over several external files. An external file is a contiguous area of a direct access data device.

Data Files

The MOSS data files are used to store the input data and the problem solution. The input data may be stored by a MOSS conversion routine from unit record inputs, or may be generated by a user-written routine that uses MOSS subroutines to open, write, and close data files, and to sort the data as required for processing. The solution report may be prepared by a MOSS output routine from disk solution files, or may be prepared by a user-written routine that uses MOSS subroutines to reorder and merge solution files for a special report. The input data is separated into identification files and coefficient files. The coefficient files contain the matrix and other problem coefficients, with set and variable indices. The set and variable indices (for example, the column and row indices of a matrix element) correspond to unique indices in the name identification file. This separation allows for an efficient storage of input data and simplifies data generation and maintenance.

The output data file contains the name and solution values of each variable, in addition to bounds and other information. MOSS routines can be used to reorder, combine, or otherwise modify this file to simplify the preparation of special reports by a user program.

Problem communication files are used to pass information to and from MOSS routines and user-written routines, to allow one or more problems to be permanently maintained by the system, and to provide interproblem communication. The communication files are used to specify the external file location of internal files (for example, the location of the output data file), and to allow one problem to reference data stored in another problem (for example, the first problem refers to the matrix coefficients of the second problem) to simplify problem definition and to simplify data maintenance.

Processing Files

The MOSS processing files are used to store computation data. MOSS routines are provided to set up an efficient computation matrix from the input coefficient data file; other MOSS routines generate additional processing files during the optimization. The output data file is generated by a MOSS routine from processing files.

Processing file elements (groups of information) represent vectors (rows or columns). The elements in a processing file may be split or divided over one or more records.

The vectors may be compressed (nonzero coefficient and index for nonzero vector elements) or expanded (all coefficients, no indices).

MOSS ROUTINES

The MOSS routines can be divided into:

- Data maintenance routines — store unit record input in data files and provide file maintenance
- Processing file generation routines — prepare processing files from data files
- Computation routines — perform computations with processing files; generate additional processing files
- Output data file generation routines — prepare data files for output reports
- Output report routines — prepare output reports from output data files

Data Maintenance Routines

MOSS contains extensive data maintenance features to simplify user preparation of unit record input and/or programmed data generation.

CONVERT — Reads unit record input to generate system data files. The input can be in any sequence convenient to the application.

REVISE — Reads unit record changes to alter the system data files. The changes can include additional rows or columns, deletions of rows or columns, and coefficient changes.

MERGE — Combines two system (for example, subproblems) data files to form a master (problem) data file. For instance, MERGE can be used to construct time period models or corporate models from individual plant models.

SORT — Sorts system data files on any specified field(s). This routine can be used to order system data files generated by a user routine.

Processing File Generation Routines

SETUP — Forms an efficient processing matrix from the problem data files. The processing matrix file contains a compact representation of the coefficients of the matrix. A variable "status" data file is also generated that governs the computation process. It is in system data format to simplify user specialization of computation procedures. The variable status file element includes name of variable, type of variable (free, nonpositive, fixed, or nonnegative), solution status (basic or intermediate, at upper bound, at lower bound), and positions for the

scale factor, effective upper bound and effective lower bound. The variable status item can be expanded to include additional processing parameters.

SETBOUND — Supplies the effective bounds on variables in the variable status file from one or more bound data files. The effective bounds are the limits on the solution activity of a variable. The activity bounds can be specified by a BOUND set in the BOUND coefficient data file to limit all variables, structural as well as logical. The activity bounds for logical variables can also be specified by a right-hand-side column, with or without a range column.

Computation Routines

These routines and subroutines are highly segmented to simplify adaptation to other optimization techniques (such as, trim, separable, quadratic).

SCALE — Calculates and applies row (Ri) and column (Cj) scale values. Ri and Cj are calculated to reduce the function $\sum_i \sum_j (CjRi\,|Aij|-1)^2$, where the summation applies only to the nonzero matrix elements Aij.

INVERT — Prepares a product form of inverse representation for variables specified at intermediate level in the variable status list. This routine prepares a sparse representation, as the product of column eta matrices, by a triangularization technique. The sparse representation improves processing efficiency and accuracy.

PRIMAL — Calculates the solution to a set of linear inequalities that maximizes (or minimizes) an objective. The objective may be a row (logical variable) or a structural variable. The objective variable may be bounded.

PRIMAL uses a bounded variable, product form of inverse, revised simplex technique. Range constraints, $\underline{bi} \leq$ Row i $\leq \overline{bi}$, are handled by bounding the logical variable.

The solution process of this highly modular routine is controlled by the variable status file to simplify user adaptation (for example, trim, separable).

CHECK — Computes a check of the solution, to verify processing accuracy.

Output Data File Generation Routines

These routines prepare systems data files from the processing files. The output data files are used by the output report routines to prepare reports.

FILESOLUTION — Forms a file containing an optimization summary for each variable. A file item includes name of variable, objective value (cost, for example), effective upper bound and effective lower bound, computed column scale, unscaled solution activity, and unscaled reduced cost. The reduced cost of a logical variable is the unsigned simplex multiplier or pi-value.

FILEANALYSIS — Forms a file containing postoptimal analysis information, in addition to the (FILESOLUTION) optimization summary. The analysis data in each item includes the cost-per-unit increase (and cost-per-unit decrease) in the solution activity of a variable, and the changes (both increase and decrease) in solution activity for which the costs are valid.

This file is used by the ANALYSIS report routine. Traditional reports (RHS RANGE, COST RANGE, DO. D/J) can be prepared from this file.

Output Report Routines

These routines use the files formed by the output data file generation routines to prepare the standard LPSOLUTION and LPANALYSIS reports (see "Solving a Problem" and "Postoptimal Analysis of a Problem").

PUTSOLUTION — Prepares the LPSOLUTION report from files generated by either the FILESOLUTION routine or the FILEANALYSIS routine.

PUTANALYSIS — Prepares the LPANALYSIS report from files generated by the FILEANALYSIS routine.

APPENDIX

TIMING

It is impossible to predict the solution time for a particular linear programming problem, even if the solution time for a similar problem is known. However, as a very rough guide to estimating throughput using LP-MOSS/1130, the table below shows the estimated approximate solution times for various problem sizes. The times shown are for the initial solution and for solution beginning at an advanced starting solution.

Number of		Approximate Solution Time (hours)	
ROWS	COLUMNS	Initial	Advanced Starting Solution
100	150	.5 - 1.3	.2 - .4
250	300	3 - 8	1 - 2
400	500	7.5 - 20	2 - 5
500	650	12 - 32	3 - 8
700	1000	24 - 60	6 - 15

PRECISION AND ACCURACY

All computations are performed in extended precision (31-bit mantissa). Linear programming accuracy is usually a function of problem size, scaling, complexity, and system mantissa length. The system scaling procedures and inversion methods are designed to produce accurate, reliable solutions within the limits of a 31-bit mantissa.

MACHINE AND SYSTEM CONFIGURATION

1130 Model 2B with 8192 words of core storage and one disk storage drive
1442 Card Read Punch and/or 1054 Paper Tape Reader and 1055 Paper Tape Punch
1132 Printer (optional).

The recommended 1130 system for best performance and simplest operation includes a 1442 Card Read Punch with an 1132 Printer.

Programming System

LP-MOSS/1130 operates under control of the IBM 1130 Monitor System. The source language is primarily IBM 1130 FORTRAN. The subroutines necessary to implement the EXECUTE and RETURN statements are written in IBM 1130 Assembler language.

BIBLIOGRAPHY

IBM PUBLICATIONS

Introduction to Linear Programming (E20-8171)

Application Manuals:

Aluminum Alloy Blending (E20-0127)

Electric Arc Furnace Steelmaking (E20-0147)

Feed Manufacturing (E20-0148)

Ice Cream Blending (E20-0156)

Blast Furnace Burdening (E20-0160)

Cotton Blending (E20-0164)

Gasoline Blending (E20-0168)

OTHER REFERENCES

Charnes, A. and W. W. Cooper, Management Models and Industrial Applications of Linear Programming, John Wiley & Sons, Inc., New York, 1961.

Dantzig, George B., Linear Programming and Extensions, Princeton University Press, 1963.

Garvin, Walter W., Introduction to Linear Programming, McGraw-Hill Book Company, Inc., New York, 1960.

Gass, Saul I., Linear Programming, Method and Applications, McGraw-Hill Book Company, Inc., New York, 1958.

Orchard-Hays, William, Matrices, Elimination and the Simplex Method, C-E-I-R, Inc., Arlington, Virginia, October 1961.

Riley, Vera and Robert Loring Allen, Interindustry Economic Studies, Operations Research Office, Johns Hopkins Press, Baltimore, Maryland, May 1955.

Riley, Vera and S. I. Gass, Bibliography on Linear Programming and Related Techniques, Johns Hopkins Press, Baltimore, Maryland, 1958.

Wolfe, Philip (ed.), The RAND Symposium on Mathematical Programming, Santa Monica, March 1959. Proceedings published by the RAND Corporation, R-351, 1960.

Printed in U.S.A. H20-0238-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601