Linear Programming on the PC: Loading through Multi-Plan and Testing.

by

Dwight Christie

B. S., Kansas State University, 1983

A Master's Report

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, KS 66506

1987

Approved by:

[illegible]

Major Professor

Acknowledgements

I wish to express my gratitude to committee members Dr. Bryan Schurle and Dr. Richard McBride. A special thank you goes to Dr. Elizabeth Unger for her guidance throughout this report. I would like to extend a very special thank you to my wife, Deborah. Without her love, support and constant encouragement this report may not have been possible.

Table of Contents

List of Figures

Chapter 1: Introduction

One of the most important and also most difficult decisions that a manager has to make are decisions involving organizational resources, such as manpower, equipment, or funds, that are to be allocated to a particular project or operation. (7) One method used to make such decisions is a 'seat-of-the-pants' process, or going with what 'feels' the best. For many cases this is the best and most economical way to arrive at such decisions, however, as the business becomes larger and more resources become involved this method becomes less and less accurate or acceptable. Linear programming is one method that has been used to present the information, about the resources, needed to make such decisions.

Definition of Linear Programming

A simple definition of linear programming, it is a form of economic analysis. Economic analysis is the study of four human activities: production, consumption, utilization and exchange. Linear programming is a quantitative technique of analysis, with the relationships quantified, the results specify the amount of each product that maximizes or minimizes the objective. (2)

A linear programming model can be mathematically expressed as:

$$
\begin{array}{ll}
\text{maximize or minimize} & Z = C(1)X(1) + C(2)X(2) + \ldots + C(n)X(n) \\
\text{subject to} & A(11)X(1) + A(12)X(2) + \ldots + A(1n)X(n) <= B(1) \\
 & A(21)X(1) + A(22)X(2) + \ldots + A(2n)X(n) <= B(2) \\
 & \vdots \\
 & A(m1)X(1) + A(m2)X(2) + \ldots + A(2n)X(n) <= B(m) \\
 & \\
\text{where} & X(1), X(2) \ldots X(n) >= 0 \\
 & C(i)\text{'s, } B(i)\text{'s and } A(i,j)\text{'s are known constants} \\
 & \text{all functions are linear.}
\end{array}
$$

There are certain assumptions that must apply for linear programming. These assumptions are linearity, additivity, and divisibility.

Usually, the input-output relationship is that it follows the law of diminishing returns. Roughly stated this says that if one continually increases the amount of one resource, without changing the amount supplied of other resources, the process will eventually result in a reduction of the product. This does not fit the assumption of linearity, which states that for a given amount of resources used as input the result will be in a given amount of output produced. Therefore, a linear function must be found that most closely represents the actual function.

The assumption of additivity is the assumption that the production of one product will not effect the production of another product. If the amount of one product produced is increased, it will not aid or restrict the production of another product. Put another way, the total is the sum of its parts. As simple as this sounds it is not always the case as one input will often cause another input to be more or less effective.

Linear programming assumes divisibility, that is that all of the resources and products can be used or produced in fractions of units. This results in linear programming giving answers such as: 1.54 units of labor, or 3.2 units of a product, etc. This may or may not be a problem, but, it will always need to be considered when interpreting the results of linear programming.

History of Computers and Linear Programming

The systematic development of practical computing methods for linear programming began in 1952 at the Rand Corporation in Santa Monica, under the direction of George B. Dantzig. (13) It is not just a coincidence that work and advancements in linear programming run parallel with those of the computer industry. Because of the complexity and 'number crunching' involved in linear programming there is simply no way that problems of any size could be done quickly enough or with suffecient accuracy without the aid of computers.

The beginnings of linear programming could be seen as early as 1939, when L. V. Kantorovich of the Soviet Union did work that went highly unnoticed until 1976, when he received a Nobel Prize. In 1947 the U. S. Air Force began to study planning and scheduling problems. In 1947-1948, Dantzig developed a simplex algorithm for optimization of a linear form, subject to linear inequality constraints, i.e., linear programming. Computerizing the algorithm was started in 1952. By the end of the 1950's most of the mathematical and computer problems were solved.

Linear programming still had a problem with overlapping I/O, or the transmission of data to and from auxiliary storage while computations were on going. This occurred because the computers of the day were not large enough to hold all the data and the program code in the main storage. This problem was solved for linear programming in the 1960's as it was also solved for computers in general, larger computers were introduced and then were multiprogrammed.

In the late 1950's and early 1960's CEIR, Inc. in Washington, D.C. began to work on large linear programming packages and systems, at this time all work was done on IBM computers. Simultaniously the Rand Corporation initiated development of a set of FORTRAN programs for linear programming. The introduction of IBM computers with 8K of core memory allowed packages to be enhanced to handle over five hundred rows (equations). This allowed linear programming solutions to be produced for regular production work, and was used mostly in the petroleum industry. At this time each algorithmic program existed as a deck of punched cards in absolute format whose loading had to be done by hand. As late as 1965 it was still taking up to a day to run a linear programming solution.

The major linear programming systems underwent significant changes and extensions during the mid-1970's, taking on their present and perhaps their final form for mainframe computers. Even though the form has remained constant for some time, there is still work being done on system speed, reliability, supporting data management and control, and application techniques.

With the increase in use and affordability of micro-computers there has been a lot of work being done to bring linear programming to micro-computers. It might seem that linear programming has no chance of being implemented successfully on micro-computers because of the memory requirements, however the present machines have memories and are much faster than the mainframes that the first packages were installed on.

Present Capabilities

At present time there are several linear programming packages for large mainframe computers. These are often referred to as mathematical programming systems (MPS).

Although there has been extensive research and testing done at the University level, there are very few packages available commercially. Four such available packages are described below.

One package available in Applesoft is a program called System Solver. (5) This is one of the more powerful packages available in that it allows the user to solve a system with a size of up to seventy equations and seventy unknowns. With System Solver the user must put all equations, which must be linear, in the given form:

```
A(1)X(1) + A(2)X(2) + ... + A(N)X(N) = B
```

The program will first ask for the number of equations and then it will ask for the coefficients (A values) of each equation. The program will file the result by listing the varibles and their computed values i.e., X(1) = 1.

"What's Best" is the name of a linear programming package for micros developed by General Optimization Inc. (8) What's Best consists of a 6K-byte memory-resident module that is used with Lotus 1-2-3 and a separate FORTRAN program that performs the calculations. The program uses Lotus 1-2-3 as an input mechanism and guides the user through the process of building models. The optimizer is the same as that in Lindo, a mainframe linear programming package used at some universities. When the user invokes What's Best he unloads Lotus 1-2-3, invokes and runs the optimizer and then reloads Lotus 1-2-3 to insert the answer into the worksheet.

Another linear programming package for micro-computers is LPMaster, developed and marketed by Applied Operations Research of Canoga Park, California. (17) Unlike the previous packages, the input is logical in nature instead of mathematical. The different sections of the input are separated by control cards. The first section is the definition section which will define the resources to be used and the products to be produced. The second section is the data section, it is used to enter the known information such as gross profit, production cost, etc. This is followed by "compile", "solve", and "report" cards. The input is in a file format, whereas System Solver had an interactive input format, and "What's Best" used Lotus 1-2-3 as the input mechanism.

MPS-PC is a linear programming system designed by Dr. George H. Pfeiffer and marketed by Research Corporation. It is designed to emulate MPS-X, for mainframe computers, on the IBM Personal Computer and IBM compatible personal computers. (14) This system will allow up to seventy activities and fifty constraints to be entered. An input file is used for MPS-PC in a similar format as the LPMaster system. The file is divided into three sections. The first is the rows or activity section, the second is the columns or constraint section, and the third is the RHS (right hand side) or limits section.

There has also been work done on linear programming systems that are not available commercially. One such system was developed by Robert D. Conte. (3) This system consists of several analysis techniques, including linear programming; it is implemented on an Apple II micro-computer in Applesoft. This interactive package was designed and implemented with consideration for the non-programmer oriented user. The linear programming portion of the system is capable of solving problems of a size of up to twenty constraints and twenty variables. It also has extensive editing features, allowing the user to respecify various parameters.

In the TIMS/ORSA meeting in Detroit, Michigan during April, 1982, there were four micro-computer based linear programming systems. The first was developed by Ralph W. Swain. (15) Its primary purpose was that of graphical demonstration of techniques commonly used in operations research. Rolf A. Daininger developed and implemented an instructional aid which will display the iterations tables for a maximum of nine constraints and twenty variables. (4) This allowed students to concentrate on the simplex methodology and solution process, rather than the numeric operations involved. A third system was presented by Gary

E. Whitehouse and Yassar A. Hosni is a package involving forty-two problem oriented programs, several of which involve linear programming. (17) Versions of this system are available for both the Apple II and the TRS-80 micro-computers. One other system was presented by Byron Gottfried. (9) There are two versions available, one for the Apple II and one for the IBM micro-computer, both of which are capable of solving problems of a size of about forty-five constraints and ninety variables.

There has also been some thesis work done on the subject of linear programming. Possibly the most powerful linear programming package available on micro-computers was developed by Theodore R. E. Fraley and Dale A. Kem, at the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio. (7) This is a FORTRAN package supported by USCD Pascal Operating System on an Apple II plus computer. This package is capable of handling up to twenty contraints and twenty variables.

Another package was developed at the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio by M.L. Mullennex. (12) This package was developed on the Apple IIe micro-computer, and written in UCSD Pascal and operates on the Pascal operating system.

The packages presented at the Detroit TIMS/ORSA meeting and the Air Force theses all have similar data entry modules. They are all menu driven. The data entry is one of two methods. Some of the systems used a series of questions asked by the computer which are to be answered by the user. Others used formulas to enter the data, actually typing in mathmatical equations when prompted by the computer. This requires the user to format the data into a formula before he/she may enter the data.

Chapter 2: Definition of Problem

There are a few problems involved with linear programming. One problem common with linear programming systems on micro-computers is that of data entry. Of the systems available, most have the data entered by specifying the equations. The other common form of data entry is to give the constraints, variables, and limits of the problem in lists. Both of these methods are considered awkward as the common method of defining the problem is to put it in matrix form, which requires that the information must be converted to formulas or lists.

Another problem common to linear programming systems is one of accuracy. The problem seems to surface quite often when the system is required to execute several iterations. With the micro-computer size limits, values will often have to be rounded. Even though this rounding is slight, when it is compounded with each iteration of the matrix, this can result in a noticable error can result. A problem also exists with the readability of the output of the systems. Some systems give the results as a list of numbers and the user must interpret them by knowing the format and order of the input.

Objective

The objective of the project is to design, implement and validate a program that will allow a user to use the Multi-Plan spread-sheet as the data entry module to a linear programming system. The user will enter the name of each column in the first row of the spread sheet and enter the name of each row in the first column. This will accomplish two things: 1). the user will be able to use the title option to enter the data, and 2). provides the names that are needed for input into the system.

It was decided to use the MPS-PC Linear Programming System, developed by Dr. George H. Pfeiffer. This system was chosen for a variety of reasons. MPS-PC is one of the most powerful systems available for micro-computers, allowing seventy column activities and fifty row constraints. This system was written for the IBM Personal Computer and IBM compatible computers. The large availability and usage of these personal computers makes this system a desirable target for the project.

The MPS-PC system was designed to appear the same as the MPS-X system available for large mainframe computers. Since MPS-X is a popular system for mainframes and with a large portion of people using linear programming being familiar with MPS-X, it follows that MPS-X would be a good system to imitate.

Chapter 3: MPS-MP Program

A program was developed as an instrument to convert a Multi-Plan worksheet into a data file, which can be used as the input file for the MPS-PC package. The program was called MPS-MP. This program has been placed on both the printer Working Diskette and the Screen Display Working Diskette of the MPS-PC package. This will allow the user to use MPS-MP whether they are printing the results of the MPS-PC program or are just looking at the results on the screen. By putting the MPS-MP program on the Working Diskettes, no new diskettes are required, other than the Multi-Plan diskette when this operation is compared to just using the MPS-PC program.

In designing the MPS-MP program there were two tasks that needed to be accomplished. The first task was to read and store the data in such a way as the rest of the program can understand and use the data. The second task is to take the stored data and write it to a data file in the proper format that the MPS-PC program can use it.

The first step in writing the program was to determine how the worksheet is written on the diskette when it is printed as text to a file. The PRINT FILE function of Multi-Plan is used to write the worksheet to the diskette. The same data diskette is used to write this file as is used for the data diskette of the MPS-PC program. When

Multi-Plan writes to the diskette the resulting text file will list all the row values of seven columns, then produce a page break, the number of rows may also cause a page break. Then, on the next page, all the rows of the next seven columns will be printed, and so on until the entire worksheet has been printed to the file.

Each page will begin with a series of blank lines, followed by the first row of the seven columns that are to be printed on the page and then the second row of the seven columns are printed, etc. Each line ends with a carriage return followed by a line feed character, these two characters are all that make up a blank line. These characters are placed immediately after the last (rightmost) non-blank character of each line. This results in the blank lines from a page break looking the same as blank lines coming from a row with no values for the seven columns that are to be printed on that page. Each line that is not a blank line begins with a series of blank characters.

It was decided that the file should be read and evaluated a character at a time. An attempt was made to read a field, i.e., one row one column, at a time but the carriage return and line feed characters ended up in the fields and caused other problems. Reading a character at a time required the program to keep track of several values so as to know where to store the character. Some of the values that it is neccessary to keep track of are listed below:

1) The number of pages read
2) The number of lines read for the page
3) The number of columns read for the row
4) The number of characters read for the field

The main process of the program is used to communicate with the user. Procedures are called from the main process to accomplish the two major tasks. Reading and storing the data is accomplished in the procedure Build-Table, while writing the data file is done in the Write-Output procedure. Write-Output calls a variety of other procedures and functions to find the values that are needed to write the output correctly.

A two dimensional array of ten character strings was created to store the worksheet in memory the same way that it appeared in the worksheet itself. This needed to contain fifty-two rows to accommodate the column headings, objective function row and the fifty rows allowed by the MPS-PC program. It contains seventy-three columns to hold the row headings, restriction types, right hand sides and the seventy columns that are allowed by the MPS-PC program.

To keep track of where the incoming data belongs, repeat loops were used in the Build-Table procedure. One repeat loop is used to read through the blank lines at the first of the first page. The loop will read until a blank character is encountered. Since a blank line has only a carriage return and a line feed character in it, a blank character means that the line has data. A second repeat loop is used to read until the end-of-file is read, this insures that all the data is read. One level inside this repeat loop are two other repeat loops. One is used to read to the end of the line. Inside this loop is a repeat loop that reads one field. The second loop inside the end-of-file loop is used to read from the end of one line until another line is found that has data on it.

Each loop has a counter that coordinates with it. One keeps track of the characters in a field. Another keeps track of the fields in a line. A third keeps track of the lines on the page. The fourth keeps track of the blank lines between lines with data. The last keeps track of the pages read. The different counters are used as subscripts to assign the character read to its proper place in the array.

A problem arose with telling the difference between a blank line created from a row with no data on it and a blank line created from page breaks, both may be several blank lines back-to-back. As a result of this, the loop that reads from one line with data to another line with data, will treat all blank lines that are read as rows without data and not as a page break. This means that when a line with data is read the program must be able to analyze this data and decide whether the blank lines were a page break or not. The program can tell if the blank lines were a page break by whether the first non-blank character of the line is a number or not. If it is a number the blank lines were rows without data. Since the column headings cannot begin with a numeric character, if the first character is non-numeric the program knows the blank lines were a page break and can reset the row count to one and add seven to the column count. This sets the row and column counters for a new page.

Once the worksheet data has been recreated in the program the Write-Output procedure and the different procedures and functions that it calls become fairly straight forward. The number of rows are found in the Number-Of-Rows function by searching through the first column of the array and counting until a blank field is encountered. The first column is for row headings which must have a value in them. Row two is not counted as a blank field as it can be blank because the objective function does not require a row heading.

The number of columns in the problem are found in a function called Number-Of-Columns. The first row is used as column headings. Therefore, the number of columns can be and is found by searching the first row until a blank field is located and counting just as with the rows. If row two is blank this does not stop the counting as this row holds what type each restriction and does not need a heading.

The number of each of the types of restrictions (less thans, greater thans, equal tos) are found by calling a procedure called RHS-Totals. In this procedure row two is examined. The number of L's, G's and E's are counted and this is the number of less thans, greater thans and equal tos respectfully.

To actually write the output to a disk file a series of for loops are used. Each is used to print the different portions of the array in the order that is needed for the data file for MPS-PC to work.

Chapter 4: Evaluation

There are a few limitations to the Multi-Plan worksheet. One major limitation when using it for the purpose of creating a matrix for MPS-PC is the fact that Multi-Plan gives the user only sixty-three columns to. This is more restrictive than the seventy columns that are allowed by the MPS-PC program. There is a way to get around this restriction but it is not very convenient. The user must enter the first sixty columns of the problem. The first two rows are the row headings and restriction types followed by the sixty columns, with column sixty-three being reserved for the right hand sides. Then the user can run the conversion program just as if it were a normal problem. When this is done the user needs to get into the MPS-PC program and add additional columns.

There are also limitations to the conversion program. The program requires that all of the Multi-Plan default values remain when entering the problem on the Multi-Plan worksheet. This means that the user may not make adjustments in Multi-Plan to enter larger headings or field values. This is also a restriction from the MPS-PC program, as it allows only eight significant digits and headings of only eight characters. The MPS-MP program assumes that the data has been entered correctly in the Multi-Plan worksheet, see appendix A for instructions.

MPS-MP will not edit the input data to insure that it is in the correct format. If it is not the program will write its output file as if the data was in the correct format. This means that the problem will not show up until the MPS-PC program is run. Another limitation to the MPS-MP program is that it assumes that the user already knows how to use Multi-Plan and MPS-PC. It does not help the user with these programs.

Testing MPS-PC

A series of linear programming problems were used to test the MPS-PC program. Each of the twenty-seven problems were run first on the MPS-PC program and then on the MPS program found on the mainframe computer in Cardwell Hall, to be used as a control. The MPS program on the main-frame computer is readily accepted as a good and accurate linear programming program. Each problem run on the MPS-PC program was compared to its corresponding run on the MPS program. The results of these comparisons were then analyzed using three different methods to group the problems, see Figure 1. In the first method the results were grouped depending on the number of rows that were present in each of the problems. In the second method the problems were grouped according to how many columns were present in each of the problems. In the third and final method the

CRITERIA	PROBLEMS	OBJFCNS/ AVERAGE	ROWS/ AVERAGE	COLUMNS/ AVERAGE
1-10 rows	9	1 22.00%	2 13.27%	1 12.76%
11-25 rows	10	2 23.86%	2 27.14%	2 13.18%
> 25 rows	8	1 0.41%	1 1.44%	1 13.88%
1-15 columns	10	1 3.86%	2 10.42%	1 7.67%
16-30 columns	9	1 22.00%	1 9.63%	1 12.76%
> 30 columns	8	2 3.30%	3 29.47%	3 20.97%
1-10 itertns	8	0	1 4.09%	0
11-20 itertns	9	2 26.28%	3 20.25%	3 27.01%
> 20 itertns	10	2 20.33%	2 21.16%	2 16.61%

Figure 1. Analysis of Problems Run on the MPS-PC Program

problems were grouped according to the number of iterations that were necessary for the MPS-PC program to solve each of the problems.(*)

For each grouping the problems were evaluated on three different criteria. The first criteria is any difference in the value of the objective function of the problem when it was run on the MPS-PC program and the objective function of the problem when it was run on the MPS program. The second criteria is the average difference of the row activities. This entailed figuring the difference of each row activity and then taking an average of all the row activities. The third criteria is the average difference of the column activities. This entailed figuring the difference of each column activity and then taking an average of all the column activities.

Nine problems were tested that had ten or fewer rows in them. The solution to one of these nine problems produced values that were different with regards to the objective function, the row activities, and the column activities, between the runs on the MPS-PC program and the MPS program. The objective function differed by 198%. The row activities differed by an average of 86.67% and the column activites differed by an average of 114.82%.

* A linear programming problem is solved by continually dividing the matrix by determined values until a solution is reached. Each time the matrix is divided it is considered an iteration of the matrix.

Another problem had row activities that differed even though the objective function and column activites were the same. The average difference of these row activities for this problem was 32.74%.

With between eleven and twenty-five rows, there were ten problems tested. Of the ten problems, two differed in the objective function, row activities and the column activities. One problem had a difference in the objective function of 38.55%, an average difference in the row activities of 71.43% and an average difference in the column activities of 76.68%. The other problem with bad values had differences of 200% in the objective function, 200% average difference in the row activities and 55.1% difference in the column activities.

There were eight problems with twenty-six or more rows in the problem. Of these eight one had different results in the objective function, row activities and column activities. The objective function differed by 3.3%, the row activities differed by and average of 11.55% and the column activities differed by an average of 111.02%.

The problems were also divided and analyzed according to the number of columns in the problem. Ten problems with less than sixteen columns were tested. One of these gave erroneous results for all three areas. The objective function was off by 38.55%. The row activities were off by an average of 71.43% and the column activities were off by an average of 76.68%. Another problem had row activities that were off by an average of 32.74%.

For the range of sixteen to thirty columns there were nine problems. In this range, one problem gave different results when run on MPS-PC as opposed to when it was run on MPS. Its objective function differed by 198%, its row activities differed by an average of 86.67% and its column activities differed by an average of 114.82%.

The last range of column sizes tested was problems with more than thirty columns. In this group there were a total of eight problems. Two of these problems had differences in the three areas being tested and one problem had a difference in the row activities, while yet another problem had differences in the column activities. Of the two problems first mentioned, the objective functions differed by 200% and 3.3%. The average differences of the row activities were 200% and 11.55%. With the average differences of the column activities being 55.1% and 111.02%. The problem with only the row activities off, was off by an average of 24.18%. The problem with only the column activities off, was off by an average of 51.61%.

A final method of dividing the problems was used to analyze the problems. This method was to divide the problems depending on the number of iterations required by the MPS-PC program to solve the problem. The first range analyzed was for less than eleven iterations. In this group one problem gave row activities that were an average of 32.74% different than when run on MPS. All other problems gave correct solutions.

For the range of eleven to twenty iterations there were nine problems. Two of the nine gave different results for their objective functions, row activites and column activities. One problem gave different results for just the row activities and one gave different results for just the column activities. The two objective functions were off by 38.55% and 198%. The three row activities were off by averages of 71.43%, 86.67% and 24.18%. The three column activities were off by averages of 76.68%, 114.82% and 51.61%.

There were ten problems with over twenty iterations needed to solve the problem. Of the ten, two had variances between the MPS and the MPS-PC runs. The first had a 200% variance in the objective function, an average of 200% variance in the row activities and an average of 55.1% variance in the column activities. The second had a 3.3% variance in the objective functions, an average of 11.55% variance in the row activities and an average of 111.02% variance in the column activities.

When all twenty-seven problems were analyzed together there were seven problems with some form of difference. Four had objective function differences, giving an average of 12.59% difference for the entire twenty-seven. Six problems had row activities that differed. This gave the entire group an average difference for the row activities of 15.8%. Finally, five problems had column activities that differed, giving the group an average difference in the column activities of 15.16%.

Chapter 5: Conclusions

There are seven of the twenty-seven problems, or 25.93%, giving some form of erroneous answer when run on the MPS-PC program if the MPS program is considered accurate. This is much too large of a number to accept the MPS-PC program as an accurate linear-programming solution program. With four of the twenty-seven, or 14.81%, giving bad objective functions the program can not even be relied on for this value.

The test data was divided and analyzed by different criteria, number of rows, number of columns and number of iterations, in an attempt to see if the size of the problem had anything to do with the reliability of the results. As shown earlier, the size of the problem makes little or no difference in the reliability of the program. Therefore, the program must be rejected for use on any size of problem.

The program does, however, have some usefulness. For several reasons the program may be used as an effective teaching tool. The program now has the capability to be loaded from Lotus 1-2-3 or the Multi-Plan worksheet. This becomes more and more advantageous as more students learn to use computers and these software packages. The MPS program is a very popular linear programming package for mainframe computers. This and the fact that the output from MPS and the output from MPS-PC look a lot alike, MPS-PC can be used to get students use to reading the output without having to take the extra time and effort to use a mainframe computer.

The MPS-MP conversion program was tested for conversion time. In this test the two different times were tested. The time for the program to read the input file that was created by Multi-Plan was tested and the time for the program to write the output file was also timed. The longest reading time was fifty-nine seconds, within an acceptable length of time. The longest output time was nine seconds, again within an acceptable length of time. These two times occured in a problem which had fifty rows and seventy columns, the maximum allowed for both.

Future Study

Since the MPS-PC program is not reliable, the need for a reliable linear programming package for micro-computers still exists. In examining the text problems, often the number of iterations required for the MPS-PC program to solve the problem was different than the number of iterations needed by the MPS program. This implies that the two use different methods to solve the problem. One area of future study, therefore, could be to find the algorithm that the MPS program uses and adapt it to the micro-computer.

Another consideration for future study is to examine the possibilities of expanding the conversion program to be more powerful. The program could recognize when a duplicate row or column name has been entered, it does not make the check at this time, and allow the user to make the necessary corrections in MPS-MP instead of returning to Multi-Plan. MPS-MP could also be changed to allow the user to add columns or rows without either going to Multi-Plan or MPS-PC.

The need for a program that could convert a worksheet, whether it be Multi-Plan or not, to an input data file to the MPS program would be nice also. This is true because many businesses and universities are investing in software that will allow the user to transfer files from a diskette to a main-frame's memory. Kansas State University posseses such a package, Kermit. This could be used until a good, reliable linear programming package can be created for micro-computers.

Appendix A:

MPS-MP Instructions

This program is a utility program designed to allow the user to create linear programming matricies for the MPS-PC linear programming system using the Multi-Plan worksheet. While the MPS-MP program does not do anything that the MPS-PC program can not do, in fact it will only allow the user to create the initial matrix. To add, subtract or change anything in the matrix the user must go back to the Multi-Plan worksheet or go on to the MPS-PC program. These instructions do not presume to teach the user how to use the Multi-Plan or the MPS-PC programs. These programs have manuals of their own and it is left to the user to find and use these manuals.

Creating the Matrix

The matrix to be created on the Multi-Plan worksheet is identical in format to the TRANCOL matrix tableau printout available with MPS-PC. When using Multi-Plan to create a matrix, all default values for the worksheet are to be used. This means that once Multi-Plan is entered the user cannot change any values, only enter the matrix as directed.

The matrix should be placed in the upper left hand corner of the worksheet. This means that the constraint (row) names should be listed down column one, beginning in column three. The activity (column) names should be listed across row one, beginning in column three. The row and column names may be any combination of characters of up to eight characters in length. The first character of each name must be a letter or a special character (!,@,#,$,%,&). The rest of the characters must be a letter, one of the special characters, a number or a hyphen. No blank row or columns names are allowed. The names cannot be 'END' either. Either "min" for a minimizing problem or "max" for a maximizing problem is placed in cell (1,1), row one column one.

The objective function values of each column are listed across row two beginning in column three. The objective function value for each column obviously goes in that column. Zero values may be represented with blank cells. If the value is negative this is represented by placing a minus sign before the value.

The nature of each constraint (less than, equal to, or greater than) should be entered in column two beginning in row three. The nature is listed in the same row as the constaint that it pertains too. An "L" is used for less than, "E" for equal to and a "G" for greater than. These are to be left justified in their respective cells.

The right hand side values are listed in the column after the last activity column. "RHS" is written in row one of this column. The second row of the column is blank as the right hand side has no objective function value.

Then the right hand side values are listed starting in row three and in the row that the value pertains too. Zeros may be represented by blank cells. Negative values are represented by a minus sign before the value.

The constraint set values (matrix values) are entered after column three and below row three. Each value is placed in the cell corrisponding to the row and column that pertains to the value. If the value is zero the cell may be left blank. If it is a negative value this is indicated with a leading minus sign.

Saving the Matrix

There are two ways that the matrix should be saved. The first method is to save the matrix so that it may be recalled for changes. This is done using the TRANSFER SAVE instructions. The second method is used to save the matrix as a test file so that the MPS-MP program can read it. This is done by using the PRINT FILE instructions.

Using MPS-MP

To use the MPS-MP program the MPS-PC working diskette or the MPS-PC screen display working diskette with the MPS-MP program on it is inserted into drive A, while a formatted data disk is inserted into drive B. When the A: prompt appears on the screen the user types:

```
A:MPS-MP
```

and presses return or enter. This will initiate the program. The program will then display:

```
Enter the name of the file to be converted.
```

The user will then type the file name that was saved from Multi-Plan using the PRINT FILE instructions. If the file-name entered is not on the data disk the following appears on the screen:

```
<filename> is not on the disk.
Enter "Q" to exit to system.

Enter the name of the file to be converted.
```

with <filename> being the filename that the user just entered. If the user enters "Q" the program exits the system. Anything else entered will be interpreted as a filename and the program will treat it the same as the file name entered earlier.

When a valid filename is entered, the program will read the file from the data disk in drive B. Then program will then prompt the user to name the output for MPS-PC to use by displaying:

```
Enter filename for MPS-PC.
Enter an 'X' for the default.
Default will be "filename".
```

with "filename" being the file that was read by MPS-MP. When entering the filename the user does not give the .dat extension. Upon enter the filename or 'X' the program will allow the user to change the data disk in drive B by displaying:

```
Put the data disk in drive B:
Press any key to continue.
```

The user places the desired disk in drive B and presses any key. At this point the program creates and saves the output file (data file for MPS-PC). When this is done the program will ask the user if he/she wishes to convert another file by displaying:

```
Do you wish to convert another file? Y/N
```

If "Y" is entered the program will start over with the enter file to be converted prompt. If "N" is entered the program will exit to the system.

The program will tell the user if the data disk is full by displaying:

```
Disk is full.

Do you wish to convert another file? Y/N
```

At this point the user may place another formatted data disk in drive B and convert the file again or exit to the system by entering "Y" or "N" respecfully.

Appendix B:

MPS-MP Program Listing

```
program mps_mp (filein, fileout);

type
    cell_type  = string[10];
    table_type = array [1..53,1..73] of cell_type;
    line_type  = array [1..7] of cell_type;

var
    done,
    dummy        : char;
    filename     : string[8];
    infilename   : string[10];
    outfilename  : string[8];
    fileinname   : string[14];
    fileoutname  : string[14];
    table        : table_type;
    filein       : text;
    fileout      : text;
    i,j          : integer;
    filefound,
    too_large    : boolean;

(*********************************************************)
(*      a  procedure  to make all letters  capitals     *)
(*********************************************************)

procedure capitalize(rows, columns: integer);

var
    i,j,k: integer;

begin
    for  i  :=  1  to  rows  do {capitalize row headings}
        for  k  :=  1  to  10  do
            if  table[i,1][k]  in  ['a'..'z']  then
                table[i,1][k]  :=
                    chr(ord(table[i,1][k]) -32);
    {capitalize column headings}
        for  j  :=  1  to  columns  do
            for  k  :=  1  to  10  do
                if  table[1,j][k]  in  ['a'..'z']  then
                    table[1,j][k]  :=
                        chr(ord(table[1,j][k]) - 32);
end; {capitalize}
```

```
(**********************************************************)
(* a function to find the number of rows read  from file *)
(**********************************************************)

function  number_of_rows : integer;

var
    i : integer;

begin
    i := 3;
    (**  read through column one until blank cell read  **)
    while table[i,1] <> '            ' do  begin
        i  :=  i  +  1;
    end;{while}
    number_of_rows  :=  i - 1;
end; {number_of_rows}


(**********************************************************)
(*    a function to find the number of columns read in   *)
(**********************************************************)

function  number_of_columns : integer;

var
    i : integer;

begin
    i  :=  3;
    (**  read table until blank column heading read  **)
    while table[1,i]  <>  '            '  do  begin
        i := I + 1;
    end; {while}
    number_of_columns  :=  i - 1;
end; {number_of_columns}
```

```
(**********************************************************)
(* a procedure to determine the number of less thans,     *)
(* equal tos, and greater than constraints               *)
(**********************************************************)

procedure rhs_totals
    (var lessthans,equals,greaterthans,rows: integer);

var
    i : integer;

begin
    lessthans  :=  0;
    equals  :=  0;
    greaterthans  :=  0;
    (**  read row 2 thru end of column two  **)
    for  i  :=  3  to  rows  do  begin
        case  table[i,2][1]  of
            'L' : lessthans  :=  lessthans + 1;
            'E' : equals  :=  equals + 1;
            'G' : greaterthans  :=  greaterthans + 1;
        end; {case}
    end; {for}
end; {rhs-totals}
```

```
(**********************************************************)
(* a procedure to read the input file and create a table *)
(* in memory the same as in the multi-plan spread sheet. *)
(**********************************************************)

procedure build_table;

var
    filler : char;
    row,col,column,i : integer;
    last_line_blank : boolean;

label  error;

begin
    row  :=  1;
    column  :=  1;
    last_line_blank  :=  false;

    repeat  { read beginning blank lines }
        read(filein,filler);
    until  filler  =  ' ';

    repeat  { read entire file }
        col  :=  column  -  1;

        for  i  :=  1  to  4  do  {read blanks}
            read(filein,filler);

        repeat  { read one line }
            i  :=  0;
            col  :=  col  +  1;
            if  col  >  72  then  begin
                writeln;   writeln
                    ('Too many columns, please correct.');
                too_large  :=  true;
                goto  error;
            end; {if}

            repeat  { read one field }
                i  :=  i  +  1;
                read(filein,filler);
            if  (i  =  1)  and  (last_line_blank)  and
                ((filler in ['a'..'z']) or
                (filler in ['A'..'Z'])  or
                (filler in ['!','@','#','$','%','&']))
                then  begin  { new page }
                    row  :=  1;
                    column  :=  column  +  7;
                    col  :=  column;
            end; {if}
            if  row  >  52  then  begin
                writeln;   writeln
                  ('Too many rows, please correct.');
                too_large  :=  true;
                goto  error;
            end; {if}
            if  (ord(filler)  <>  13)  and
                (ord(filler)  <>  10)  then
                    table[row,col][i]  :=  filler;
            last_line_blank  :=  false;

        until  (ord(filler) = 13)  or  (i = 10)  or
               (eof(filein));
    until  (ord(filler)  =  13)  or
           (col  =  column + 6)  or  (eof(filein));
    if  (col = column + 6)  and  (i = 10)  then
        read(filein,filler);
    i  :=  1;
    row  :=  row  +  1;
    repeat  { read through blank lines }
        read(filein,filler);
        if  ord(filler)  =  13  then  begin
            row  :=  row  +  1;
            i  :=  i  + 1;
        end;
    until  (filler  =  ' ')  or (eof(filein));
    if  i  >  1  then
        last_line_blank  :=  true;
  until  eof(filein);
  error:  writeln;
end;
```

```
(*********************************************************)
(*  a procedure to write the output file MPS-PC needs.   *)
(*********************************************************)

procedure write_output(var fileout:text);

var
    disk_full : boolean;
    rows,
    columns,
    lessthans,
    equals,
    greaterthans,
    i, j, k : integer;
    rhs : string[8];
    label error;

begin
    disk_full  :=  true;
    rows   :=  number_of_rows;
    columns := number_of_columns;
    capitalize(rows,columns);
    rhs_totals(lessthans,equals,greaterthans,rows);
    (***  write number of constraints, activities, **)
    (***  less thans, equal tos, and greater thans **)
    {$I-}
    write(fileout, rows - 2,      ','  ,
                   columns - 3,   ','  ,
                   lessthans,     ','  ,
                   equals,        ','  ,
                   greaterthans,  ','  ,   '"');
    if  not (IOresult = 0)  then
        goto  error;

    (**  write type of problem  **)
    for  k  :=  1  to  3  do  begin
        write(fileout,table[1,1][k]);
        if  not (IOresult = 0)  then
            goto  error;
    end; {for}

    writeln(fileout,'"');
    if  not (IOresult = 0)  then
        goto  error;
```

```
(**  write constraint type of each row  **)
for  i  :=  3  to  rows  do  begin
    write(fileout, '"');
    if  not (IOresult = 0)  then
        goto  error;
    for  k  :=  1  to  10  do
        if  not  (table[i,2][k]  =  ' ')  then  begin
            write(fileout,table[i,2][k]);
            if  not (IOresult = 0)  then
                goto  error;
        end; {if}
    writeln(fileout, '"');
    if  not (IOresult = 0)  then
        goto  error;
end; {for}

(**  write row names  **)
for  i  :=  3  to  rows  do  begin
    write(fileout, '"');
    if  not (IOresult = 0)  then
        goto  error;
    for  k  :=  1  to  10  do
        if  not  (table[i,1][k]  =  ' ')  then  begin
            write(fileout,table[i,1][k]);
            if  not (IOresult = 0) then
                goto  error;
        end;  {if}
    writeln(fileout, '"');
    if  not (IOresult = 0)  then
        goto  error;
end; {for}

(**  write column names  **)
for  i  :=  3  to  columns - 1   do  begin
    write(fileout, '"');
    if  not (IOresult = 0)  then
        goto error;
    for  k  :=  1  to  10  do
        if  not  (table[1,i][k]  =  ' ')  then  begin
            write(fileout,table[1,i][k]);
            if  not (IOresult = 0)  then
                goto  error;
        end;  {if}
    writeln(fileout,'"');
    if  not (IOresult = 0)  then
        goto  error;
end; {for}
```

```
(**  write objective function values   **)
for  i  :=  3  to  columns - 1  do
    if  table[2,i]  =  '          '  then begin
        writeln(fileout,0);
        if  not (IOresult = 0)  then
            goto  error;
    end  {if}
    else  begin
        for  k  :=  1  to  10  do
            if  not (table[2,i][k]  =  ' ') then begin
                write(fileout,table[2,i][k]);
                if  not (IOresult = 0)  then
                    goto  error;
            end;  {if}
        writeln(fileout);
        if  not (IOresult = 0)  then
            goto  error;
    end; {else}

(**  write matrix values  **)
for  i  :=  3  to  columns - 1  do
    for  j  :=  3  to  rows  do
        if  table[j,i]  =  '          '  then  begin
            writeln(fileout,0);
            if  not (IOresult = 0)  then
                goto  error;
        end  {if}
        else  begin
            for  k  :=  1  to  10  do
                if  not (table[j,i][k]  =   ' ')
                then begin
                    write(fileout,table[j,i][k]);
                    if  not (IOresult = 0)  then
                        goto  error;
                end;  {if}
            writeln(fileout);
            if  not (IOresult = 0)  then
                goto  error;
        end; {else}
    (**  write right hand side values  **)
    for  i  :=  3  to  rows  do
        if  table[i,columns]  =  '          '  then  begin
            writeln(fileout,0);
            if  not (IOresult = 0)  then
                goto  error;
        end  {if}
        else  begin
            for  k  :=  1  to  10  do
                if  not (table[i,columns][k]   =   ' ')
                then begin
                    write(fileout,table[i,columns][k]);
                    if  not (IOresult = 0)  then
                        goto  error;
                end;  {if}
            writeln(fileout);
            if  not (IOresult = 0)  then
                goto  error;
        end; {else}

    writeln(fileout, '"END"');
    if  not (IOresult = 0)  then
        goto  error;
    disk_full  :=  false;

    error:  (*** disk write error occured  ***)
    if  disk_full  then  begin
        writeln;
        writeln;
        writeln('Disk is full.');
    end;  {if}

end; {write_output}
```

```
(**********************************************************)
(**************         MAIN PROCESS       ****************)
(**********************************************************)

begin
    (**  loop until user wants to return to system  **)
    repeat
        done  :=  'N';
        too_large  :=  false;
        for  i  :=  1  to  53  do
            for  j  :=  1   to  73  do
                table[i,j]  :=  '         ';

        (**  loop until user enters a valid filename **)
        repeat
            writeln
           ('Enter the name of the file to be converted.');
            readln (filename);
            fileinname  :=  'b:'  +  filename;
            assign(filein, fileinname);
            {$I-} reset(filein) {$I+};
            if  IOresult  =  0  then
                filefound  :=  true
            else
                if  (filename = 'q')  or  (filename =  'Q')
                    then  filefound  :=  true
                else  begin
                    filefound  :=  false;
                    clrscr;
                    writeln
                        (filename,' is not on the disk.');
                    writeln
                        ('Enter "Q" to exit to system.');
                    writeln;
                end {else}
       until  filefound;
```

```
        if  not ((filename = 'q')  or  (filename = 'Q'))
        then begin
            build_table;
            if  not  too_large  then  begin
                writeln;
                writeln('Enter filename for MPS-PC.');
                writeln('Enter an "X" for the default.');
                writeln
                   ('Default will be "', filename, '".');
                writeln;
                readln(outfilename);
                if  (outfilename = 'X')  or
                    (outfilename = 'x') then
                    outfilename  :=  filename;
                fileoutname  :=
                    'b:'  +  outfilename  +  '.dat';
                writeln;  writeln;  writeln
                    ('Put the data disk in drive b:');
                writeln('Press any key to continue.');
                readln(dummy);
                assign(fileout,fileoutname);
                {$I-}  rewrite(fileout) {$I+};
                if  IOresult = 0  then  begin
                    write_output(fileout);
                    close(fileout);
                end  {if}
                else
                    writeln('Disk is full.');
                writeln;
                writeln('Do you wish to convert',
                    ' another file? Y/N ');
                readln(done);
            end;  {if}
        end; {if}
        clrscr;
    until  (done  =  'N')  or  (done  =  'n');
end.
```

BIBLIOGRAPHY

1. Barton, David G. and Francis, Cheryl Parks. "A Users Guide to MPS: A Linear Programming System from IBM," Department of Agricultural Economics, New York State College of Agriculture and Life Sciences, Cornell University, July 1976.

2. Buller, Orlan. "Linear Programming: Notes on Theory, Geometry and Use," Department of Agricultural Economics, Kansas State University, 1985.

3. Conte, Robert D. "Computer Assisted Analysis for Military Managers," Masters Thesis, Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December, 1979.

4. Daininger, Rolf A. "Teaching Linear Programming on a Micro-computer," TIMS/ORSA Meeting, Detroit, Michigan, April, 1982.

5. Darr's, W. "Solving Systems of Linear Equations," Nibble, Vol. 6, No. 1, January 1985, p. 60-62, 64.

6. DBS Corporation, Arlington, Virginia, "Development of Software for Computer Assisted Model Simplification. Final Report," Department of Energy, Washington, D.C., April 25, 1980.

7. Fraley, Theodore R. E. and Kem, Dale A. "FORTRAN Based Linear Programming for Microcomputers," Masters Thesis, Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December, 1982.

8. Gillin, Paul. "Package Eliminates Trials and Errors in 'What If' Models," PC Week, Vol. 4, No. 4, July/August 1985, p. 46.

9. Gottfried, Byron. "Micro-LP: A Microcomputer-Based Linear Programming system," TIMS/ORSA Meeting, Detroit, Michigan, April, 1982.

10. Landis, K. "Micro Finance: Linear Programming," Softalk for the IBM Personal Computer, Vol. 2, May 1984, p. 59-60.

11. Landis, K. and Herbers, M. "Micro Finance: Linear Programming, Part II; LPMasters," Softalk for the IBM Personal Computer, Vol. 3, June 1984, p. 133-135.

12. Mullennex, M. L. "Microcomputer-Based Graphical Linear Programming Package," Masters Thesis, Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December, 1093.

13. Orchard-Hays, W. "History of Mathematical Programming Systems," *Annals of the History of Computing*, Vol. 6, No. 3, July 1984, p. 296-312.

14. Pfeiffer, Dr. George H. "MPS-PC Linear Programming System," Research Corporation, 1984.

15. Swain, Ralph W. "Microcomputers in the Classroom," TIMS/ORSA Meeting, Detroit, Michigan, April 1982.

16. White, G. R. "Personal Computer Aided Decision Analysis", Masters Thesis, Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, December 1984.

17. Whitehouse, G. E. and Hosni, Y. "Use of Microcomputers to Solve Industrial Engineering and Operations Research Problems," TIMS/ORSA Meeting, Detroit, Michigan, April 1982.

Linear Programming on the PC: Loading through Multi-Plan and Testing.

by

Dwight Christie

B. S., Kansas State University, 1983

An Abstract of A Master's Report
submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, KS 66506

1987

Abstract

The developement of linear programming and the development of computers have coincided since L. V. Kantorovich began work on linear programming in 1939. With the complexity and number of steps required to solve a linear programming problem it was not feasable to solve the problem by hand. As computers became larger and faster linear programming programs became more prevalent and reliable.

Now with personal computers with relatively large memories and being quite fast, linear programming packages are being developed for personal computers. MPS-PC is one such package. For this report a conversion program (MPS-MP) was written to allow a user to load the data for MPS-PC through the Multi-Plan worksheet.

The MPS-PC package was tested against a program for mainframe computers (MPS by IBM). The test showed that 25.93% of the problems run on MPS-PC gave errorneous results, if the results from the MPS package are considered accurate.