1986

Goal programming: computational solutions for large-scaled models

Lee Ann Crowder
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd
Part of the Operational Research Commons

Recommended Citation
Crowder, Lee Ann, "Goal programming: computational solutions for large-scaled models " (1986). Retrospective Theses and Dissertations. 7990.
https://lib.dr.iastate.edu/rtd/7990
INFORMATION TO USERS

This reproduction was made from a copy of a manuscript sent to us for publication and microfilming. While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. Pages in any manuscript may have indistinct print. In all cases the best available copy has been filmed.

The following explanation of techniques is provided to help clarify notations which may appear on this reproduction.

1. Manuscripts may not always be complete. When it is not possible to obtain missing pages, a note appears to indicate this.

2. When copyrighted materials are removed from the manuscript, a note appears to indicate this.

3. Oversize materials (maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or in black and white paper format.*

4. Most photographs reproduce acceptably on positive microfilm or microfiche but lack clarity on xerographic copies made from the microfilm. For an additional charge, all photographs are available in black and white standard 35mm slide format.*

*For more information about black and white slides or enlarged paper reproductions, please contact the Dissertations Customer Services Department.
Crowder, Lee Ann

GOAL PROGRAMMING: COMPUTATIONAL SOLUTIONS FOR LARGE-SCALED MODELS

Iowa State University

University Microfilms
International 300 N. Zeeb Road, Ann Arbor, MI 48106
Goal programming: Computational solutions for large-scaled models

by

Lee Ann Crowder

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Co-majors: Statistics Industrial Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

Signature was redacted for privacy.

For the Major Departments

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1986
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>INTRODUCTION: GOAL PROGRAMMING</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>LINEAR GOAL PROGRAMMING</td>
<td>11</td>
</tr>
<tr>
<td>III</td>
<td>DUAL LINEAR GOAL PROGRAMMING</td>
<td>29</td>
</tr>
<tr>
<td>IV</td>
<td>APPLICATIONS</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>A Transportation Problem</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>$L_1$ Estimation</td>
<td>61</td>
</tr>
<tr>
<td>V</td>
<td>SUMMARY AND EXTENSIONS OF GOAL PROGRAMMING</td>
<td>72</td>
</tr>
<tr>
<td>VI</td>
<td>BIBLIOGRAPHY</td>
<td>82</td>
</tr>
<tr>
<td>VII</td>
<td>ACKNOWLEDGEMENTS</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>APPENDIX A:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Section A1</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>Section A2</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>Section A3</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>APPENDIX B:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Section B1</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Section B2</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>Section B3</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>APPENDIX C:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Section C1</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>Section C2</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>Section C3</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>APPENDIX D:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Section D1</td>
<td>173</td>
</tr>
<tr>
<td></td>
<td>Section D2</td>
<td>183</td>
</tr>
<tr>
<td></td>
<td>Section D3</td>
<td>196</td>
</tr>
</tbody>
</table>
I. INTRODUCTION: GOAL PROGRAMMING

The use of linear programming in decision making has been widespread for twenty to thirty years, with its theoretical and solution concepts well-documented and explored. As noted by Sposito (1975), ever since the development of the simplex algorithm by Dantzig in 1947, efforts have been made to improve on the uses and methods of solving linear programming problems in a wide variety of applications: engineering, economics, statistics and business to name a few. Among those who worked to improve on linear programming techniques were A. Charnes and W. W. Cooper (1961). They found that a major difficulty with linear programming is that it is designed to optimize only one objective function. If the user has other objectives, which are possibly in conflict with each other, then somehow these must be incorporated into the linear programming problem as constraints. The one objective function is then optimized over all possible feasible solutions and a resulting optimum is determined. This optimum is the result of an imposed ordering: all constraints are satisfied first with equal importance before the objective is optimized. It is often unclear as to which objective should be chosen as the one to be optimized. To combat this problem Charnes and Cooper (1961), as well as many others since, have explored methods of solving multiple objective problems.

Three basic means of solving multiple objective linear programming problems appear in the literature. One method, which is one specified by Charnes and Cooper (1961), employs a weighting or utility method.
In this method the decision maker tries to express all objectives in terms of one measure by assigning to each objective a weight to indicate its importance with respect to the other objectives. Once these weights are known, the multiple objective problem can be converted into a conventional linear programming problem which can then be solved using any conventional linear programming package. A distinct disadvantage to this method, mentioned by Steuer (1976), is that decision makers often have "fuzzy notions ... about their preferences when asked to express them in fixed point terms" and thus, their ability to decide on correct weights is not very good even when using "disciplined procedures."

One way to avoid the problem of deciding on accurate weights would be to find the total set of efficient or nondominated solutions. A nondominated solution is one in which the vector of solutions to the vector of objective functions is such that at least one member is not dominated by a corresponding member in another solution. This second method of obtaining "optima" to multiple objective systems generates solutions which optimize the utility function of the decision maker without having a utility function explicitly defined. An obvious problem with this method is that the number of efficient solutions is often large, which overwhelms a decision maker who must in some way sift through the total set to find the solution most suitable to his purposes. Also, generating the set of efficient solutions can be impractical and difficult from the computing point of view.

Finally, a third general method for determining solutions for multiple objective systems is one in which the objectives are ranked by
the decision maker according to their perceived importance. This ranking forms a preemptive priority structure in the sense that the ranks assigned to the decision maker's objectives are translated into priority levels where achievement of an objective at a high priority level is immeasurably preferred to achievement of objectives at a lower priority level. It is often easier for a decision maker to specify which objective is more important than it is to specify how much more important that objective is. As noted by Ignizio (1978, 1983a), this preemptive priority concept was formalized by Ijira in the mid-1960's.

A characteristic common to most methods of solving multiple objective problems is that the objectives are converted into goals, thus yielding the name "Goal Programming." Goal programming as used in this dissertation will employ a combination of the preemptive priority and weighting methods of solving multiple objective systems of equations, all will be done in a linear framework. The preemptive priority structure has been chosen because it seems the most natural for decision makers. Weighting of objectives within priority levels will be allowed because the user may feel two objectives are at the same level of importance, yet feel he should weight them to express them more appropriately in terms of a common measure. Now, how to obtain a goal programming problem from a multiple objective problem will be discussed.

Generally, in a multiple objective problem one has a number of constraints, in the usual sense, along with the set of objectives. In the goal programming problem these constraints as well as the objectives
will be called goals. The constraints will usually already have aspiration levels associated with them, where an aspiration level is a desired or acceptable level of achievement. Thus, one of the first steps in developing the goal programming model is to assign aspiration levels to the objectives. The aspiration level will be used to help measure the achievement of the objective. This value should be realistic, chosen by the decision maker because of his understanding of the system under study.

Let
\[
\sum_{j=1}^{n} c_{ij} x_j
\]
represent goal i as a function of the decision variables \( x = (x_1, x_2, \ldots, x_n) \).

Let \( b_i \) represent the aspiration level associated with goal i.

As one would expect, three types of goals are possible: The user could desire \( \sum c_{ij} x_j \) to be less than, greater than, or exactly equal to what he aspires, \( b_i \). In order to achieve what has been aspired, deviation variables are assigned to each goal and appropriate combinations of these deviation variables are minimized. A negative deviation variable \( (\eta_i \geq 0) \) is added and a positive deviation variable \( (\rho_i \geq 0) \) is subtracted from each function of the decision variables in each goal. The negative
The deviation variable represents the underachievement of the aspired goal and the positive deviation variable represents the overachievement of the aspired goal. Table 1.1 summarizes what has been described.

Table 1.1. Summary

<table>
<thead>
<tr>
<th>Goal type</th>
<th>Goal programming form</th>
<th>Deviation variables to be minimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{j=1}^{n} c_{ij} x_j \leq b_i$</td>
<td>$\sum_{j=1}^{n} c_{ij} x_j + \eta_i - \rho_i = b_i$</td>
<td>$\rho_i$</td>
</tr>
<tr>
<td>$\sum_{j=1}^{n} c_{ij} x_j &gt; b_i$</td>
<td>$\sum_{j=1}^{n} c_{ij} x_j + \eta_i - \rho_i = b_i$</td>
<td>$\eta_i$</td>
</tr>
<tr>
<td>$\sum_{j=1}^{n} c_{ij} x_j = b_i$</td>
<td>$\sum_{j=1}^{n} c_{ij} x_j + \eta_i - \rho_i = b_i$</td>
<td>$\eta_i + \rho_i$</td>
</tr>
</tbody>
</table>

Thus, if the decision maker has a function of the decision variables that he desires to minimize, he should set an aspiration level, $b_i$, and then choose to minimize the positive deviation variable, $\rho_i$. This forces the optimizing routine to try for a solution at or below $b_i$. Likewise, if a function should be maximized then the negative deviation variable, $\eta_i$, is to be minimized to try to force the solution at or above the chosen aspiration level, $b_i$. Obviously, if equality between a function and the aspiration level is desired, one would minimize the sum of both deviation variables.
So far no mention has been made of how one is to minimize these deviation variables. Recall that goal programming will make use of a preemptive priority structure, therefore all goals should be assigned a priority, though each goal need not be assigned a unique priority level. Within each priority level, the user should create a function to be minimized which consists of a linear combination of the deviation variables to be minimized associated with the goals assigned to that priority level. While these functions should be linear, the deviation variables within these priority levels can be assigned weights to ensure a consistent scale of measurement. This vector of functions to be minimized, \( \mathbf{a}' = (a_1, a_2, \ldots, a_m) \), is called the achievement function where \( a_i = g_i(\eta, \rho) \) is some linear combination of the deviation variables at priority level \( i \) and \( m \) is the number of priority levels. In goal programming the achievement function is lexicographically minimized. A lexicographic minimum is one in which for an ordered vector \( \mathbf{a} \), solution \( \mathbf{a}^{(1)} \) is preferred to solution \( \mathbf{a}^{(2)} \) if \( a_i^{(1)} < a_i^{(2)} \) and \( a_i^{(1)} = a_i^{(2)} \) for \( i = 1, \ldots, k-1 \). \( \mathbf{a}^{(1)} \) is then the lexicographic minimum if it is preferred over all other possible solutions.

To achieve an \( \mathbf{a}' \) which is a lexicographic minimum of the goal programming model one must minimize \( a_1 \), then minimize \( a_2 \) subject to keeping the optimum \( a_1^* \) intact, then minimize \( a_3 \) subject to keeping the optima \( a_1^* \) and \( a_2^* \) intact, etc. Of course, at each priority level the goals act as conditions to the minimization. Thus, the function, \( g_i(\eta, \rho) \), of the deviation variables at priority level \( i \) is minimized
subject to restrictions ensuring the optima at all higher priority levels are not violated. This method of solving a multiple objective system implies that all rigid constraints need to be placed in priority level one to guarantee that all subsequent solutions satisfy as close as possible, if not absolutely, the set of rigid constraints.

If the rigid constraints placed at priority level one cannot be satisfied absolutely the solution is not infeasible because the form of goal programming does not allow for infeasible solutions. Rather, \( a^*_1 > 0 \), and what would have been an infeasibility in linear programming is simply absorbed by the deviation variables being minimized in

\[ a_1 = g_1(\eta, \rho). \]

Goal programming will still give the solution which comes closest to one satisfying the rigid constraints. It is then the user's responsibility to evaluate these absolute constraints to see why they are too rigid. In general, when the value of the function of deviation variables at priority level 1 is greater than zero in the final solution the goals associated with priority level 1 were not achieved. This indicates that successive goals are in conflict. It is the decision maker's responsibility to evaluate the solution in terms of his aspiration levels to decide if it is a viable solution.

Thus, the goal programming model is

\[
\min \ a' = \{ g_1(\eta, \rho), \ldots, g_m(\eta, \rho) \}
\]

such that \( Cx + \eta - \rho = b \)

\[ x, \eta, \rho > 0 \]
where

\( m \) is the number of priority levels
\( \mathbf{x} \) is the nx1 vector of decision variables
\( \mathbf{\eta} \) is the tx1 vector of negative deviation variables
\( \mathbf{\rho} \) is the tx1 vector of positive deviation variables
\( \mathbf{b} \) is the tx1 vector of aspiration levels
\( \mathbf{C} \) is the txn matrix of goal coefficient values
\( g_i(\mathbf{\eta}, \mathbf{\rho}) \) is the linear function of deviation variables at priority level \( i \)

An example of a goal programming problem which will be used for illustration purposes in this dissertation is one used by Ignizio (1982):

\[
\begin{align*}
\min \quad & \mathbf{z}' = \{ (\rho_1 + \rho_2), \eta_3, \rho_4 \} \\
\text{such that} \quad & 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
\quad & x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
\quad & x_1 + \eta_3 - \rho_3 = 7 \\
\quad & x_1 + 4x_2 + \eta_4 - \rho_4 = 4 \\
\quad & x, \eta, \rho \geq 0
\end{align*}
\]

The first two goals comprise those that the user considers to be rigid constraints. They were originally of the form:
Notice that $a_1$ consists of the sum of the positive deviation variables for these two goals. The third goal is from an objective the user desired to maximize. An aspiration level of 7 units was set and the negative deviation variable is minimized in $a_2$. The fourth goal is from an objective the user desired to minimize. An aspiration level of 4 units was set and the positive deviation variable is minimized in $a_3$. If the user had felt that the objectives associated with goals three and four could be at the same priority level, if weighted appropriately, then, one could have had an $a$ of the form

$$\min a' = \{(p_1 + p_2), (u_3 + w_4)\}$$

where $u$ and $w$ are weights chosen by the user. Normally, goals are weighted and assigned to the same priority level when there are a large number of goals as compared to the number of priority levels.

Chapter II will discuss the sequential method for solving linear goal programming problems. The sequential method implemented in this dissertation is a modified and improved version of that found in the literature. Chapter III presents the multidimensional dual linear goal programming problem. The algorithm used to solve the multidimensional dual in this dissertation corrects and improves upon algorithms found
in the literature. Chapter IV presents a transportation problem as a base-line problem to compare the algorithms of this dissertation and also presents an application in statistics. Chapter V summarizes results and presents some extensions in the analysis of goal programming problems. Appendices A, B, and C contain computational solutions to solve the sequential linear goal programming problem. Appendix D contains a computational solution to the multidimensional dual linear goal programming problem.
II. LINEAR GOAL PROGRAMMING

Chapter II will deal with ways to solve the linear goal programming problem,

\[ \min a' = \{g_1(n, \rho), \ldots, g_m(n, \rho) \} \]

such that \( Cx + \eta - \rho = b \)

\[ x, \eta, \rho \geq 0 \]

as initially presented in Chapter I. In general, there are two methods of solving this problem; the multiphase method which looks at the problem as an enlarged tableau and the sequential method which partitions the problem into linear programming segments corresponding to priority levels. This dissertation does not explore the multiphase method for solving linear goal programming problems, rather a discussion of this method is adequately covered by Ignizio (1982) and Leon (1985). The sequential method for solving linear goal programming problems will be extensively discussed and computational solutions will be presented.

The success of the sequential linear goal programming (SLGP) method hinges on maintaining the preemptive priority structure of linear goal programming. Recall that this preemptive priority structure specifies that the optimum to a higher priority level is infinitely more desirable than the optimum to a lower priority level. Thus, the SLGP method requires that the optima of previous solutions are maintained as one moves sequentially through priority levels. First, consider the objective
function and goals associated with priority level one:

\[
\min a_1 = g_1(\eta, \rho) \\
\text{such that } \sum_{j=1}^{n} c_{i,j} x_j + \eta_i - \rho_i = b_i \quad i \in P_1 \\
\]

\[x_i, \eta_i, \rho_i \geq 0\]

where constraints \(i\) are those associated with priority level one (\(i \in P_1\)). This is a conventional linear programming problem and any conventional linear programming software package can be used to computationally solve this problem. Denote the optimum to this problem as \(a_1^*\). Next, consider the objective function and goals associated with priority level two:

\[
\min a_2 = g_2(\eta, \rho) \\
\text{such that } \sum_{j=1}^{n} c_{k,j} x_j + \eta_k - \rho_k = b_k \quad k \in P_2 \\
\]

\[x_k, \eta_k, \rho_k \geq 0\]

This, too, is a conventional linear programming problem but further conditions must be imposed on the problem to ensure that the optimum from priority level one is not violated. The goals from priority level one,

\[
\sum_{j=1}^{n} c_{i,j} x_j + \eta_i - \rho_i = b_i \quad i \in P_1 ,
\]
as well as a new constraint using priority level one's objective function equated to its corresponding optimum,

\[ g_1(\eta, \rho) = a_1^* , \]

must be included in the linear programming problem to maintain the preemptive priority structure at priority level two. In general, then, each current priority level's linear programming problem will be of the form:

\[
\begin{align*}
\min & \quad g_h(\eta, \rho) \\
\text{such that} & \quad \sum_{j=1}^{n} c_{\ell,j}x_j + \eta_\ell - \rho_\ell = b_\ell \quad \ell \in p_1, p_2, \ldots, p_h \\
& \quad g_1(\eta, \rho) = a_1^* \\
& \quad \vdots \\
& \quad g_{h-1}(\eta, \rho) = a_{h-1}^* \\
& \quad x, \eta, \rho \geq 0
\end{align*}
\]

In other words, at any given priority level one minimizes the objective function associated with the current priority level subject to the goals associated with all priority levels through the current level plus constraints created from previous priority level objective functions equated to their optima.
For example, consider Example A, the goal programming problem presented in Chapter I.

\[
\min \ a^* = \{(\rho_1 + \rho_2), \eta_3, \rho_4\}
\]

such that
\[
\begin{align*}
2x_1 + x_2 + \eta_1 - \rho_1 &= 12 \\
x_1 + x_2 + \eta_2 - \rho_2 &= 10 \\
x_1 + \eta_3 - \rho_3 &= 7 \\
x_1 + 4x_2 + \eta_4 - \rho_4 &= 4
\end{align*}
\]

\[x, \eta, \rho \geq 0\]

The first two goals are associated with priority level one, the third goal with priority level two and the last goal with priority level three. The SLGP approach specifies that one would first solve the linear programming problem:

\[
\min \ \rho_1 + \rho_2
\]

such that
\[
\begin{align*}
2x_1 + x_2 + \eta_1 - \rho_1 &= 12 \\
x_1 + x_2 + \eta_2 - \rho_2 &= 10
\end{align*}
\]

\[x, \eta, \rho \geq 0\]

to obtain an optimum of \( a^*_1 = 0 \). Next, one adds goal three and a constraint to ensure the present optimum is not violated before solving:
\[
\min \eta_3
\]
such that \[2x_1 + x_2 + \eta_1 - \rho_1 = 12\]
\[x_1 + x_2 + \eta_2 - \rho_2 = 10\]
\[x_1 + \eta_3 - \rho_3 = 7\]
\[\rho_1 + \rho_2 = 0\]
\[x, \eta, \rho > 0\]
to obtain an optimum of \(a_2^* = 1\). Lastly, one adds goal four and a constraint to ensure that this new optimum is not violated before solving:

\[
\min \rho_4
\]
such that \[2x_1 + x_2 + \eta_1 - \rho_1 = 12\]
\[x_1 + x_2 + \eta_2 - \rho_2 = 10\]
\[x_1 + \eta_3 - \rho_3 = 7\]
\[x_1 + 4x_2 + \eta_4 - \rho_4 = 4\]
\[\rho_1 + \rho_2 = 0\]
\[\eta_3 = 1\]
\[x, \eta, \rho > 0\]
to obtain a solution with an optimum of \(a_3^* = 2\). The final goal programming solution to this problem is \(a^* = (0,1,2)'\), \(x_1^* = 6\), \(x_2^* = 0\).

The sequential method for solving linear goal programming problems was chosen over the multiphase method because the bulk of the work in the SLGP method involves solving a traditional linear programming problem,
which allows one to make use of any available linear programming software package. Evans (1984) points out the need for research to address the problem of solving large-scale multi-objective problems. Thus, there appeared to be an ideal situation at Iowa State University where IBM Mathematical Programming System Extended packages (MPSX and MPSX/370) were available for use in developing the SLGP programs.

MPSX and its updated version MPSX/370 are well-established in their ability to deliver precise solutions for relatively large linear programming problems. Procedures in MPSX (or MPSX/370) are carried out in long precision, though solution accuracy is dependent upon the problem size, complexity and density (average number of nonzero elements per column). Problem size and corresponding data storage requirements are documented in the user's manual (1979b). Naturally, the goal programming problem's size must fall within the limits set by MPSX (or MPSX/370). MPSX (or MPSX/370) can handle up to 16,383 rows in its linear programming problems, with the number of variables allowed dependent upon available storage. Thus, the largest goal programming problem that can be solved is one where the sum of the number of priority levels plus the number of goals is less than 16,383. Ignizio and Perlis (1980) present an algorithm (without computer code) for implementing SLGP via MPSX. The algorithm they present differs from the one used in this dissertation, nevertheless they present some performance results for moderate sized problems when they implement their algorithm. Most algorithms presented in the literature to solve linear goal programming problems, accompanied by some form of computer code, are efficient only with small problems. A
recent study by Olson (1984) compares four such algorithms, yet the study is limited to problems with 120 constraints and 100 decision variables. This dissertation presents a means for solving large linear goal programming problems accurately and with the ease of using the highly efficient MPSX (or MPSX/370) systems.

The method of implementing SLGP via MPSX (or MPSX/370) is slightly modified from the SLGP method described earlier. At the beginning of this chapter, any particular iteration in the SLGP problem was said to consist of the portion of the achievement function to be optimized subject to the goals associated with the current priority level and all higher priorities as well as constraints ensuring previous optima are not violated. Recall from Chapter I, that all goals are constructed with positive and negative deviation variables which eliminates the concept of an infeasible solution, rather the overachievement or underachievement of a goal is absorbed by the deviation variables. Because the function of the deviation variables prevents infeasibility, the inclusion of all goals at any particular iteration (including goals associated with a lower priority level) will not adversely affect the optimum at the given priority level. This allows the information on all goals and all levels of the achievement function to be entered into the MPSX (or MPSX/370) system initially, keeping the changes after each iteration to a minimum. For example, recall when using SLGP to solve Example A, the first two linear programming problems were:
\[
\begin{align*}
\text{min } & \quad \rho_1 + \rho_2 \\
\text{such that } & \quad 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
& \quad x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
& \quad x, \eta, \rho \geq 0
\end{align*}
\]

and

\[
\begin{align*}
\text{min } & \quad \eta_3 \\
\text{such that } & \quad 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
& \quad x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
& \quad x_1 + \eta_3 - \rho_3 = 7 \\
& \quad \rho_1 + \rho_2 = a_1^* \\
& \quad x, \eta, \rho \geq 0
\end{align*}
\]

One can see that including the goal associated with priority level two,

\[
x_1 + \eta_3 - \rho_3 = 7
\]

in step one cannot affect the optimum for priority level one since \(\eta_3\) and \(\rho_3\) are not in objective function one and any resulting difference between 7 and \(x_1\) is absorbed by \(\eta_3 - \rho_3\). Thus, the linear programming problem at priority level one can be written:
\[
\begin{align*}
\text{Priority One} \\
\text{min } & \rho_1 + \rho_2 \\
\text{such that } & 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
& x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
& x_1 + \eta_3 - \rho_3 = 7 \\
& x_1 + 4x_2 + \eta_4 - \rho_4 = 4 \\
& x, \eta, \rho > 0
\end{align*}
\]

This idea can be carried through to all priority levels until one would obtain the following series when sequentially solving Example A:

\[
\begin{align*}
\text{Priority Two} \\
\text{min } & \eta_3 \\
\text{such that } & 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
& x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
& x_1 + \eta_3 - \rho_3 = 7 \\
& x_1 + 4x_2 + \eta_4 - \rho_4 = 4 \\
& \rho_1 + \rho_2 = 0 \\
& x, \eta, \rho > 0
\end{align*}
\]
Priority Three

\[
\begin{align*}
\min & \quad \rho_4 \\
\text{such that} & \quad 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
& \quad x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
& \quad x_1 + \eta_3 - \rho_3 = 7 \\
& \quad x_1 + 4x_2 + \eta_4 - \rho_4 = 4 \\
& \quad \rho_1 + \rho_2 = 0 \\
& \quad \eta_3 = 1 \\
& \quad x, \eta, \rho \geq 0
\end{align*}
\]

This idea is appealing from a programmer's viewpoint because one need only keep track of objective functions and their optima as one iterates through the priority levels of the goal programming problem. All of the information for the linear goal programming problem, information on its goals and on its achievement function, can be entered into the program before the first priority level's problem is solved. At each iteration the latest objective function is added by the program as a constraint in the next priority level's linear programming problem. Then, minimizing the next objective function causes the set of goals which have deviation variables in this new objective function to have an impact on the solution at this next priority level.

The MPSX (or MPSX/370) REVISE procedure is used to modify the goal programming problem after solving the linear programming problem for each priority level. Depending on which MPSX or MPSX/370 SLGP program is
used (there are three in all which will be discussed later), at each iteration a revise file is created on disk or internally to add the last objective function as a row constraint to the problem and assigns its optimum as the new row's right hand side value. This new constraint adds only one row dimension to the basis whereas in the traditional SLGP approach the dimension change is variable.

The method presented here to implement SLGP via MPSX differs from that presented by Ignizio and Perlis (1980) both in the problem input and in the sequential problem changes performed before solving a given priority level's linear programming problem. Detailed descriptions of the three programs as well as how they solve the SLGP problem are presented in Appendices A, B, and C.

Three programs were created because of various differences in MPSX and MPSX/370, though in each case the general form for solving the SLGP problem is that which was just presented. The first program, detailed in Appendix A, makes use of MPSX and the READCOMM procedure. This program was created because MPSX is always available on ISU's campus. The second program, detailed in Appendix B, uses MPSX/370 and its READCOMF procedure. MPSX/370 is a later version of MPSX with a number of changes, especially in the use of its READCOMF procedure, as well as numerous enhancements. Some of these enhancements allow for performance improvement in MPSX/370 over that of MPSX. The user's manual (1979b) notes that improvement is most noticeable for problems with over 800 rows or problems having a dense nucleus during inversion.
The PRIMAL optimizing procedure was redesigned to be more efficient. For this reason a program using MPSX/370 was created, though MPSX/370 may not always be available on ISU's computer system. MPSX/370 has two control languages for use in programming. The control language used in the program in Appendix B is one that is specific to MPSX/370, the MPS Control Language, and is the one compatible or comparable to MPSX. The second control language, Extended Control Language (hereafter referred to as MPSX/370 ECL), is one based on the high-level programming language PL/I. MPSX/370 ECL has greater capabilities than MPSX and MPSX/370, thus while it is as efficient as MPSX/370 it allows for greater precision in linear goal programming than either MPSX or MPSX/370. The one flaw in the precision of SLGP via MPSX (or MPSX/370) is that the revise files are written and then read from disk, causing some loss in precision in the optimum value when it is subsequently used as a right hand side. MPSX/370 ECL, on the other hand, allows one to make use of structures in the REVISE procedure, which allows all results to remain internal and the optimum value's precision is preserved. The third program to solve the SLGP using MPSX/370 ECL is detailed in Appendix C.

As has been mentioned, the goal programming problem has traditionally been described as one which cannot have an infeasible solution. At each priority level one attempts to minimize the difference between a set of goals and their desired aspiration levels, given that higher priority levels are optimized. If a set of goals is considered by the user to be absolute in the sense that constraints are absolute in linear programming, then it is recommended that the deviation variables
associated with these goals be minimized in priority level one. An optimum of zero for priority level one would then yield a goal programming solution in what could be called the feasible region in the sense that all absolute goals were met. An optimum different from zero for priority level one would indicate that these absolute goals were unrealistic and the solution obtained would be the one as close to achieving these goals as allowed by their aspiration levels. It would then be up to the user to decide whether this solution was one that could be used or whether the problem would need to be solved with new aspiration levels. A benefit of this method is that one always obtains a solution. Goals that the user considered to be absolute are shown to have been too stringent, or the aspiration levels for these goals were chosen inaccurately, yet the best possible solution given these goals is provided without the SLGP algorithm (and hence computer program) terminating prematurely. The disadvantage to this method is apparent when the user has a problem whose solution would be useless unless the absolute goals were satisfied or where these goals had aspiration levels whose values were not questionable. It would be a waste of time and money, in this case, to solve beyond the first priority level if the optimum at priority level one were not zero. Rather than have the SLGP program check the optimum at priority one before deciding to continue or stop, a user in the situation where his absolute goals must be satisfied would be better off using a slightly different goal programming problem formulation. In this formulation, the user would
include the absolute goals in the problem without their deviation variables. This is their original form (greater than, equal to or less than the right hand side) and there is no longer a priority level associated with these goals. Now, a feasible solution need not always exist and the SLGP program will terminate if one cannot be found. For example, consider goal programming problem Example A presented in Chapter I. Recall that the goals associated with priority level one were originally of the form:

\begin{align*}
2x_1 + x_2 & \leq 12 \\
-x_1 + x_2 & \leq 10
\end{align*}

Suppose these two goals were absolute and that a solution that did not satisfy these two goals was unacceptable. Also, assume that the right hand side values 12 and 10 are not questionable. Example A could be reformulated in the following way:

\begin{align*}
\min \{ \eta_3, \rho_4 \} \\
\text{such that} \quad 2x_1 + x_2 & \leq 12 \\
x_1 + x_2 & \leq 10 \\
x_1 + \eta_3 - \rho_3 & = 7 \\
x_1 + 4x_2 + \eta_4 - \rho_4 & = 4 \\
x, \eta, \rho & \geq 0
\end{align*}
The original priority level one has been dropped and the goals are entered in MPSX (or MPSX/370) as presented in (2.1). Notice that a benefit to this method is that there is one less priority level to solve and two fewer variables per absolute goal in the problem.

There is another way to reformulate the goal programming problem which maintains the flexibility one has in using the deviation variables in the achievement function, yet decreases the number of variables in the problem. The disadvantage to this formulation, is that more work is required of the user before the problem is solved. Recall the general form of the linear goal programming problem

\[ \min \ a' = \{g_1(\eta, \rho), \ldots, g_m(\eta, \rho)\} \]

such that \( Cx + In - Ip = b \)

\[ \begin{align*} x, \eta, \rho &> 0 \end{align*} \]

One could solve for \( \eta_i, i = 1, \ldots, n \) in the goal section:

\[ \eta = b + Ip - Cx \]

and then substitute for \( \eta \) in the achievement function to obtain the new problem formulation:

\[ \min \ a' = \{g_1(b + Ip - Cx, \rho), \ldots, g_m(b + Ip - Cx, \rho)\} \]

\[ Cx + In - Ip = b \]

\[ x, \eta, \rho > 0 . \]
Now, since the $\eta_i$, $i = 1, ..., n$ are no longer in the achievement function and since they form the initial problem basis one could say that they are posing as slack variables in the sense of slack variables in a traditional linear programming problem. MPSX (and likewise MPSX/370) does not require the user to include slack variables when entering the problem, therefore the linear goal programming problem could be formulated and entered in the following form:

$$\min \ a = \{g_1(b + Ip - Cx, \rho), ..., g_m(b + Ip - Cx, \rho)\}$$

such that $Cx - Ip \leq b$

$$x, \rho \geq 0$$

To illustrate, reconsider Example A:

$$\min \ a' = \{(\rho_1 + \rho_2), \eta_3, \rho_4\}$$

such that $2x_1 + x_2 + \eta_1 - \rho_1 = 12$

$x_1 + x_2 + \eta_2 - \rho_2 = 10$

$x_1 + \eta_3 - \rho_3 = 7$

$x_1 + 4x_2 + \eta_4 - \rho_4 = 4$

$$x, \eta, \rho \geq 0$$

We need only substitute for $\eta_3$ in $a$, $\eta_3 = 7 + \rho_3 - x_1$. The new problem is then
\[
\min a' = \{(\rho_1 + \rho_2), (7 + \rho_3 - x_1), \rho_4\}
\]
such that
\[
\begin{align*}
2x_1 + x_2 - \rho_1 &\leq 12 \\
-x_1 + x_2 - \rho_2 &\leq 10 \\
x_1 - \rho_3 &\geq 7 \\
x_1 + 4x_2 - \rho_4 &\leq 4
\end{align*}
\]
\[x, \rho \geq 0.\]

The problem size has been reduced by one variable per goal.

If the user felt that priority level one goals were absolute and a solution not satisfying the associated two goals was unacceptable, then the last two formulations could be combined to yield the following linear goal programming problem:

\[
\min a' = \{(7 + \rho_3 - x_1), \rho_4\}
\]
such that
\[
\begin{align*}
2x_1 + x_2 &\leq 12 \\
-x_1 + x_2 &\leq 10 \\
x_1 - \rho_3 &\leq 7 \\
x_1 + 4x_2 - \rho_4 &\leq 4
\end{align*}
\]
\[x, \rho \geq 0.\]

This problem has been reduced in size from the original traditional linear goal programming problem by one priority level, two deviation variables per absolute goal and one deviation variable per nonabsolute
goal. The usefulness and ease of implementation of these two problem reformulations depends on the user's definition of the problem and the form of the achievement function.
III. DUAL LINEAR GOAL PROGRAMMING

This chapter presents a computational procedure to solve the dual of the primal linear goal programming problem. Just as a conventional linear programming problem has a dual problem associated with it, the linear goal programming problem has an associated dual problem, called the multidimensional dual problem, Ignizio (1983b). The theory and properties of the dual problem of a linear goal programming problem are explored by Markowski and Ignizio (1983b). The theory is similar to that for a conventional linear programming problem and its dual, translated into a preemptive priority framework. This chapter does not explore the theory associated with the dual, rather methods of obtaining the multidimensional dual are presented followed by an explanation of the algorithm used to solve the linear goal programming dual problem.

Recall the form of the linear goal programming (LGP) problem:

\[
\min a' = \{g_1(\eta, \rho), \ldots, g_m(\eta, \rho)\}
\]

such that \( Cx + I\eta - I\rho = b \)

\[
x, \eta, \rho \geq 0
\]

For the above linear goal problem, each member of the achievement function, \( g_i(\eta, \rho) \), can be written as \( u^i\eta + v^i\rho \) where \( u^i \) and \( v^i \) are the vector of weights at priority level \( i \) associated, respectively, with the negative and positive deviation variables in the achievement function. Our achievement function can therefore be written as:
The dual problem of the LGP problem can be obtained from this form of the primal, but for reasons to be explained later other modifications will be made before presenting the dual problem. As mentioned in Chapter II, one could solve for \( \eta \) and use this substitution in the achievement function. Thus, the general form of the LGP problem used in this chapter will be:

\[
\min a' = \{ u' \eta + w' \rho, \ u' \eta + w' \rho, \ldots, \ u' \eta + w' \rho \}
\]

such that \( Cx + \eta - \rho = b \) \hspace{1cm} \text{(3.1)}

\( x, \eta, \rho > 0 \)

Before finding the dual problem to the LGP problem in 3.1, Ignizio (1983b), multiplies the goal section through by a minus sign; i.e.,

\(-Cx - \eta + \rho = -b.\) The reason for this will be explained later. The multidimensional dual problem, as presented by Ignizio (1983b), is then:

\[
\max \{-b' y^1 + u' b, \ldots, -b' y^m + u' b\}
\]

such that

\[
\begin{bmatrix}
-C' \\
-I \\
-I
\end{bmatrix}
\begin{bmatrix}
y^1 \\
y^2 \\
\vdots \\
y^m
\end{bmatrix}
\begin{bmatrix}
u' C' \\
0 \\
1' + \omega'
\end{bmatrix}
\begin{bmatrix}
u C' \\
0 \\
u' + \omega'
\end{bmatrix}
\]

\( y^1, y^2, \ldots, y^m \) unrestricted
with $y_{txi}^1$ for $i = 1, 2, \ldots, m$. The resulting problem is a series of conventional linear programming problems with multiple, prioritized right hand sides. The objective function is essentially the same for each right hand side if one excludes the ordered set of constants $(u_1^b, \ldots, u^m b)$.

This is the form Ignizio (1983b) uses in his algorithm to solve the multidimensional dual problem. However, if one rewrites this form slightly one obtains a problem which is more easily adapted to conventional linear programming software packages. In particular:

$$
\begin{align*}
\max \{ -b^1 y^1 + u_1^b, \ldots, -b^m y^m + u^m b \} \\
\text{such that } [-C'] [y^1, y^2, \ldots, y^m] \leq [-u_1'^C, \ldots, -u^m C] \\
[I] [y^1, y^2, \ldots, y^m] \geq [0], \ldots, [0] \\
y^1, y^2, \ldots, y^m \text{ unrestricted}
\end{align*}
$$

which becomes

$$
\begin{align*}
\max \{ -b^1 y^1 + u_1^b, \ldots, -b^m y^m + u^m b \} \\
\text{such that } [-C'] [y^1, y^2, \ldots, y^m] \leq [-u_1'^C, \ldots, -u^m C] \\
[I] [y^1, y^2, \ldots, y^m] \geq [0], \ldots, [0] \\
y^1, y^2, \ldots, y^m \geq 0
\end{align*}
$$
Thus, substituting for $\eta$ in the LGP achievement function assures a series of zero right hand side vectors in the dual problem for the rows corresponding to the $\eta$ variables. Multiplying through by the negative sign in the LGP goal section allows one to condense the dual problem and change the dual variables from being unrestricted to now having the usual nonnegativity restrictions. This form is alluded to by Markowski and Ignizio (1983a, 1983b) as they develop the multidimensional dual theory, but its usefulness is not explored. An example will clarify what has been presented. Reconsider Example A from Chapter I:

$$\min \{ (\rho_1 + \rho_2), \eta_3, \rho_4 \}$$

such that

$$2x_1 + x_2 + \eta_1 - \rho_1 = 12$$

$$x_1 + x_2 + \eta_2 - \rho_2 = 10$$

$$x_1 + \eta_3 - \rho_3 = 7$$

$$x_1 + 4x_2 + \eta_4 - \rho_4 = 4$$

$$x_1, x_2, \eta_i, \rho_i \geq 0 \quad i = 1, 2, 3, 4$$

Substituting for $\eta$ in the achievement function and multiplying the goal section through by a minus sign gives us:

$$\min \{ (\rho_1 + \rho_2), (7 - x_1 + \rho_3), \rho_4 \}$$

such that

$$-2x_1 - x_2 - \eta_1 + \rho_1 = -12$$

$$-x_1 - x_2 - \eta_2 + \rho_2 = -10$$

$$-x_1 - \eta_3 + \rho_3 = -7$$

$$-x_1 - 4x_2 - \eta_4 + \rho_4 = -4$$

$$x_1, x_2, \eta_i, \rho_i \geq 0 \quad i = 1, 2, 3, 4$$
The multidimensional dual problem in the form used by Ignizio is then:

\[
\max \{-12y_1^2 - 10y_2^1 - 7y_3^1 - 4y_4^1\}, \quad \{-12y_2^2 - 10y_2^2 - 7y_3^2 - 4y_4^2 + 7\},
\]

\[
\{-12y_3^3 - 10y_2^3 - 7y_3^3 - 4y_4^3\}\]

such that

\[
\begin{bmatrix}
-2 & -1 & -1 & -1 \\
-1 & -1 & 0 & -4 \\
-1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 \\
0 & 0 & -1 & 0 \\
0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1^1 \\
y_1^2 \\
y_1^3 \\
y_2^1 \\
y_2^2 \\
y_2^3 \\
y_3^1 \\
y_3^2 \\
y_3^3 \\
y_4^1 \\
y_4^2 \\
y_4^3
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
-1 \\
0 \\
0 \\
-1 \\
0 \\
0 \\
1 \\
1 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]

\[y_i^1, y_i^2, y_i^3 \text{ unrestricted for } i = 1, 2, 3, 4.\]

This multidimensional dual can be equivalently written as:

\[
\max \{-12y_1^2 - 10y_2^1 - 7y_3^1 - 4y_4^1\}, \quad \{-12y_2^2 - 10y_2^2 - 7y_3^2 - 4y_4^2 + 7\},
\]

\[
\{-12y_3^3 - 10y_2^3 - 7y_3^3 - 4y_4^3\}\]

such that

\[
\begin{bmatrix}
-2 & -1 & -1 & -1 \\
-1 & -1 & 0 & -4 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1^1 \\
y_1^2 \\
y_1^3 \\
y_2^1 \\
y_2^2 \\
y_2^3 \\
y_3^1 \\
y_3^2 \\
y_3^3 \\
y_4^1 \\
y_4^2 \\
y_4^3
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
1 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]
\[ y_i^1, y_i^2, y_i^3 > 0 \text{ for } i = 1, 2, 3, 4. \]

This last form yields a problem with a smaller constraint set with variables under the usual nonnegativity restrictions, thus, it is easily solved using any conventional linear programming software package.

One obtains the same result when one considers the reformulated LGP problem presented in Chapter II with the \( \eta_i \) variables viewed as slacks:

\[
\begin{align*}
\min a' &= \{ u^1(b-Cx+\rho) + \rho, u^2(b-Cx+\rho) + \rho, \ldots, u^m(b-Cx+\rho) + \rho \} \\
\text{such that } Cx - \rho &\leq b \\
x, \rho &> 0
\end{align*}
\]

The multidimensional dual of this formulation is then:

\[
\begin{align*}
\max \{-b'y^1 + u^1'b, \ldots, -b'y^m + u^m'b\} \\
\text{such that } \begin{bmatrix} -C' \\ I \end{bmatrix} \begin{bmatrix} y^1, y^2, \ldots, y^m \end{bmatrix} &\leq \begin{bmatrix} -u^1'C \\ u^1' + \rho \end{bmatrix}, \ldots, \begin{bmatrix} -u^m'C \\ u^m' + \rho \end{bmatrix} \\
y^1, y^2, \ldots, y^m &> 0
\end{align*}
\]

Example A, as a reformulated LGP problem was:
\[
\min \{ (\rho_1 + \rho_2), (7 - x_1 + \rho_3), \rho_4 \}
\]
such that \[2x_1 + x_2 - \rho_1 \leq 12 \]
\[x_1 + x_2 - \rho_2 \leq 10 \]
\[x_1 - \rho_3 \leq 7 \]
\[x_1 + 4x_2 - \rho_4 \leq 4 \]
\[x_1, x_2, \rho_i \geq 0 \quad i = 1, 2, 3, 4 \]

and its resulting multidimensional dual problem is:

\[
\max \{ (-12y_1^1 - 10y_2^1 - 7y_3^1 - 4y_4^1), (-12y_1^2 - 10y_2^2 - 7y_3^2 - 4y_4^2 + 7), (-12y_1^3 - 10y_2^3 - 7y_3^3 - 4y_4^3) \}
\]
such that

\[
\begin{bmatrix}
-2 & -1 & -1 & -1 \\
-1 & -1 & 0 & -4 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
y_1^1 \\
y_2^1 \\
y_3^1 \\
y_4^1 \\
y_1^2 \\
y_2^2 \\
y_3^2 \\
y_4^2 \\
y_1^3 \\
y_2^3 \\
y_3^3 \\
y_4^3
\end{bmatrix}
\leq
\begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
0 \\
0 \\
1
\end{bmatrix}
\]

\[y_1^i, y_2^i, y_3^i \geq 0 \quad \text{for } i = 1, 2, 3, 4 \]

This dissertation uses this form of the multidimensional dual problem, where the dual variables have the usual nonnegativity restrictions, when a solution is sought. Another example of how the multidimensional dual problem is obtained from a LGP problem is presented in Chapter IV.
Just as the LGP problem was solved sequentially, the multi-dimensional dual problem is solved sequentially. Recall from Chapter II, that the SLGP problem consists of the following series of problems:

\[
\min a_1 = \{u^1(b - Cx + \eta) + w^1\rho\} \quad (3.2)
\]

such that \( Cx + \eta - \rho = b \)
\[ x, \eta, \rho \geq 0 \]

\[
\min a_2 = \{u^2(b - Cx + \eta) + w^2\rho\} \quad (3.3)
\]

such that \( Cx + \eta - \rho = b \)
\[-u^1Cx + (u^1 + w^1)\rho = a_1^* - u^1b \]
\[ x, \eta, \rho \geq 0 \]

where \( a_1^* \) is the optimum of (3.2).

\[
\min a_3 = \{u^3(b - Cx + \eta) + w^3\rho\} \quad (3.4)
\]

such that \( Cx + \eta - \rho = b \)
\[-u^1Cx + (u^1 + w^1)\rho = a_1^* - u^1b \]
\[-u^2Cx + (u^2 + w^2)\rho = a_2^* - u^2b \]
\[ x, \eta, \rho \geq 0 \]
where

\[ a_1^* \] is the optimum of (3.2) and

\[ a_2^* \] is the optimum of (3.3)

\[ \vdots \]

\[ \min a_m = \{u^m (b - Cx + Ip) + w^m \rho \} \]

such that

\[ Cx + \eta - \rho = b \]

\[ -u^1 Cx + (u^1 + w^1) \rho = a_1^* - u^1 b \] \hspace{1cm} (3.5)

\[ -u^2 Cx + (u^2 + w^2) \rho = a_2^* - u^2 b \]

\[ \vdots \]

\[ -u^{m-1} Cx + (u^{m-1} + w^{m-1}) \rho = a_{m-1}^* - u^{m-1} b \]

\[ x, \eta, \rho > 0 \]

where \( a_i^* \) is the optimum of the linear programming problem associated with priority level \( i \).

These are all traditional linear programming problems, thus, we can consider a series of dual problems related to each of these primal problems. Also, as before, we can multiply the original goal set through by a minus one without losing any meaning. The dual problems of (3.2), (3.3), (3.4) ... (3.5) are the following:
\[
\begin{align*}
\text{max} & \quad b'y + u'1'b \\
\text{such that} & \quadegin{bmatrix} -C' \\ -I \\ I \end{bmatrix} y \leqegin{bmatrix} -u'1'C \\ 0 \\ u'1' + w'1' \end{bmatrix} \\
& \quad y \text{ unrestricted} \\
\text{max} & \quad b'y + (a_1 - u'1'b)\lambda_1 + u'2'b \\
\text{such that} & \quadegin{bmatrix} -C' \\ -I \\ I \end{bmatrix} y \leqegin{bmatrix} -u'2'C \\ 0 \\ u'2' + w'2' \end{bmatrix} \\
& \quad y, \lambda_1 \text{ unrestricted} \\
\text{max} & \quad b'y + (a_1 - u'1'b)\lambda_1 + (a_2 - u'2'b)\lambda_2 + u'3'b \\
\text{such that} & \quadegin{bmatrix} -C' \\ -I \\ I \end{bmatrix} y \leqegin{bmatrix} -u'3'C \\ 0 \\ u'3' + w'3' \end{bmatrix} \\
& \quad y, \lambda_1, \lambda_2 \text{ unrestricted} \\
\vdots \\
\text{max} & \quad b'y + (a_1 - u'1'b)\lambda_1 + (a_2 - u'2'b)\lambda_2 + \ldots + (a_{m-1} - u'm-1'b)\lambda_{m-1} + u'm'b \\
\text{such that} & \quadegin{bmatrix} -C' \\ -I \\ I \end{bmatrix} y \leqegin{bmatrix} -u'm'C \\ 0 \\ u'm' + w'm' \end{bmatrix} \\
& \quad y, \lambda_1, \lambda_2 \ldots \lambda_{m-1} \text{ unrestricted.}
\end{align*}
\]
Now, recall how in the SLGP problem, each step contained extra constraints of the form,

\[-u^i'Cx + (u^i' + w^i')\rho = a^*_i - u^i'b,\]

one for each priority level lower than the one currently being solved. These ensured that the previous optima were not violated. Since each optimum was a minimum, one could write these constraints as

\[-u^i'Cx + (u^i' + w^i')\rho \leq a^*_i - u^i'b\]

without fear of violating the optimum of that priority level. Using this form for the extra constraints, as well as reducing the dual because of the zero right hand sides, one could rewrite the series of sequential dual problems, (3.6), (3.7), ... (3.9), to obtain the following series of problems with variables under the usual nonnegativity restrictions.

\[
\begin{align*}
\text{max} & - b'y + u^1'b \\
\text{such that} & \begin{bmatrix} -C' \\ I \end{bmatrix} \begin{bmatrix} y \\ \lambda^1 \end{bmatrix} \leq \begin{bmatrix} -u^1'C \\ u^1' + w^1' \end{bmatrix} \\
\lambda^1 & \geq 0
\end{align*}
\]

(3.10)

\[
\begin{align*}
\text{max} & - b'y + (a^*_1 - u^1'b)\lambda^1 + u^2'b \\
\text{such that} & \begin{bmatrix} -C' & -u^1'C \\ I & u^1' + w^1' \end{bmatrix} \begin{bmatrix} y \\ \lambda^1 \end{bmatrix} \leq \begin{bmatrix} -u^2'C \\ u^2' + w^2' \end{bmatrix} \\
y \geq 0 & \quad \lambda^1 \geq 0
\end{align*}
\]

(3.11)
\[
\text{max } - b'y + (a_1 - u'v' b)\lambda_1 + (a_2 - u'v' b)\lambda_2 + u'v' b \\
\text{such that } \begin{bmatrix} -C' & -u'v' C & -u'v' C \\ I & u'v' + w' + w' & u'v' + w' + w' \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} \leq \begin{bmatrix} -u'v' C \\ u'v' + w' \end{bmatrix} \tag{3.12}
\]
\[
y \geq 0 \quad \lambda_1, \lambda_2 \geq 0.
\]

\[
\text{max } - b'y + (a_1 - u'v' b)\lambda_1 + (a_2 - u'v' b)\lambda_2 + \ldots + (a_{m-1} - u'v' b)\lambda_{m-1} + u'v' b \\
\text{such that } \begin{bmatrix} -C' & -u'v' C & -u'v' C & \ldots & -u'v' C \\ I & u'v' + w' + w' + w' & u'v' + w' + w' + w' & \ldots & u'v' + w' + w' + w' \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_{m-1} \end{bmatrix} \leq \begin{bmatrix} -u'v' C \\ u'v' + w' \end{bmatrix} \tag{3.13}
\]
\[
y \geq 0 \quad \lambda_1, \lambda_2, \ldots, \lambda_{m-1} \geq 0.
\]

These dual problems are all conventional linear programming problems, therefore, they can be solved by any conventional linear programming software package.

Example A will again be used to illustrate these ideas. At the first priority level, the linear goal programming problem is:
\[
\begin{align*}
\min & \quad \rho_1 + \rho_2 \\
\text{such that} & \quad 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \\
& \quad x_1 + x_2 + \eta_2 - \rho_2 = 10 \\
& \quad x_1 + \eta_3 - \rho_3 = 7 \\
& \quad x_1 + 4x_2 + \eta_4 - \rho_4 = 4 \\
& \quad x, \eta, \rho \geq 0
\end{align*}
\]

and its corresponding dual problem is:

\[
\max - 12y_1 - 10y_2 - 7y_3 - 4y_4 \\
\text{such that} \\
\begin{bmatrix}
-2 & -1 & -1 & -1 \\
-1 & -1 & 0 & -4 \\
-1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 \\
0 & 0 & -1 & 0 \\
0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
\leq 
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0 \\
0 \\
\end{bmatrix}
\]

\(y_i\) unrestricted for \(i = 1, 2, 3, 4\).

At priority level two the linear programming problem is:
\[ \min \eta_3 = 7 - x_1 + \rho_3 \]
such that \[ 2x_1 + x_2 + \eta_1 - \rho_1 = 12 \]
\[ x_1 + x_2 + \eta_2 - \rho_2 = 10 \]
\[ x_1 + \eta_3 - \rho_3 = 7 \]
\[ x_1 + 4x_2 + \eta_4 - \rho_4 = 4 \]
\[ \rho_1 + \rho_2 = 0 \]
\[ x, \eta, \rho > 0 \]

and its corresponding dual problem is:

\[ \max -12y_1 - 10y_2 - 7y_3 - 4y_4 + 0\lambda_1 + 7 \]
such that
\[
\begin{bmatrix}
-2 & -1 & -1 & -1 & 0 \\
-1 & -1 & 0 & -4 & 0 \\
-1 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\lambda_1
\end{bmatrix}
\leq
\begin{bmatrix}
-1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\]
\[ y_1, y_2, y_3, y_4, \lambda_1 \text{ unrestricted.} \]
Finally, at priority level three the linear programming problem is:

\[
\begin{align*}
\min \ &= \rho_4 \\
\text{such that} \ &\quad \begin{align*}
2x_1 + x_2 + \eta_1 - \rho_1 &= 12 \\
3x_1 + x_2 + \eta_2 - \rho_2 &= 10 \\
x_1 + \eta_3 - \rho_3 &= 7 \\
x_1 + 4x_2 + \eta_4 - \rho_4 &= 4 \\
\rho_1 + \rho_2 &= 0 \\
-x_1 + \rho_3 &= -6 \\
x, \eta, \rho &> 0
\end{align*}
\end{align*}
\]

and the final dual problem is:

\[
\begin{align*}
\max \ &= -12y_1 - 10y_2 - 7y_3 - 4y_4 + 0\lambda_1 - 6\lambda_2 \\
\text{such that} \ &\quad \begin{pmatrix}
-2 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & 0 & -4 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\lambda_1 \\
\lambda_2
\end{pmatrix}
\leq
\begin{pmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
1
\end{pmatrix}
\]

\(y_1, y_2, y_3, y_4, \lambda_1, \lambda_2\) unrestricted.
These sequential dual problems can then be condensed to yield the following series of linear programming problems:

\[
\text{max } -12y_1 - 10y_2 - 7y_3 - 4y_4
\]

such that

\[
\begin{bmatrix}
-2 & -1 & -1 & -1 \\
-1 & -1 & 0 & -4 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
\leq
\begin{bmatrix}
0 \\
0 \\
1 \\
1 \\
0 \\
0 \\
\end{bmatrix}
\]

\[y_1, y_2, y_3, y_4 \geq 0\]

\[
\text{max } -12y_1 - 10y_2 - 7y_3 - 4y_4 + 0\lambda_1 + 7
\]

such that

\[
\begin{bmatrix}
-2 & -1 & -1 & -1 & 0 \\
-1 & -1 & 0 & -4 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\lambda_1 \\
\end{bmatrix}
\leq
\begin{bmatrix}
-1 \\
0 \\
0 \\
0 \\
1 \\
0 \\
\end{bmatrix}
\]

\[y_1, y_2, y_3, y_4, \lambda_1 \geq 0\]
\[
\max - 12y_1 - 10y_2 - 7y_3 - 4y_4 + 0\lambda_1 - 6\lambda_2
\]
such that
\[
\begin{bmatrix}
-2 & -1 & -1 & 0 & -1 \\
-1 & -1 & 0 & -4 & 0 \\
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\lambda_1 \\
\lambda_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
1
\end{bmatrix}
\]
\[y_1, y_2, y_3, y_4, \lambda_1, \lambda_2 \geq 0\]

This same series of dual problems results when one forms the dual problems to the reformulated SLGP problem which views the \(\eta\) variables as slack variables.

When one examines the dual problems associated with two consecutive linear programming problems at any stage of the SLGP problem, some similarities and differences become apparent. One obvious difference is that each problem has a different set of right hand side vectors, with these right hand side values related to the consecutive LGP objective function coefficient values. Another difference that occurs when going from one dual problem to the next is that a new variable has been added in the latter of the two problems. The coefficient for the new variable in each row is the previous dual problem's right hand side value for that row, while the new variable's objective function value is the previous dual problem's optimum. Otherwise, the majority
of the objective function as well as the majority of the constraint section remains the same from one problem to the next.

Before discussing an algorithm that uses these similarities and differences, the usefulness of complementary slackness is mentioned. Consider the dual problem at priority level \( i \), \( 1 < i < m \) and suppose an optimum is found with one constraint nonbinding. Because of the nature of the Kuhn-Tucker conditions associated with this problem, the primal variable corresponding to the nonbinding dual constraint must be zero (Sposito (1975)). Within a preemptive priority framework this implies that this primal variable will never enter the basis in subsequent priority level's primal linear programming problems, therefore it could essentially be removed. The removing of this primal variable at priority level \( i \) corresponds to removing the nonbinding constraint in the dual problem at priority level \( i \). Thus, after the optimum of the dual linear programming problem for a given priority is found, all nonbinding constraints can be dropped before moving on to solve the dual linear programming problem at the next priority level.

Therefore, to solve the multidimensional dual problem in a sequential fashion, the following series of steps must be undertaken:

1. Set up the dual linear programming problem associated with priority level one.

2. Solve the current problem. If the current priority level is \( m \), the problem is done and go to step (5). Otherwise, use the optimum as the objective function
coefficient for the new variable in the dual problem at the next priority level.

(3) Use the current right hand side values as row coefficient values for the new variable in the dual problem at the next priority level. Bring in the new right hand side for the problem at the next priority level.

(4) Remove any constraints that were nonbinding in the dual problem just solved in step (2). If the resulting problem has no constraints, the problem is done and go to step (5). Otherwise, increase the priority level by one and go to step (2).

(5) The optimal solution to the LGP problem can be obtained from this final dual linear programming problem.

These steps in solving the sequential multidimensional dual problem are a modified version of the algorithm presented by Ignizio (1983b). Ignizio (1983b) neglects to include the new variable after the problem at each priority level is solved, arguing that the removal of slack rows after solving the problem for a given priority level ensures an optimum for the problem at the next priority level. Using his approach yielded a wrong answer for Example A, indicating the new variable is needed after each iteration. Markowski and Ignizio (1983a) and Ignizio (1982) mention the series of dual linear programming problems that result from the SLGP problem yet they don't take advantage of the
nonnegativity restrictions mentioned in this dissertation, nor do they take advantage of the chance to remove slack rows after each iteration. The algorithm used by this dissertation, then, is essentially an improved and corrected version of previously mentioned algorithms. The actual control program to solve the sequential multidimensional dual problem is presented in Appendix D. IBM Mathematical Programming System Extended package with the Extended Control Language, (MPSX/370 ECL), was used to solve this problem because of this package's combination of flexibility and precision.

The sequential multidimensional dual problem has the potential of reducing itself greatly in size as it iterates through priority levels. To gain a perspective on problem dimensions for the LGP primal and dual sequential problems, consider the following maximum dimensions at each priority level:

<table>
<thead>
<tr>
<th>Priority Level</th>
<th>Sequential LGP Problem</th>
<th>Sequential Multidimensional Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rx(2R+V)</td>
<td>(V+R)xR</td>
</tr>
<tr>
<td>2</td>
<td>(R+1)x(2R+V)</td>
<td>(V+R)x(R+1)</td>
</tr>
<tr>
<td>3</td>
<td>(R+2)x(2R+V)</td>
<td>(V+R)x(R+2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>m</td>
<td>(R+(P-1))x(2R+V)</td>
<td>(V+R)x(R+(P-1))</td>
</tr>
</tbody>
</table>
where

\[ P = \text{the number of priority levels} \]
\[ V = \text{the number of variables in the LGP problem} \]
\[ R = \text{the number of rows in the LGP constraint section} \]

The dimensions for the sequential LGP problem could be reduced to
\[ (R+(k-1)) \times (R+V) \] at each priority level, \( k \), if one uses a reformulated
version as presented in Chapter II. The dimensions of the sequential
multidimensional dual problem presented here do not take into account
that the row dimension could be decreasing at each priority level as
the nonbinding constraints are removed. Thus, when the number of rows
in the LGP problem is large relative to the number of variables, the
sequential multidimensional dual problem at priority level \( m \) has the
potential of being a much smaller problem than the SLGP problem at
priority level \( m \).
IV. APPLICATIONS

A. A Transportation Problem

Goal programming is in use today in a wide variety of areas. Forestry, capital budgeting, industrial and health related fields are to name but a few of the many fields that have looked to goal programming for a means to solve the optimization problems they have encountered. This dissertation first turns to a transportation problem presented in the literature by Lee and Moore (1973) as a benchmark problem for comparing solutions obtained by the three SLGP computer codes and the sequential multidimensional dual code. This transportation problem has numerous optima. This enables the benefits of goal programming to become readily apparent because one moves from optimum to optimum as one moves through the problem's priority levels, ending with the optimum that best satisfies the objectives at all priority levels. Lastly, a means of obtaining a unique $L_1$ solution via goal programming will be presented.

The transportation problem presented by Lee and Moore (1973) is moderately small; a single product is distributed by a company from three warehouse locations to four customers at different locations. Table 4.1 summarizes shipping costs of going from location $i$ to destination $j$ ($X_{ij}$), as well as total supply available at location $i$, $i = 1, 2, 3$, and total demand needed at destination $j$, $j = 1, 2, 3, 4$. Notice demand exceeds supply by 100 units.
Table 4.1. Shipping costs, supply and demand

<table>
<thead>
<tr>
<th></th>
<th>Customer 1</th>
<th>Customer 2</th>
<th>Customer 3</th>
<th>Customer 4</th>
<th>Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Demand</td>
<td>200</td>
<td>100</td>
<td>450</td>
<td>250</td>
<td>1000/900</td>
</tr>
</tbody>
</table>

Of course a main objective of the company is to minimize transportation costs. Another goal is to fill as much customer demand as possible, keeping other factors in mind, even though the company is unable to completely meet these demands. The other factors, together with cost minimization, have been summarized into seven prioritized goals for the company.

Priority One: Meet the entire demand of customer 4.

Priority Two: Ship at least 100 units from warehouse 3 to customer 1.

This minimum shipment level is due to a union agreement.

Priority Three: Meet no less than 80% of the demand of each customer so that the supply is balanced among customers.
Priority Four: Keep transportation cost to no more than 110% of the budgeted figure of $2,950. This figure was determined by a standard cost-minimization transportation method.

Priority Five: Minimize shipping over the route from warehouse 2 to customer 4 because this route is characterized by severe hazards.

Priority Six: Balance the percentage of demand filled between customers 1 and 3.

Priority Seven: Minimize the total transportation costs for goods shipped.

Formulating this problem as a standard linear programming problem with a goal of minimizing transportation cost, specifying all the other goals as constraints, results in an infeasible solution. Thus, the company turned to goal programming as a means to obtaining a solution.

Lee and Moore (1973) specify the goal section to their goal programming problem as the following:

Since supply cannot exceed warehouse capacity, there are three goals associated with supply.
\[ x_{11} + x_{12} + x_{13} + x_{14} + \delta_1^- = 300 \]
\[ x_{21} + x_{22} + x_{23} + x_{24} + \delta_2^- = 200 \]
\[ x_{31} + x_{32} + x_{33} + x_{34} + \delta_3^- = 400 \]

There are four demand goals, and it is assumed one does not wish to overfill customer's demand.

\[ x_{11} + x_{21} + x_{31} + \delta_4^- = 200 \]
\[ x_{12} + x_{22} + x_{32} + \delta_5^- = 100 \]
\[ x_{13} + x_{23} + x_{33} + \delta_6^- = 450 \]
\[ x_{14} + x_{24} + x_{34} + \delta_7^- = 250 \]

In accordance with a union agreement, the company is required to ship at least 100 units from warehouse 3 to customer 1.

\[ x_{31} + \delta_8^- - \delta_8^+ = 100 \]

The company's third priority specifies that at least 80% of a customer's demand be met, thus there are four more goals associated with demand.
The following budget goal is a result of the company's desire to not exceed 110% of the budget found from a standard cost-minimization routine.

\[
x_{11} + x_{21} + x_{31} + \delta_{9}^{-} - \delta_{9}^{+} = 160
\]
\[
x_{12} + x_{22} + x_{32} + \delta_{10}^{-} - \delta_{10}^{+} = 80
\]
\[
x_{13} + x_{23} + x_{33} + \delta_{11}^{-} - \delta_{11}^{+} = 360
\]
\[
x_{14} + x_{24} + x_{34} + \delta_{12}^{-} - \delta_{12}^{+} = 200
\]

The route from warehouse 2 to customer 4 was deemed to be hazardous, therefore the company desired to minimize shipping over this route, which led to the goal:

\[
x_{24} - \delta_{14}^{+} = 0
\]

Priority level six specifies that the percentage of demand filled for customers 1 and 3 be balanced with respect to each other. This results in the goal:

\[
x_{11} + x_{21} + x_{31} - 0.4444444(x_{13} + x_{23} + x_{33}) + \delta_{15}^{-} - \delta_{15}^{+} = 0
\]
Lastly, the company wishes to minimize transportation costs.

\[ 5x_{11} + 2x_{12} + 6x_{13} + 7x_{14} + 3x_{21} + 5x_{22} + 4x_{23} + 6x_{24} + 4x_{31} + 5x_{32} + 2x_{33} + 3x_{34} - d_{16}^+ = 0 \]

Obviously, this last goal will never be met. Rather, it is designed to pick a minimum cost solution if multiple solutions exist at priority level seven. In the above equations

\[ x_{ij} = \text{the amount to be transferred from warehouse i to customer j} \]

\[ d_i^- = \text{underachievement of the goal in equation i} \]

\[ d_i^+ = \text{overachievement of the goal in equation i} \]

Lee and Moore (1973) use these deviation variables in a slightly different manner from that presented in Chapter II; in particular, their use of deviation variables does not guarantee that an infeasible solution will not occur.

The resulting achievement function, according to Lee and Moore (1973), based on priority levels one through seven and on the goals just presented is:

\[
\min \ a' = \{ d_7^-, d_8^-, (d_9^- + d_{10}^- + d_{11}^- + d_{12}^-), d_{13}^+, d_{14}^+, (d_{15}^- + d_{15}^+), d_{16}^+ \}. 
\]
The solution Lee and Moore claim to have achieved from the problem as presented is listed in column 1 in Exhibit 4.1. According to their solution all 900 units of the supply are shipped out. This causes transportation costs to run $115 over the modified budget of $3245, and it causes an imbalance of 30 units in shipment size for customer 1 versus that for customer 3. Unfortunately, this same solution was not duplicated when this particular model was solved using the program listed in Appendix A. The resulting solution is presented in column 2 in Exhibit 4.1. This solution allows that not all 900 supply units are to be shipped out. As a result, all goals but the last are met at a transportation cost of $3170 which is $75 under the modified budget amount. This second solution seems consistent with the wording used by Lee and Moore (1973) to describe their goals, especially since they did not devise a goal which specifies that all 900 supply units must be shipped. If one removes the deviation variables $\delta_1$ through $\delta_6$ from the problem, keeping the equality conditions intact, one essentially dictates that all 900 supply units must be shipped. The solution that results when this new problem is solved using the program in Appendix A is listed in column 3 in Exhibit 4.1. Here the actual shipments from warehouse i to location j differ from those obtained by Lee and Moore (1973), but the same goals are achieved. As in their result, transportation costs overrun the modified budget amount by $115, to a cost of $3360. Also, there is an imbalance of 30 units between shipments to customers 1 and 3.
This transportation problem is next reformulated to a form consistent with that described for goal programming problems as presented in Chapter II. The goal section is modified slightly from that presented by Lee and Moore (1973). In general, both a negative and positive (η and ρ) deviation variable are added to each goal, except for the supply constraints and the first three demand constraints. These six constraints are left as inequalities since their deviation variables will not be used in any part of the achievement function. Note that since these six comprise absolute constraints they could have been assigned deviation variables whose values would have been minimized in a priority level preceding the current priority level one. The new goals are now presented in the same order as they were earlier, from supply through minimal transportation cost goals.

\[
\begin{align*}
X_{11} + X_{12} + X_{13} + X_{14} &\leq 300 \\
X_{21} + X_{22} + X_{23} + X_{24} &\leq 200 \\
X_{31} + X_{32} + X_{33} + X_{34} &\leq 400 \\
X_{11} + X_{21} + X_{31} &\leq 200 \\
X_{12} + X_{22} + X_{32} &\leq 100 \\
X_{13} + X_{23} + X_{33} &\leq 450 \\
X_{14} + X_{24} + X_{34} + η_1 - ρ_1 & = 250 \\
X_{31} + η_2 - ρ_2 & = 100
\end{align*}
\]
The achievement function is also modified to make it consistent with the method of minimizing deviation variables as presented in Chapter II. Thus, the new achievement function is the following:

\[
\min a' = \{(n_1 + \rho_1), n_2, (n_3 + n_4 + n_5 + n_6), \rho_7, (n_8 + \rho_8), (n_9 + \rho_9), (n_{10} + \rho_{10})\}
\]

The reasoning for this function is as follows. Priority levels one, five, six and seven have equality with their respective aspiration levels as the desired goal, thus one minimizes the sum of the positive and negative deviation variables at the appropriate level. The goal at priority levels two and three is to be greater than their respective
aspiration levels, thus one minimizes the negative deviation variable at these levels. Finally, the goal of priority level four is to be less than its aspiration level, so the positive deviation variable is minimized.

This problem was solved using all three SLGP programs described in Appendices A, B and C and the results are listed in columns 4, 5 and 6 of Exhibit 4.1. Two different solutions detailing how much is shipped from warehouse i to location j result, but all three have all goals but the last satisfied with a resulting transportation cost of $3170. This corresponds to the result that was found when testing the original problem formulation by Lee and Moore (1973) (actually the MPSX solution is exactly the same as that previous solution). As before, not all 900 units in storage are destined to be shipped, and the cost of $3170 is $75 less than the amount allowed within the modified budget.

This reformulated problem was also solved by the sequential multi-dimensional dual program presented in Appendix D. The dual problem to the reformulated transportation problem is listed in Exhibit 4.2. A third solution that satisfies all goals but the last is found by the dual program. This solution, listed in column 7 of Exhibit 4.1, also has a transportation cost of $3170 and does not require that all 900 units be transferred from storage.

It is obvious that there is a tradeoff between cost and the number of units shipped. These last four solutions may be unacceptable, depending on the views of company management, because not all 900 units are shipped when there is a demand for 1000 units. It was seen that
when all 900 units are shipped, costs overran the modified budget allowance (refer to column 3 Exhibit 4.1). On the other hand, the last four solutions had budgets under that which was required, therefore one could ship more units than is indicated by the last four solutions. To pursue this within the reformulated model, a new goal (associated with priority eight) was constructed which required that the number of units shipped be 900:

\[ \sum_{i,j} x_{ij} + \sum_{i} \pi_i^7 = 900 \]

and the corresponding function in the achievement function is: \( \min \eta_{17} + \rho_{17} \). If one places this goal after the current priority level three and before the current priority level four, one achieves a solution with cost overruns, but with all 900 units being shipped. This solution is listed in column 8 Exhibit 4.1. It is the same solution as that recorded by Lee and Moore (1973) with a transportation cost of $3360. If one places this goal after current priority level six but before current priority level seven one achieves a solution at the border of the modified budget aspiration of $3245, yet with not all 900 units being shipped (see column 9 Exhibit 4.1). This solution specifies that 876 units are to be shipped, which is 26 more than the number being shipped from the solutions first found using the reformulated model. Thus, the company is faced with a tradeoff between minimizing cost and shipping all 900 units when it specifies the other goals it would like to achieve. All 900 units cannot
be shipped within the allowed modified budget framework and still allow all of the other goals to be achieved. Goal programming allows the company to play with goals and priority levels to achieve a solution that is optimal or best for their objectives.

B. $L_1$ Estimation

An important problem in statistics is the study of obtaining estimates for the linear model

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{e}$$

where $\mathbf{y}$ is a given $n \times 1$ vector, $\mathbf{X}$ is a given $n \times p$ matrix and $\beta$ is an unknown $p \times 1$ vector. The classical approach used to determine $\beta$ is to obtain a vector which

$$\min_{\beta} \sum_{i=1}^{n} (y_i - x_i' \beta)^2$$

where $x_i'$ is the $i^{th}$ row of $\mathbf{X}$. This estimator, $\hat{\beta}$, is denoted in the literature as the least squares estimator, $L_2$. It has been noted by many, Dielman (1984), Edgeworth (1888), and Pfaffenberger and Dinkel (1978), that even though the $L_2$ estimator has many useful properties it is not resistant to outliers in the data or to heavy-tailed error distributions. Thus, more robust estimators have been considered in these cases. One such alternate estimator, the $L_1$ estimator, can be obtained by determining a vector $\hat{\beta}$ satisfying:
\[ \min_{\beta} \sum_{i=1}^{n} |y_i - x_i'\beta| \]

i.e., minimize the sum of absolute deviations. This estimator has a number of useful characteristics; for one it is highly resistant to outliers (Harter (1976)) and as shown by Sposito (1982) it can be made to be unbiased. Unfortunately, one unappealing characteristic in \( L_1 \) estimation is the existence of multiple solutions in a given model. As noted by Harter (1976), this problem in \( L_1 \) estimation is more serious than the nonuniqueness that could arise in least squares problems since the concept of estimability makes nonuniqueness less disturbing in least squares. Moreover, uniqueness in least squares depends in a simple way on the rank of the design matrix \( X \), whereas no such simple characteristic of \( X \) determines uniqueness in \( L_1 \) problems. A given data set could yield a unique \( L_1 \) estimator but adding just one additional observation will often yield a nonunique \( L_1 \) estimate for the same model.

Until recently another unappealing characteristic was that the lack of efficient computational procedures had restricted the use of \( L_1 \) estimators in practical data analysis. Charnes et al. (1955) showed that the \( L_1 \) problem could be formulated and solved as the following linear programming problem:

\[ \min \ e^+ + e^- \]

such that \( X\beta + e^+ - e^- = y \)
\[ e^+, e^- > 0 \]
where $e^+_1(e^-_1)$ denotes the positive (negative) vertical deviation above (below) any particular hyperplane at the $i^{th}$ observation. In 1973, Barrodale and Roberts (1973) presented an extremely efficient modified simplex algorithm to obtain $L_1$ estimates. Further improvements in obtaining $L_1$ estimates have since been made by McCormick and Sposito (1976) and Josvanger and Sposito (1983), among others. This dissertation explores the possibility of using goal programming together with the $L_1$ linear programming formulation to alleviate the previously mentioned problem of multiple solutions.

Suggestions have been put forth for choosing which solution to use in the case of multiple $L_1$ solutions. Edgeworth (1888), one of the early supporters of $L_1$ estimation, suggested for the simple linear model, that the "middle" line be taken if multiple solutions exist. In several situations which "middle" line to take is unclear or nonunique. Harter (1976) proposed, for the simple linear model, that one should select the line which is the locus of the points whose vertical distances from the two limiting lines are equal. Gentle (1977) proposed that a unique $L_1$ estimator could be obtained if one conditioned on the $L_2$ solution space, i.e., an $L_1$ estimator closest to the least squares estimator ($L_1/L_2$). Actually, as will be shown, conditioning on the $L_2$ space does not guarantee a unique estimator but leads one in that direction. Fortunately, even though the $L_1$ estimator is not necessarily unique even in the full rank case, all members of the $L_1$ solution space enjoy the robustness properties of the $L_1$ estimators. Therefore, as various conditions are considered to obtain a unique solution, the $L_1$
properties are not lost.

Recall from Chapter II that the preemptive priority structure of goal programming requires that as the linear programming problem at each priority level is solved, the previous priority level's optimum must not be violated. So, consider the location model \( y = \beta + \epsilon \) with a set of four observed values: \(-2, -1, 1, 10\). The \( L_1 \) solution for this set of data is any \( \beta \in [-1, 1] \). In practice the user would have to pick an \( L_1 \) estimator based on his favorite criteria, i.e., the mid-median, the lowest value, or the largest value in the interval. With goal programming this criteria could be incorporated into the model before a solution is sought. For example, suppose the user desired that \(|\beta|\) be minimized in the case of multiple \( L_1 \) solutions. The resulting model is

\[
\min a' = \{(1'e^+ + 1'e^-), (\eta_1 + \rho_1)\}
\]

such that

\[
\begin{align*}
\beta + \eta_1 - \rho_1 &= 0 \\
\epsilon^+ + \epsilon^- &= y \\
\epsilon^+, \epsilon^-, \eta_1, \rho_1 &> 0
\end{align*}
\]

Priority level one ensures that an \( L_1 \) solution is obtained and priority level two seeks to minimize \(|\beta|\). Note that if the solution obtained after the problem at priority level one is solved is unique, then necessarily the optimum at priority level two would yield this same unique solution. In the above example this unique estimator corresponds to the mid-median.
For a general linear model one could consider various criteria for reducing the \( L_1 \) solution space, hopefully to a single vector. Consider the criteria proposed by Gentle (1977): the \( L_1 \) estimator closest to the least squares estimator, call it the \( L_1/L_2 \) estimator. "Closest" in this sense is in terms of absolute differences. For the general linear model this would look like:

\[
\min \ a' = \{(1' e^+ + 1' e^-), (1' \eta + 1' \rho)\}
\]

such that \( X\hat{\beta} + Ie^+ - Ie^- = y \)
\[
I\hat{\eta} + I\eta - I\rho = \hat{\beta}
\]
\( e^+, e^-, \eta, \rho > 0 \)

where the criteria that \( \sum_{j=1}^{p} |\beta_j - \hat{\beta}_j| \) be minimized has been incorporated into priority level two. \( \hat{\beta} \) is the least squares estimator. Gentle (1977) notes that this estimator is unique, however, the following example shows that multiple solutions can still exist. Consider the simple linear regression model \( y = \beta_1 + \beta_2 x + e \) where

\[
\begin{array}{c|ccccc}
  x & -1 & 0 & 0 & 2 & 2 \\
  y & 1 & 2 & -2 & 2 & 0 \\
\end{array}
\]

As shown in Exhibit 4.3, there exists two extreme \( L_1 \) solutions,
\[ y^{(1)} = \frac{4}{3} + \frac{1}{3}x \text{ and } y^{(2)} = \frac{2}{3} - \frac{1}{3}x \]

as well as any convex combination of the two which yields an optimum \( a^*_1 = 6 \) for priority level one. The least squares estimator for this data is \( \hat{\beta}' = (\frac{1}{2}, \frac{1}{6}) \). When this problem is solved using model 4.1 multiple solutions still exist. For example, \( \hat{\beta}^{*'} = (\frac{2}{3}, -\frac{1}{3}) \) and \( \hat{\beta}^{*'} = (1, 0) \) as well as any convex combination of the two are optimal for model 4.1. To obtain a unique \( L_1/L_2 \) estimator, one would need to incorporate other criteria at levels lower than priority level two. In a majority of cases, though, model 4.1 would yield a unique estimator.

Many other criteria exist that one could consider using at priority levels three, four and so on. In the simple linear model example above, one could \( \max \beta_2 \) as the criteria for priority level three to obtain the unique solution \( \hat{\beta}^{*'} = (1, 0) \). See Exhibit 4.3. The model incorporating this extra criteria would be:

\[
\min a' = \{(1'e^+ + 1'e^-), (n_1 + n_2 + p_1 + p_2), n_3\}
\]
such that
\[
\begin{align*}
\beta_1 - \beta_2 + e_1^+ - e_1^- &= 1 \\
\beta_1 + e_2^+ - e_2^- &= 2 \\
\beta_1 + e_3^+ - e_3^- &= -2 \\
\beta_1 + 2\beta_2 + e_4^+ - e_4^- &= 2 \\
\beta_1 + 2\beta_2 + e_5^+ - e_5^- &= 0
\end{align*}
\]
\[ \beta_1 + \eta_1 - \rho_1 = 1/2 \]
\[ \beta_2 + \eta_2 - \rho_2 = 1/6 \]
\[ \beta_3 + \eta_3 - \rho_3 = 1/6 \]
\[ e_i^+, e_i^- > 0 \quad i = 1, 2, 3, 4, 5; \quad \eta_j, \rho_j > 0 \quad j = 1, 2, 3 \]

It is simply up to the user to choose the appropriate and useful criteria for their problem, and to incorporate these criteria into the appropriate priority levels.

One possible objection a user could have with model 4.1 is the need to know the least squares estimator before using it to obtain a solution. To avoid this problem and still obtain an \( L_1/L_2 \) estimate of \( \beta \), one could solve for the least squares estimator within the goal programming problem. The resulting model would be:

\[
\begin{align*}
\min \quad & \mathbf{a}' = \{ (1 e^+_i + 1 e^-_i), (1 \eta_i + 1 \rho_i) \} \\
\text{such that} \quad & X \hat{\beta} = X'y \\
& X\hat{\beta} + Ie^+_i - Ie^-_i = y \\
& I\hat{\beta} - I\hat{\beta} + I\eta - I\rho = 0 \\
& e^+_i, e^-_i, \eta_i, \rho_i > 0
\end{align*}
\]

In this model \( \beta \) and \( \hat{\beta} \) are both unknown vectors to be determined.

The normal equations are used as rigid constraints to guarantee that
the resulting value for \( \hat{\beta} \) is the least squares estimate. If we apply this model to the previous data set (4.2), our goal programming problem would be:

\[
\min \quad a' = \{(1 \ e^+ + 1 \ e^-), \ (\eta_1 + \eta_2 + \rho_1 + \rho_2)\}
\]

such that

\[
\begin{align*}
5\hat{\beta}_1 + 3\hat{\beta}_2 &= 3 \\
3\hat{\beta}_1 + 9\hat{\beta}_2 &= 3 \\
\beta_1 - \beta_2 + e^+_1 - e^-_1 &= 1 \\
\beta_1 + e^+_2 - e^-_2 &= 2 \\
\beta_1 + e^+_3 - e^-_3 &= -2 \\
\beta_1 + 2\beta_2 + e^+_4 - e^-_4 &= 2 \\
\beta_1 + 2\beta_2 + e^+_5 - e^-_5 &= 0 \\
\beta_1 - \beta_1 + \eta_1 - \rho_1 &= 0 \\
\beta_2 - \beta_2 + \eta_2 - \rho_2 &= 0 \\
e^+_i, \ e^-_i &> 0 \quad i = 1, 2, 3, 4, 5 \\
\eta_j, \rho_j &> 0 \quad j = 1, 2
\end{align*}
\]

There are still multiple solutions to this problem, therefore further criteria would need to be incorporated to obtain a unique solution.

To summarize, goal programming allows the user to make use of criteria which can lead to a desirable, unique \( L_1 \) solution. These criteria are added to the problem by minimizing appropriate goal deviation variables in priority levels other than one. Priority one guarantees that the obtained solution is in the \( L_1 \) solution space. The choice of which criteria to apply is up to the user.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Lee &amp; Moore model</th>
<th>Reformulated model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>From article</td>
<td>Test one</td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>100</td>
<td>80</td>
</tr>
<tr>
<td>$X_{13}$</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>$X_{14}$</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>$X_{21}$</td>
<td>90</td>
<td>60</td>
</tr>
<tr>
<td>$X_{22}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$X_{23}$</td>
<td>110</td>
<td>140</td>
</tr>
<tr>
<td>$X_{24}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$X_{31}$</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$X_{32}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$X_{33}$</td>
<td>250</td>
<td>50</td>
</tr>
<tr>
<td>$X_{34}$</td>
<td>50</td>
<td>250</td>
</tr>
<tr>
<td>Priority</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Two</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Three</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Four</td>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>Five</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Six</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>Seven</td>
<td>3360</td>
<td>3170</td>
</tr>
<tr>
<td>Eight</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Exhibit 4.1 Results for the transportation problem
\[
\begin{align*}
\text{max} & \quad 300y_1 - 200y_2 - 400y_3 - 200y_4 - 100y_5 - 450y_6 - 250y_7 - 100y_8 - 160y_9 - 80y_{10} - 360y_{11} \\
& \quad - 200y_{12} - 3245y_{13} + \{250, 100, 800, 0, 0, 0, 0\} \\
\begin{bmatrix}
-1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -5 & 0 & -1 & -5 \\
-1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -2 & 0 & -2 \\
-1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & -6 & 0 & .7 & -6 \\
-1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & -7 & 0 & -7 \\
0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -3 & 0 & -1 & -3 \\
0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -5 & 0 & -5 \\
0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & -4 & 0 & .7 & -4 \\
0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & -6 & -1 & 0 & -6 \\
0 & 0 & -1 & -1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & -4 & 0 & -1 & -4 \\
0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -5 & 0 & -5 \\
0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & -2 & 0 & .7 & -2 \\
0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & -3 & 0 & 0 & -3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6 \\
y_7 \\
y_8 \\
y_9 \\
y_{10} \\
y_{11} \\
y_{12} \\
y_{13} \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & -1 & 0 & 0 & 0 & -1 & -5 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -2 \\
0 & 0 & -1 & 0 & 0 & 0 & .7 & -6 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -7 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -3 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -5 \\
0 & 0 & -1 & 0 & 0 & 0 & .7 & -4 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -6 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -4 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -5 \\
0 & 0 & -1 & 0 & 0 & 0 & .7 & -2 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & -3 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\end{align*}
\]
\[y_i \geq 0 \quad i = 1, 2, \ldots, 13 \quad \bar{q} = .444\ldots\]

Exhibit 4.2 Reformulated transportation problem dual
\[ y = \frac{4}{3} + \frac{1}{3}x \]
\[ y = \frac{1}{2} + \frac{1}{6}x \]
\[ y = \frac{2}{3} - \frac{1}{3}x \]

\( \tilde{y} \) indicates an \( L_1 \) line

\( \hat{y} \) indicates an \( L_2 \) line

\( \hat{z} \)

\( y \) indicates an \( L_1/L_2 \) line

Exhibit 4.3 Solutions to data set 4.2
V. SUMMARY AND EXTENSIONS OF GOAL PROGRAMMING

The usability of an optimal solution to a multiple objective problem found via goal programming depends, primarily, on the successful choosing of priority levels for the goals and on accurately choosing aspiration levels. All four computational solution procedures presented in the Appendices allow the user to rearrange priority levels and resolve his goal programming problem with minimal effort if he should decide his original choice in priority levels was in error. Descriptions of the proper way to do this are detailed in the Appendices, but briefly, in the MPSX and MPSX/370 SLGP programs of Appendices A and B this involves a rearrangement in the ROWS section of the data and an alteration of the OBJS constant in the control program, while in the MPSX/370 ECL SLGP and sequential dual LGP programs of Appendices C and D it simply involves a change in the data entry line of the control program which specifies either objective function order or right hand side order. This chapter will discuss solution changes that occur as aspirations change and will then summarize the benefits of the goal programming model and the computational solutions, presented in this dissertation.

One of the possibly more difficult aspects of forming a goal programming model is choosing the aspiration levels associated with the objectives. The user must choose these values based on his knowledge of the system under study. If the user's aspirations at higher priority levels are too lofty, the solution space is very quickly narrowed down, often to a point, and the lower priority levels contribute little or
nothing to the solution. This idea was mentioned previously with respect to rigid constraints. If \( a_1 > 0 \), assuming the traditional goal programming form is used, then the rigid constraints have right hand sides which are too restricting. In general, if at priority level \( i \), \( a_i > 0 \), solutions at lower priority levels choose from a very small subset of solutions, if not being restricted to a single solution for the remainder of the priority levels. On the other hand, if aspirations are set too loosely the resulting optimal solution may not be satisfactory. The user must strive to select aspirations which, if met, result in an optimal solution which is usable.

Once an optimal solution is found, the user may wonder what changes in this solution would result if small changes were made in his aspiration levels. This leads one into the area of sensitivity analysis of goal programming problems which is not much different from the sensitivity analysis of linear programming discussed by Sposito (1975) and which is extensively discussed by Ignizio (1982) and Leon (1985). A few points relative to changes that occur when one changes the aspiration levels will be brought out here. This chapter will investigate small changes allowed with the restriction that the current basis remains feasible.

Let

\[
\begin{align*}
b^0 & = \text{the initial vector of goal right hand sides which consists of original constraint right hand sides and objective function aspirations} \\
x^0_B & = \text{the initial vector of basic variable values}
\end{align*}
\]
$B^{-1}$ = the solution basis inverse

$a^o$ = the initial optimal achievement function

$U$ = the matrix of coefficients from all priority levels of the achievement function associated with the current basis

we know

$$x^o_B = B^{-1}b^o$$

$$a^o = Ux^o_B$$

Suppose we want to investigate the effects of

$$b^1 = b^o + qd; \text{ i.e.}$$

a new vector of goal right hand side values. The new vector of solutions is $x^1_B = B^{-1}b^1$, and to remain feasible we need $x^1_B > 0$.

$$x^1_B > 0$$

thus,

$$x^o_B + B^{-1}qd > 0$$

and

$$B^{-1}qd > -x^o_B$$

hence,

$$qd > -Bx^o_B = -BB^{-1}b^o = -b^o$$

Therefore, a change of $qd > -b^o$ will preserve the present feasible solution. The impact of this new $b^1$ on the achievement function can
be found by computing \( a^1 = Ux_B \).

For example, reconsider Example A presented in Chapter I and suppose the user wanted to know the affect on the solution of simultaneously decreasing the first original constraint right hand side by two units and of changing the aspiration level of objective one from 7 units to 8 units. Thus, letting \( d = 1 \),

\[
\begin{bmatrix}
12 \\
10 \\
7 \\
4
\end{bmatrix}
\]

\[
-\begin{bmatrix}
10 \\
8 \\
4
\end{bmatrix}
\]

\[
-\begin{bmatrix}
-2 \\
0 \\
1 \\
0
\end{bmatrix}
\]

The new solution will remain feasible since \( qd \geq -b^o \). Solving Example A we find that

\[
B^{-1} = \begin{bmatrix}
1/2 & 0 & 0 & -1 \\
-1/2 & 1 & 0 & 0 \\
-1/2 & 0 & 1 & 0 \\
1/2 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
\rho_4 \\
\eta_2 \\
\eta_3 \\
x_1
\end{bmatrix} = \begin{bmatrix}
2 \\
4 \\
1 \\
6
\end{bmatrix}
\]

and

\[
U' = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
a^o_1 \\
a^o_2 \\
a^o_3 \\
a^o_4
\end{bmatrix} = \begin{bmatrix}
0 \\
1 \\
2
\end{bmatrix}
\]
therefore,
\[
\begin{bmatrix}
76 \\
3 \\
5 \\
1 \\
5
\end{bmatrix}
\]
and
\[
\begin{bmatrix}
0 \\
3 \\
1
\end{bmatrix}
\]

Further restricting the first goal and putting higher expectations on the third goal improved the position with respect to priority level three, yet degraded the solution at priority level two. The optimum values of the decision variables changed from \(x_1^* = 6, x_2^* = 0\) to \(x_1^* = 5, x_2^* = 0\).

For a new solution vector, \(x_B^1\), associated with a changed right hand side vector, \(b^1\), to improve the achievement function, we need \(a^1 < a^0\).

\[
a^1 < a^0
\]

thus,
\[
U\begin{bmatrix}1 \\ x_B^1\end{bmatrix} < a^0
\]

thus,
\[
U(x_B^0 + B^{-1}qd) < a^0
\]

thus,
\[
a^0 + U B^{-1}qd < a^0
\]

hence,
\[
U B^{-1}qd < 0
\]

This result can be used to investigate what change in \(b^0\) will allow an optimum of \(a_1^1 = 0\) when the current value of \(a_1^0 > 0\). This can be
beneficial when optimum $a_1^0$ is positive because the user can determine
what changes need to be made to satisfy the rigid constraints. Consider
Example A where $a_2^{0'} = (0, 1, 2)$. Since $a_1^0 = 0$, we will investigate the
change in goal right hand sides necessary to achieve $a_2^1 = 0$. The
general procedure is the same for investigation of any $a_i$. Because
$a_2^0 > 0$ this will require finding $q_d$ such that $(U^'_B^{-1}q_d)_2 = -a_2^0$.

Let $d = 1$.

Then, $U^'_B^{-1}q = \begin{bmatrix} 0 \\ -\frac{1}{2}q_1 + q_3 \\ \frac{1}{2}q_1 - q_4 \end{bmatrix}$

In general, then, to get an improved solution the user needs:

$q_3 \leq \frac{1}{2}q_1 \leq q_4$.

For this change to yield a feasible solution, recall $q_d \geq -b^0$ is needed,
therefore, one desires that $q_1 \geq -12$, $q_2 \geq -10$, $q_3 \geq -7$ and $q_4 \geq -4$.
The more specific request was for the improved solution to have $a_2^1 = 0$,
which further requires $-\frac{1}{2}q_1 + q_3 = -1$. There is no unique $q$ that
satisfies these conditions.

$q_1^1 = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}$ and $q_1^2 = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ are two possible $q$ vectors that
do satisfy all conditions.
With \( q^1 \) the new right hand side is

\[
\begin{bmatrix}
12 \\
10 \\
6 \\
4
\end{bmatrix}
\]

which yields

\[
\begin{bmatrix}
2 \\
4 \\
0 \\
6
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
0 \\
0 \\
2
\end{bmatrix}
\]

Thus, if the user desires to completely satisfy priority level two, a number of options for changing aspiration levels to do so are available.

Thus, if the user desires to completely satisfy priority level two, a number of options for changing aspiration levels to do so are available.

The discussion so far has briefly dealt with the user's concern over a change in aspiration levels after a solution is found. It is quite possible for the user to be unsure of which aspiration level to use at the problem formulation stage. As stated before, if the goal is a minimization (maximization) the user should choose an aspiration level low (high) enough so if the aspiration is met exactly the solution would be satisfactory. If the goal is to be an equality, the user could enter the goal in an interval form rather than trying to pinpoint an exact aspiration level. Suppose the user desires goal i to be of the
form $\sum_{j=1}^{n} c_{ij} x_j = b_i$, yet is unclear of an exact value for $b_i$. Instead, the user finds it easier to assign a range to $b_i$: $t_1 \leq b_i \leq t_2$ where $t_1$ and $t_2$ are known values. There is more than one way to account for this interval. After assigning deviation variables the user could create two goals:

$$\sum_{j=1}^{n} c_{ij} x_j + \eta_i^1 - \rho_i^1 = t_1$$

$$\sum_{j=1}^{n} c_{ij} x_j + \eta_i^2 - \rho_i^2 = t_2$$

and minimize the appropriate sum of deviation variables, $\min \eta_i^1 + \rho_i^2$, in the achievement vector at the priority level assigned to this goal.

In another method one modifies the range before using it with the goal.

The range $t_1 \leq b_i \leq t_2$

becomes $0 \leq b_i' \leq t_2 - t_1$, where $b_i = b_i' + t_1$.

The goal $\sum_{j=1}^{n} c_{ij} x_j = b_i$

becomes $\sum_{j=1}^{n} c_{ij} x_j = b_i' + t_1$.
Recall that $t^\cdot_1$ is known and $b^\cdot_1$ and thus, $b^\cdot_i$ are unknown. After assigning deviation variables the user has the goal:

$$
\sum_{j=1}^{n} c_{ij} x_j - b^\cdot_i + \eta^\cdot_i - \rho^\cdot_i = t^\cdot_1
$$

where $b^\cdot_i$ is a variable with an upper bound of $t^\cdot_2 - t^\cdot_1$. In the appropriate priority level in the achievement vector, the user would minimize $\eta^\cdot_i + \rho^\cdot_i$.

The first method requires two goals and four deviation variables as well as the usual goal decision variables, but it is rather straightforward. The second method requires some problem manipulation, but one then has one goal with two deviation variables, plus a third bounded variable along with the usual goal decision variables. Thus, the interval format can alleviate some of the user's difficulty in choosing an aspiration level.

Goal Programming as a multiple objective method of problem solving is gaining wider acceptance and use. It provides a means to find a usable, satisfying solution in the face of conflicting goals, rather than forcing one to accept the optimum to one goal subject to the possibly excess degradation of other goals. The preemptive priority structure which encompasses the goals seems reasonable in that this ranking type of decision making is not uncommon in every day life.

With regard to this, the existence of large scale computational procedures on Iowa State University's campus should be useful to
researchers with large multiple objective problems. If the number of goals is small when compared to the number of decision variables the user would want to use one of the SLGP problems of Appendices A, B, or C. The user could find one computational procedure to work no matter which of the three MPSX systems is currently available at ISU. If the number of goals is large when compared to the number of decision variables the user would probably want to solve the multidimensional dual problem and use the program given in Appendix D. It is hoped that these programs are a benefit to many researchers at ISU.
VI. BIBLIOGRAPHY


VII. ACKNOWLEDGEMENTS

I wish to acknowledge and thank Dr. Vince Sposito for his care and encouragement while I pursued this degree and for his guidance with this dissertation. Working with him over the past few years has enriched my life.

I'd like to thank Dr. Keith McRoberts for suggesting the topic for this dissertation, I have enjoyed it, and for his enthusiasm over its results.

Thanks go to Darlene Wicks for her seemingly infinite patience and her skill in typing this manuscript.

A special thanks goes to two very dear Christian friends of mine, Cindy Isenhower and Jean Jensen, for their love and support over the years.

Finally, thanks go to my husband, Steve, for his confidence in me, for his wonderful friendship and his love. Without him around to provide encouragement, this would have been a difficult job.
APPENDIX A

Section A1
IBM's Mathematical Programming System-(MPSX) was used as the major optimization procedure to solve the sequential linear goal programming problem (SLGP). An auxiliary FORTRAN subroutine was developed and augmented to MPSX via READCOMM, Read Communications Format procedure, to allow the form of the SLGP problem to be implementable in MPSX. This combination of programs allows the computational power of MPSX to be made available to solve SLGP problems. Appendix A, Section A3 contains a listing of the MPSX Control Program, the FORTRAN subroutine, an example of an input data file which resides on disk, listings of REVDATA, the revision data set on disk, and examples of SOLUTION's output.

As described in Chapter II, a SLGP problem is a series of linear programming problems solved in an order determined by some user specified priority levels. After the linear programming problem for each priority level is solved its original constraint set must be augmented with a new equality constraint, which consists of the latest objective function with a right hand side equal to the newly found objective function optimum. These new equality constraints ensure that the solutions to any previous priority levels are not violated, and thus maintain the preemptive priority structure of the goal programming problem. Any software package that solves a SLGP problem, then, would need to be able to keep track of the total number of priority levels in a particular problem, to be able to keep track of the current priority level, to be able to accurately solve a linear programming problem, and to be able to update the problem with new constraints created from old
objective functions and their optima as the linear programming problem for each priority level is solved. MPSX together with a FORTRAN subroutine was able to meet these criteria.

Before describing the MPSX control program and the FORTRAN subroutine, the input data set and the revise data set, REVDATA, will be described. The input data for the SLGP problem should be put on a file in traditional MPSX form with a data set NAME line, sections for ROWS, COLUMNS, and RHS, and an ENDDATA line. All information within each section should be placed in appropriate file columns as described in any MPSX user's guide (1972). For the data to be used with this set of programs, the data's name, listed on the NAME card, should be SLGPDAT. In general, all names must be eight alphameric characters or less, and special characters may be used except the '$' which may not be used as the first character in a name. Within the ROWS section all objective function names must be listed in the order in which they are to be solved and be placed before any row names are listed. The objective function names must be five characters in length, but trailing blanks are allowed. In the RHS section, the problem's right hand side name must be Bl. Nothing out of the ordinary is needed for the COLUMNS section, though the user needs to remember to include the data for all of the objective functions defined for their problem. An example of an input data set is Example A, listed in Exhibit A3. Example A has three priority levels, four rows and ten columns. Notice that the data is correctly named SLGPDAT and the right hand side is named Bl. All three objective function names, OBJ1, OBJ2,
and OBJ3, are listed first in the ROWS section while their column data is appropriately placed in the COLUMNS section.

The revise data set must contain a NAME line, ROWS and RHS sections, and an ENDATA line. As with the input data, the revision information must be placed in the appropriate MPSX file columns. To be able to use the revise data set with the MPSX and FORTRAN programs, the data's name on the NAME card must be REVN. The illustration in Exhibit A4, which is a listing of the revise file after Example A priority level one is solved, best explains the remainder of the file. The ROWS section indicates that objective function one will be modified to an equality. The RHS section indicates what value the new constraint row's right hand side is to take. In general, the only change after the linear programming problem for each new priority level is solved will be the objective function name in line 4 and line 7 and the right hand side value in line 7. Since the MPSX and FORTRAN programs make these changes for the user after solving the problem for one priority level and before going on to the next priority level, the user need only set up an initial revise file. This initial file must be of the form of that in Exhibit A4, though since it will be modified before it is used the name listed in lines 4 and 7 and the value in line 7 need not be meaningful. Thus, the revise file in Exhibit A4 is the file that was used before the linear programming problem at priority level one was solved, as well as being the resulting file after this problem was solved. Exhibits A5 and A6 show the revise file after the problems at priority levels two and three are solved.
Notice the change in names and right hand side values. These right hand side values are the optima at these priority levels.

In Exhibit A1 one will find a listing of the MPSX control program, more specifically the control program used to solve Example A, while a line by line detailed program description is in Section A2. The beginning and end of the program consist of initialization statements. The constants to be used in the program are defined in lines 51 through 54. The constant COUNT, initialized in line 53, is equal to the total number of priority levels minus two and is decremented by one at each new priority level. The constant OBJS, initialized in line 54, gives the sequence of objective function names associated with the different priority levels. MPSX will update the name of the current objective function to its appropriate value at each priority level. Recall that the objective function names must be five characters long. If a name is shorter than length five, trailing blanks must be added to extend the total length to five. (See Exhibit A1, line 54 and the corresponding line explanation in Section A2 for clarification.) The constant PLEVEL, initialized in line 52, keeps track of the current priority level as MPSX is working. The constant ALL1, initialized in line 51, is used by MPSX when assigning new objective function names. While the constants are defined at the end, the beginning of the program, lines 8 through 16, is given over to defining the program, assigning files, and initializing communication region cells. COMMFTMT, a file defined in line 9, will contain the solution to the linear programming problem
after it is solved for each priority level. This solution is used by the FORTRAN subroutine to set up the new constraint to be augmented to the SLGP problem before the linear programming problem associated with the next priority level is solved. The ASSIGN statement in line 10 indicates that the input data, referred to as ALLDAT in the program, can be found where the user has indicated on the FT12F001 DD card, line 61. Likewise, the revise data, referred to as REVDAT by the program, is assigned (line 11) to be that which is found where indicated by the FT13F001 DD card, line 62. Next, the objective function name, the data's internal name, the right hand side name and the data name are moved to the appropriate communication region cells.

After defining all of these constants, files, and communication region cells, the MPSX control program is set to solve the linear programming problem associated with priority level one. The objective function for priority level one is minimized subject to the constraints specified, using the PRIMAL optimizing procedure. The optimal basis associated with the optimum at priority level one is saved for the user as a starting point when it comes time to solve the problem associated with priority level two. Storing the optimal basis for use at the next priority level will save on computational time since this means restarting at a near-optimal feasible solution. The solution to the problem at priority level one is then printed and stored in file COMMFM for use by the FORTRAN subroutine. Control is transferred to the FORTRAN subroutine via the procedure GOAL. GOAL, the procedure created by the
FORTRAN subroutine, uses the current priority level whose value is passed in the parameter PLEVEL to set up the current objective function as an equality constraint with a right hand side value equal to the current optimum. In other words, GOAL creates the revise file using information passed in PLEVEL and the SOLUTION information found in COMMFMT before it passes control back to MPSX. GOAL will be described in further detail later. After control is passed back to MPSX, the priority level is updated to two. At this stage, the MPSX statements necessary to solve the linear programming problems associated with the remaining priority levels are set up in a loop. For the problem associated with a given priority level, the appropriate names for the objective function, the revision data, the oldname on PROBFIE and the name on PROBFIE are assigned to their appropriate MPSX cells. The right hand side name does not change, therefore it is never reassigned. The problem is modified according to the changes found in REVDAT, the optimal basis saved from the previous priority level's linear programming problem is restored, and the current objective function is minimized by the optimization procedure PRIMAL. The optimal basis associated with the optimum to this problem is saved, to be used when solving the problem at the next priority level. The solution to the linear programming problem at this priority level is then printed and stored in file COMMFMT. Again, control is passed to GOAL so that the new revise file can be created. Control is restored to MPSX and the priority level is updated by one. MPSX uses COUNT to check to see if the linear programming
problems associated with all of the priority levels has been solved. If yes, the program terminates, otherwise MPSX loops through lines 30 through 49 again, continuing until all problems for all priority levels are solved.

The FORTRAN subroutine, called by the procedure GOAL, provides the means for this set of two programs to augment the initial SLGP problem with a new constraint as the linear programming problem for each priority level is solved. A listing of the FORTRAN subroutine can be found in Exhibit A2. When control is passed to the FORTRAN subroutine, the communications file is positioned to the beginning of the second array found in file COMMFMT via the statement CALL POSITN (FILE, INDIC, NR). FILE was set earlier to the value 4 which is the same as that for COMMFMT in the MPSX control program (Exhibit A1, line 9). NR was set to the value 2 indicating that the first array, the SOLUTION array, is to be skipped. The current priority level value is transferred from the MPSX control program in the CALL GETARG (PLEVEL) statement. The next line, CALL ARRAY (FILE, INDIC, SARRAY), initiates reading of the second array, RSECTION, while the succeeding CALL VECTOR (FILE, INDIC, VALUES) statements transfer one row of elements from RSECTION to the FORTRAN program at a time. The variable INDIC in each CALL is an integer variable whose value is set by READCOMM to specify something about the data being transferred, but it is not used by GOAL. SARRAY is a variable whose value is the name of the array to which the file is positioned. VALUES is a one dimensional array which contains the
transferred row elements. The number of rows transferred from array RSECTION depends on the current priority level, and it is only the values from the current priority level that are stored for further use. This is because the information stored in array RSECTION is in the same order, row by row, as the order defined in the ROWS section in the input data, and it is the value of the current priority level's objective function which resides in its ACTIVITY level that is needed by GOAL to set up the revise data set. Exhibits A7, A8, and A9 contain listings of SOLUTION's output for Example A, priority levels one, two and three respectively. SECTION 1 - ROWS contains the same data as that which is stored in array RSECTION. GOAL iterates through the results for each objective function until it reaches the one associated with the current priority level. The need for the user to place the objective functions in their appropriate order in the ROWS section of the input data should be apparent. Once GOAL reaches the current priority level it is ready to write the revise file on disk. When GOAL writes out the revise file it uses VALUES(1), the current objective function name, and VALUES(2), the value of this objective function. Lastly, GOAL writes the value of this objective function on hard copy, which is printed at the end of all output from MPSX.

This set of programs, the MPSX control program and the FORTRAN subroutines, requires a small amount of user preparation before using them to solve a SLGP problem. The JCL for the programs (Exhibit A1, lines 1 through 6, 56 through 62 and Exhibit A2, lines 1 through 4 and 47 through
57) is that which is compatible with Iowa State University's NAS9160 computer and with the author's computer account. Modifications may be necessary in the JCL statements, but specifically at ISU a user needs to supply their own computer account number, bin number, and data set names. The user must choose a disk name for the library that will contain the procedure GOAL, and a disk name for the revise data set. The library disk name used here is L.U9229.ELAC (Exhibit A1, line 57, Exhibit A2, line 53) while the revise data set disk name is L.U9229.REVDATA (Exhibit A1, line 62 and Exhibit A2, line 48). Notice that the revise data set is to be modified when the FORTRAN subroutine is run, (DISP= (MOD, KEEP)), therefore the user must create an initial revise data set and save it under the appropriate name before running either program. The FORTRAN program need only be run once for the procedure GOAL to be ready and available for use in solving any SLGP problem. The library containing GOAL is created during this run and it is linked to the MPSX control program each time the control program is run. The user needs to supply the input data set's disk name in Exhibit A1, line 61 (Example A was stored on disk under L.U9229.LDATA(ORIG408)). Some information pertinent to the solving of the user's SLGP problem must be supplied in the MPSX control program in lines 53 and 54, and this is explained in Section A2 of Appendix A.
Section A2
MPSX Control Program

PROGRAM('ND')

PROGRAM indicates the start of the MPSX control program. The parameter 'ND' tells MPSX not to terminate with an error message when it encounters the calls to GOAL.

INITIALZ

INITIALZ is a macro that establishes the standard processing of all demands by setting tolerances and output frequencies to standard values.

ASSIGN('COMMFMT', 'FT04F001', 'COMM')

This statement associates the control program filename COMMFMT with the dataset specified in the JCL DD statement FT04F001. This DD statement (Exhibit A1, line 60) indicates that this dataset is not permanent on disk, it is used for the duration of the execution of the two programs. COMMFMT is the name used by the control program to reference these data. COMM specifies the data are stored in communications format.

ASSIGN('ALLDAT', 'FT12F001', 'CARD')

This statement associates the control program filename ALLDAT with the dataset specified in the JCL DD statement FT12F001. This DD statement reveals the name the user has assigned to the input data stored on disk whereas ALLDAT is the name used by the control program to reference
these data. Exhibit A1, line 61 shows that the input data for Example A is on disk with the name L.U9229.LDATA(ORIG408). The parameter CARD indicates the data are not stored in communications format.

```plaintext
ASSIGN('REVDAT', 'FT13F001', 'CARD')
```

This statement associates the control program filename REVDAT with the dataset specified in the JCL DD statement FT13F001. This DD statement indicates that the revise dataset's name on disk is L.U9229.REVDATA (Exhibit A1, line 62) which is the same name, necessarily, used by the FORTRAN subroutine (Exhibit A2, line 48). REVDAT is the name used by the control program to reference the revise dataset and CARD indicates the data are not stored in communications format.

```plaintext
MVADR (ALL1, OBJS)
```

This statement moves the address of the constant OBJS into the constant ALL1. These constants had initial values declared in lines 54 and 51 respectively.

```plaintext
MOVE(XPBNMAE, 'SLGPDAT')
```

This statement moves the problem name SLGPDAT into the communication region cell XPBNMAE. SLGPDAT is now the name of the problem on PROBFILE.

```plaintext
MOVE (XRHS, 'B1')
```

This statement moves the right hand side column name, B1, into the communication region cell XRHS.
MOVE (XDATA, 'SLGPDAT')
This statement moves the input dataset name, SLGPDAT, into the communication region cell XDATA.

MVIND (XOBJ, ALL1,5)
This statement moves the next five elements of the constant OBJS, whose beginning address is in ALL1, into the communication region cell XOBJ. This establishes the name of the current objective function.

PREPOUT ('COMMfmt')
PREPOUT prepares the file COMMfmt for output. It positions the file at the start of the data and opens the file for output.

CONVERT ('FILE', 'ALLDAT')
CONVERT instructs MPSX to read the input data found in the file assigned to ALLDAT and convert it to packed binary form on the PROBFILe. This internal representation of the problem is written on PROBFILe with the name SLGPDAT, the name found in XPBNAME.

BCDOUT
BCDOUT is used to get a listing of the complete problem at its present stage.

SETUP ('MIN')
SETUP initiates a solution to the problem, SLGPDAT, minimizing the objective function.
PRIMAL

PRIMAL is one of MPSX's optimizing procedures. It first finds a feasible solution, then an optimal solution.

SAVE

SAVE stores the current optimal basis, bounded variable status, and part of the communication region on PROBFILE. The communication region cells saved are listed in the MPSX user's guide (1972).

SOLUTION

In this case, SOLUTION prints the current solution in a tabled format.

SOLUTION ('FILE', 'COMMFM'T', 'RSECTION', '2/4D', 'CMASKS', ' ')

In this case, SOLUTION stores the specified portions of the current solution in the file COMMFM'T. 'CMASKS', ' ' instructs SOLUTION to omit the CSECTION array altogether. 'RSECTION', '2/4D' instructs SOLUTION to include only the row names and their activity levels in double precision from the RSECTION array on the file COMMFM'T. The SOLUTION array is automatically stored in COMMFM'T.

FREECORE

This statement releases core obtained by MPSX procedures for work areas and I/O buffers in case the FORTRAN procedure GOAL should exceed the usual MPSX procedure space requirement.
GOAL (PLEVEL)
GOAL is the name of the FORTRAN procedure which writes the current solution onto the revise dataset for use by MPSX in solving the problem at the next priority level. The value of the parameter PLEVEL, indicating the current priority level, is passed to the FORTRAN subroutine.

PLEVEL = PLEVEL + 1
This statement increments PLEVEL by one.

ALLl = ALLl + 5
This statement increments ALLl by five.

GOTO (LOOP)
This statement transfers program control to the line labelled LOOP.

MORE TALLY (COUNT,LOOP)
This statement causes MPSX to decrement the constant COUNT by one and then compare the value of COUNT to zero. If it is not zero program control is transferred to the line labelled LOOP, otherwise, the program proceeds.

GOTO (OUT)
This statement transfers program control to the line labelled OUT.
MOVE (XDATA, 'REVN')
This statement moves the revise dataset name, REVN, into the communication region cell XDATA.

MOVE (XOLDNAME, 'SLGPDAT')
This statement moves the old dataset name, SLGPDAT, into the communication region cell XOLDNAME.

REVISE ('FILE', 'REVDAT')
REVISE modifies the problem SLGPDAT on the PROBFILE according to the revisions recorded in the dataset associated with REVDAT. The problem is stored back on PROBFILE under the name SLGPDAT.

RESTORE
RESTORE brings back the last optimal basis from PROBFILE and reinverts this basis.

GOTO (MORE)
This statement transfers program control to the line labelled MORE.

OUT EXIT
This statement causes the program to exit MPSX.
ALL1 DC(0)
This statement declares the variable ALL1 to have the value zero. ALL1 is used by the MPSX control program when it changes objective function names at each iteration.

PLEVEL DC(1)
This statement declares the variable PLEVEL to have the value of one. PLEVEL keeps track of the current priority level.

COUNT DC(1)
This statement declares the value of the variable COUNT, which should be set equal to the number of priority levels minus two. Therefore, before running the program the user should place his problem's appropriate value for COUNT within the parentheses. In this case, this listing was used to solve Example A which has three priority levels, giving COUNT a value of one.

OBJS DC('OBJ1 ', 'OBJ2 ', 'OBJ3 ')
This statement declares the objective function names in the preemptive order in which they are to be solved, which must correspond with their order in the ROWS section of the input data. The user will need to update this line so it is correct for each problem solved. The objective function names must be five characters long, which accounts for the trailing blank character after each name in Exhibit A1, line 54.
The names must be listed within the parentheses, each within quotes, separated by commas. Line 54 gives the objective function names for Example A, where the linear programming problems are solved in an order corresponding to their objective function names.

PEND

This statement indicates the end of the MPSX control program.

FORTRAN Subroutine

INTEGER FILE, INDIC, NR, PLEVEL

This statement declares the following variables to be integer valued.

FILE is the variable that will define the file containing the SOLUTION information.

INDIC is an indicator variable needed by MPSX in the CALL statements, but it is not used by GOAL.

NR is an indicator variable that will point to the array stored in COMFPMT to be used by GOAL.

PLEVEL is the variable that will contain the current priority level. It is passed from the MPSX control program to the FORTRAN subroutine.
INTEGER*2 Y(12)
This statement declares an array, Y, of size 12 to be integer valued with 2 bytes each. Y will contain character data.

DOUBLE PRECISION SARRAY, VALUES(2)
This statement declares the following variables to be double precision.

SARRAY  is a character variable whose value will be the name of the array being used from COMMFMT.

VALUES(2)  is an array of two variables. VALUES(1) will contain the name of the row being transferred from COMMFMT while VALUES(2) will contain the activity level of this row.

FILE=4
NR=2
These statements initialize the variables FILE and NR to the values 4 and 2, respectively.

CALL POSITN (FILE, INDIC, NR)
This statement positions the communication file, COMMFMT, to the beginning of its second array.

CALL GETARG (PLEVEL)
This statement transfers the value of the parameter PLEVEL to the FORTRAN program from the control program.
CALL ARRAY (FILE, INDIC, SARRAY)
This call initiates reading of the second array. SARRAY will contain
the name of this second array.

DO 10 I=1, PLEVEL
    CALL VECTOR (FILE, INDIC, VALUES)
10 CONTINUE
This DO loop causes READCOMM to transfer one row of elements from the
second array to the FORTRAN program at each iteration. It is in the
last time through the loop when the array VALUES contains the correct
information, the information for the current priority level.

WRITE(13, 601) VALUES(1), VALUES(1), VALUES(2)
601 FORMAT ('NAME', 10X, 'REVN'/ 'ROWS'/2X,'MODIFY'/1X,'E',2X,A5,
        1/ 'RHS'/2X,'MODIFY'/4X,'Bl', 8X, A5, 5X, E12.6, '/ENDATA')
These statements write out the revise file on disk. VALUES(1) is
the current objective function name and VALUES(2) is the optimal value
of this objective function.

REWIND 13
This statement repositions the revise file REVDATA to its beginning
so that it can be used again by the FORTRAN subroutine or the MPSX
control program.

IF (VALUES(2).EQ.0.00) GOTO 20
If VALUES(2) is zero the revise file is left as originally written and
the subroutine is continued at line 20. Otherwise, control is
passed to the following READ statement.
This statement reads the revise file. VALUES(1) contains the current objective function name. The array Y contains the current objective function optimum in character format. This is done so the 'D' used by FORTRAN in E12.6 format can be replaced by an 'E' which can be recognized by MPSX.

These statements rewrite the revise file on disk with the appropriate format corrections made, as noted above.

These statements write the priority and solution for the current objective function on hard copy.

These statements signal the end of the FORTRAN subroutine.

This statement causes the Linkage Editor to insert READCOMM into the load module.
ENTRY MAIN
This statement identifies the entry point of the FORTRAN load module.

NAME GOAL(R)
This statement identifies GOAL as the FORTRAN procedure name. The parameter R causes any old GOALs to be replaced by the new one being created.
Section A3
Exhibit 1. SLGP program using MPSX
`Exhibit A1. (Continued)

1. //LEEANN JOB U9229,LAC
2. /*JOBPARM BIN=515
3. //S1 EXEC FORTGCL
4. //FORT.SYSIN DD *
5. C
6. C FT13F001 DD-NAME FOR REVISED FILE
7. C FT06F001 OUTPUT DD-NAME
8. C FT04F001 DD-NAME OF FILED SOLUTION IN COMMUNICATIONS FORMAT
9. C
10. INTEGER FILE,INDIC,NR,PLEVEL
11. INTEGER*2 Y(12)
12. DOUBLE PRECISION SARRAY,VALUES(2)
13. C
14. C FILE = 4
15. NR=2
16. CALL POSITN ( FILE,INDIC,NR )
17. C
18. C GET THE SOLUTION
19. C
20. CALL GETARG(PLEVEL)
21. CALL ARRAY ( FILE,INDIC,SARRAY )
22. DO 10 I=1,PLEVEL
23. CALL VECTOR ( FILE,INDIC,VALUES )
24. 10 CONTINUE

Exhibit A2. FORTRAN subroutine augmented to MPSX`
25. C
26. C WRITE OUT THE REVISED FILE
27. C
28. WRITE(13,601) VALUES(1),VALUES(1),VALUES(2)
29. 601 FORMAT('NAME',10X,'REVN'/5X,'ROWS'/2X,'MODIFY'/1X,'E',2X,A5,
30. '1/RHS'/2X,'MODIFY'/4X,'B1',8X,A5,5X,E12.6,/'ENDATA')
31. REWIND 13.
32. IF (VALUES(2).EQ.0.00) GOTO 20
33. C
34. READ(13,501) VALUES(1),VALUES(1),Y
35. 501 FORMAT(/,/,/4X,A8,/,/,/,14X,A8,2X,12A1,/) 
36. REWIND 13
37. WRITE(13,603) VALUES(1),VALUES(1),Y(1),Y(2),Y(3),Y(4),Y(5),
38. 1Y(6),Y(7),Y(8),Y(10),Y(11),Y(12)
39. 603 FORMAT('NAME',10X,'REVN'/5X,'ROWS'/2X,'MODIFY'/1X,'E',2X,A8,
40. '1/RHS'/2X,'MODIFY'/4X,'B1',8X,A8,2X,8A1,'E',3A1,/'ENDATA')
41. REWIND 13
42. C
43. 20 WRITE(6,602) PLEVEL,VALUES(2)
44. 602 FORMAT(' ', 'THE SOLUTION TO PRIORITY LEVEL',15,' IS',F12.6)
45. RETURN
46. END
47. //_OVERRIDE ON SYSLIB NEEDED TO AVOID SUBROUTINE "ARRAYS" CONFLICT
48. //FT13F001 DD DSN=L.U9229.REVDATA,UNIT=DISK,DISP(MOD,KEEP),
49. // DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(5,1),RLSE)
50. /*
51. //LKED.SYSLIB DD DSNAME=SYS1.MPSXSUB,DISP=SHR
52. // DD DSNAME=SYS1.FORTLIB,DISP=SHR
53. //LKED.SYSLMOD DD DSNAME=L.U9229.ELAC,UNIT=DISK,
54. // VOLUME=SER=UCC001,LABEL=EXPDT=99350,
55. // DISP=(NEW,CATLG),
56. // SPACE=(800,(10,10,1),RLSE)
57. /*
58. /*
59. **
60. **

Exhibit A2. (Continued)
Exhibit A3. Input to MPSX for Example A, residing in file L.U9229.LDATA(ORIG408)
Exhibit A4. FORTRAN output/revised MPSX input for Example A residing on file L.U9229.REVDATA

This particular file is the input and output from priority level one, as well as being the input at priority level two.

Exhibit A5. FORTRAN output/revised MPSX input for Example A residing on file L.U9229.REVDATA

This particular file is the output from priority level two and the input to priority level three.
Exhibit A6. FORTRAN output/revised MPSX input for Example A residing on file L.U9229.REVDATA

This particular file is the output from priority level three.
SOLUTION (OPTIMAL)

...NAME... ...ACTIVITY... DEFINED AS
FUNCTIONAL OBJ1
RERAINTS B1

SECTION 1 - ROWS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>ROW</th>
<th>AT</th>
<th>ACTIVITY</th>
<th>SLACK ACTIVITY</th>
<th>DUAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBJ1</td>
<td>BS</td>
<td>.</td>
<td>3.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>2</td>
<td>OBJ2</td>
<td>BS</td>
<td>.</td>
<td>3.00000-</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>OBJ3</td>
<td>BS</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>ROW1</td>
<td>EQ</td>
<td>.</td>
<td>12.00000</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>ROW2</td>
<td>EQ</td>
<td>.</td>
<td>10.00000</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>ROW3</td>
<td>EQ</td>
<td>.</td>
<td>7.00000</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>ROW4</td>
<td>EQ</td>
<td>.</td>
<td>4.00000</td>
<td>.</td>
</tr>
</tbody>
</table>

SECTION 2 - COLUMNS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>COLUMN</th>
<th>AT</th>
<th>ACTIVITY</th>
<th>INPUT COST</th>
<th>REDUCED COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>C1</td>
<td>BS</td>
<td>.</td>
<td>4.00000</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>C2</td>
<td>LL</td>
<td>.</td>
<td>4.00000</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>C3</td>
<td>BS</td>
<td>.</td>
<td>6.00000</td>
<td>.</td>
</tr>
<tr>
<td>11</td>
<td>C4</td>
<td>BS</td>
<td>.</td>
<td>3.00000</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>C5</td>
<td>BS</td>
<td>.</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>14</td>
<td>C7</td>
<td>LL</td>
<td>.</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>15</td>
<td>C8</td>
<td>LL</td>
<td>.</td>
<td>1.00000</td>
<td>1.00000</td>
</tr>
<tr>
<td>A</td>
<td>C9</td>
<td>LL</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>C10</td>
<td>LL</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

*LOWER LIMIT and UPPER LIMIT columns were omitted.

Exhibit A7. MPSX SOLUTION output*. Example A, priority level one
SOLUTION (OPTIMAL)

...NAME...  ...ACTIVITY...  DEFINED AS
FUNCTIONAL  1.00000  OBJ2
RESTRAINTS  B1

SECTION 1 - ROWS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>ROW</th>
<th>AT</th>
<th>ACTIVITY</th>
<th>SLACK ACTIVITY</th>
<th>DUAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBJ1</td>
<td>EQ</td>
<td>1.00000</td>
<td>1.00000-</td>
<td>1.00000</td>
</tr>
<tr>
<td>2</td>
<td>OBJ2</td>
<td>BS</td>
<td>2.00000</td>
<td>2.00000-</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ROW1</td>
<td>EQ</td>
<td>12.00000</td>
<td></td>
<td>.50000</td>
</tr>
<tr>
<td>5</td>
<td>ROW2</td>
<td>EQ</td>
<td>10.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ROW3</td>
<td>EQ</td>
<td>7.00000</td>
<td></td>
<td>1.00000-</td>
</tr>
<tr>
<td>7</td>
<td>ROW4</td>
<td>EQ</td>
<td>4.00000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SECTION 2 - COLUMNS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>COLUMN</th>
<th>AT</th>
<th>ACTIVITY</th>
<th>INPUT COST</th>
<th>REDUCED COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>C1</td>
<td>BS</td>
<td>6.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>C2</td>
<td>LL</td>
<td></td>
<td></td>
<td>.50000</td>
</tr>
<tr>
<td>10</td>
<td>C3</td>
<td>LL</td>
<td></td>
<td></td>
<td>.50000</td>
</tr>
<tr>
<td>11</td>
<td>C4</td>
<td>BS</td>
<td>4.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C5</td>
<td>BS</td>
<td>1.00000</td>
<td>1.00000</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>C6</td>
<td>LL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>C8</td>
<td>LL</td>
<td></td>
<td></td>
<td>.50000</td>
</tr>
<tr>
<td>16</td>
<td>C9</td>
<td>LL</td>
<td></td>
<td></td>
<td>1.00000</td>
</tr>
<tr>
<td>17</td>
<td>C10</td>
<td>BS</td>
<td>2.00000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*LOWER LIMIT and UPPER LIMIT columns were omitted.

Exhibit A8. MPSX SOLUTION output*. Example A, priority level two
SOLUTION (OPTIMAL)

...NAME... ...ACTIVITY... DEFINED AS
FUNCTIONAL 2.00000 OBJ3
RERAINTS B1

SECTION 1 - ROWS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>...ROW...</th>
<th>AT</th>
<th>...ACTIVITY...</th>
<th>SLACK ACTIVITY</th>
<th>.DUAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>OBJ1</td>
<td>EQ</td>
<td>1.00000</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OBJ2</td>
<td>EQ</td>
<td>1.00000</td>
<td>.</td>
<td>1.00000</td>
</tr>
<tr>
<td>3</td>
<td>OBJ3</td>
<td>BS</td>
<td>2.00000</td>
<td>2.00000-</td>
<td>1.00000</td>
</tr>
<tr>
<td>A</td>
<td>ROW1</td>
<td>EQ</td>
<td>12.00000</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>ROW2</td>
<td>EQ</td>
<td>10.00000</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ROW3</td>
<td>EQ</td>
<td>7.00000</td>
<td>.</td>
<td>1.00000-</td>
</tr>
<tr>
<td>7</td>
<td>ROW4</td>
<td>EQ</td>
<td>4.00000</td>
<td>.</td>
<td>1.00000</td>
</tr>
</tbody>
</table>

SECTION 2 - COLUMNS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>.COLUMN.</th>
<th>AT</th>
<th>...ACTIVITY...</th>
<th>.INPUT COST..</th>
<th>.REDUCTED COST.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>C1</td>
<td>BS</td>
<td>6.00000</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>C2</td>
<td>LL</td>
<td>.</td>
<td>.</td>
<td>4.00000</td>
</tr>
<tr>
<td>10</td>
<td>C3</td>
<td>BS</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>11</td>
<td>C4</td>
<td>BS</td>
<td>4.00000</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>12</td>
<td>C5</td>
<td>BS</td>
<td>1.00000</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>13</td>
<td>C6</td>
<td>LL</td>
<td>.</td>
<td>.</td>
<td>1.00000</td>
</tr>
<tr>
<td>14</td>
<td>C7</td>
<td>BS</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>A</td>
<td>C8</td>
<td>LL</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>16</td>
<td>C9</td>
<td>LL</td>
<td>.</td>
<td>.</td>
<td>1.00000</td>
</tr>
<tr>
<td>17</td>
<td>C10</td>
<td>BS</td>
<td>2.00000</td>
<td>1.00000</td>
<td>.</td>
</tr>
</tbody>
</table>

*LOWER LIMIT and UPPER LIMIT columns were omitted.

Exhibit A9. MPSX SOLUTION output*. Example A, priority level three
APPENDIX B

Section B1
The second version of the program that solves the Sequential Linear Goal Programming (SLGP) problem was created using IBM's Mathematical Programming System Extended 370 (MPSX/370). The control language specific to MPSX/370, MPS, was augmented with a FORTRAN subroutine created using READCOMF to allow the SLGP problem to be implementable in MPSX/370. READCOMF is the Read Communications Format procedure for MPSX/370, comparable to READCOMM in MPSX. Appendix B, Section B3 contains a listing of the control program, a listing of the FORTRAN subroutine, an example of an input data file which resides on disk, and a listing of REV370, the revision dataset. The characteristics of the SLGP problem and of the program that can solve it were discussed in Chapter II and Appendix A, Section A1, respectively.

The input dataset and the revision dataset are comparable to those used for the MPSX program described in Appendix A. As before the input data should be on a file in traditional MPSX form with a dataset NAME line, sections for ROWS, COLUMNS, and RHS, and an ENDATA line. Also, as before, all information should be placed in appropriate file columns as described in the user's guide (1979b). In general, all names must be eight alphameric characters or less, and special characters may be used except the $ which may not be used as the first character in a name. Specifically, the data's name, listed on the NAME line, must be SLGPDAT. Objective function names must be of the form OBJn where n can range from 1 to 99. Within the ROWS section all objective function names must be listed in the order in which they are to be solved and must be placed before any row names.
The right hand side name, in the RHS section, must be Bl. Even though nothing unusual is needed for the COLUMNS section the user needs to include the data for all of the objective functions defined previously in the ROWS section. An example of an input dataset is the reformulated transportation problem of Chapter IV, listed in Exhibit B3. This example follows all of the rules for this program and thus can be solved by this version of the SLGP program. The data are correctly named SLGPDAT with a right hand side named Bl. All seven objective function names, OBJ1, OBJ2, OBJ3, OBJ4, OBJ5, OBJ6 and OBJ7 are listed first in the ROWS section and their data is placed in the COLUMNS section.

The revise dataset is exactly like the one used by the MPSX program described in Appendix A. Refer to the description in Appendix A, Section A1 together with the illustration in Exhibit B4 to understand the revise dataset. Exhibit B4 contains the revise dataset after priority level seven was solved in the reformulated transportation problem.

In Exhibit B1 one will find a listing of the MPSX/370 control program specifically set up to solve the reformulated transportation problem. A line by line detailed program description is given in Section B2. As in the MPSX program, the beginning and end of this program are used for initialization. The constants are the same as used previously. COUNT, initialized in line 55, is equal to the total number of priority levels minus two and is decremented by one at each new priority level. The constant OBJS, initialized in line 56, gives
the sequence of objective function names associated with the different priority levels. MPSX/370 updates the current objective function name to its appropriate value at each new priority level. The objective function names must be of length five, therefore for names OBJ1 through OBJ9 a trailing blank must be added in this initialization line. PLEVEL, initialized in line 54, keeps track of the current priority level as MPSX/370 sequentially solves the problem. The constant ALL1, initialized in line 53, is used when MPSX/370 assigns new objective function names at each successive priority level. LPLEVEL, initialized to zero in line 52, will be updated when the program starts to a value equal to the total number of priority levels for the current problem. Lines 9, 10 and 11 define the files used by MPSX/370. OUTDATA will contain the solution to the linear programming problem after it is solved for each priority level. This solution is used by the FORTRAN subroutine to set up the new constraint to be augmented to the SLGP problem before the linear programming problem associated with the next priority level is solved. Unlike the solutions in MPSX, those stored in OUTDATA are not stored in Communications Format. Rather, OUTDATA is stored on disk in a manner specified by the MPSX/370 user's guide (1979b). The FT04F001 DD statement, line 61, specifies how OUTDATA is associated with a disk file, also called OUTDATA. The input data, referred to as ALLDAT in the program, is assigned to the disk file indicated by the FT12F001 DD statement, line 64. The revise data, referred to by the program as REVDAT, is associated with the disk file defined in the FT13F001 DD
statement, line 65. Next, the objective function name, the data's internal name, the right hand side name and the data name are moved to the appropriate communication region cells.

The MPSX/370 program is now set to solve the linear goal programming problem. The method used by the MPSX program of Appendix A is the same as that used here except in the way the solution is stored for use by the FORTRAN procedure, GOAL. The MPSX program stores the solution in communications format while the MPSX/370 program stores the solution in file OUTDATA in standard format. Otherwise, refer to Appendix A, Section A1 for a general description of the working of the control program.

Just as in the MPSX program, the FORTRAN subroutine which augments the SLGP problem at each iteration with a new constraint is called by the procedure GOAL. A listing of the FORTRAN subroutine can be found in Exhibit B2. Due to changes IBM made between MPSX and MPSX/370 (a major one being they did away with communications format) this subroutine is quite different from the one given in Appendix A. Instead of over-writing the solution file as the program iterates through priority levels, like READCOMM does in MPSX, READCOMF appends the solution results from the latest priority level to those of the previous priority levels. Thus, when control is passed to the FORTRAN subroutine it first iterates through solutions from all previous priority levels before reaching the solution to the problem at the latest priority level. When it reaches the results for this current priority level it
searches for the solution to the appropriate objective function. This optimum is then written on the revise file and printed for the user. The structure of this file that the FORTRAN procedure is searching through is described in the MPSX/370 user's guide (1979b). The value of the current priority level as well as the number of priority levels are passed to the FORTRAN subroutine from the control program via the statement CALL GETARG(PELEVEL,LPLEVEL) to enable the successful completion of the search for the current optimum.

The MPSX/370 control program and the FORTRAN subroutine require a small amount of user preparation before using them to solve a SLGP problem. The JCL for the programs (Exhibit B1 lines 1 through 6 and 58 through 65 and Exhibit B2 lines 1 through 5 and 48 through 62) is compatible with the author's computer account for Iowa State University's NAS9160 computer. Modifications in the JCL may be necessary for all users but specifically at ISU the user needs to supply their computer account number, bin number, and dataset names. The user must choose a disk name for the library that will contain the procedure GOAL, and a disk name for the revise dataset. The library disk name used here is L.U9229.LAC370 (Exhibit B1, line 59 and Exhibit B2, line 54) while the revise dataset disk name is L.U9229.REV370 (Exhibit B1, line 65 and Exhibit B2, line 49). Notice that the revise dataset is to be modified when the FORTRAN subroutine is run, (DISP=(MOD,KEEP)), therefore the user must create an initial dataset and save it under the appropriate name before running either program. The FORTRAN program should be run
only once for the procedure GOAL to be ready and available for use in solving any SLGP problem. The library containing GOAL is created during this run and it is linked to the MPSX/370 control program each time the control program is run. The user also needs to supply the input dataset's disk name in Exhibit B1, line 64. Notice that the reformulated transportation problem was stored under L.U9229.LDATA (TRANDAT2). Some information pertinent to the solving of the user's SLGP problem must be supplied in the MPSX/370 control program in lines 55 and 56, and this is explained in Section B2 of Appendix B.
Section B2
MPSX/370 CONTROL PROGRAM

PROGRAM('ND')

PROGRAM indicates the start of the MPSX/370 control program. The parameter 'ND' tells MPSX/370 not to terminate with an error message when it encounters the calls to GOAL.

INITIALZ

INITIALZ is a macro that establishes the standard processing of all demands by setting tolerances and output frequencies to standard values.

ASSIGN('FILE','OUTDATA','JCL','FT04F001')

This statement associates the control program filename OUTDATA with the dataset specified in the JCL DD statement FT04F001. This DD statement (Exhibit B1, line 61) indicates that the name assigned on disk, L.U9229.OUTDATA, is a temporary dataset used for the duration of the execution of the two programs. The data are stored on disk in standard format.

ASSIGN('FILE','ALLDAT','JCL','FT12F001')

This statement associates the control program filename ALLDAT with the dataset specified in the JCL DD statement FT12F001. This DD statement reveals the name the user has assigned to the input data stored on disk.
whereas ALLDAT is the name used by the control program to reference this data. Exhibit B1, line 64 shows that the input data for the reformulated transportation problem is on disk with the name L.U9274.LDATA (TRAN DAT2).

**ASSIGN('FILE','REVDAT','JCL','FT13F001')**

This statement associates the control program filename REVDAT with the dataset specified in the JCL DD statement FT13F001. This DD statement indicates that the revise dataset's name on disk is L.U9229.REV370 (Exhibit B1, line 65) which is the same name used by the FORTRAN subroutine (Exhibit B2, line 49). REVDAT is the name used by the control program to reference the revise dataset.

**MVADR(ALL1,OBJ S)**

This statement moves the location address of the constant OBJ S into the constant ALL1. These constants had initial values declared in lines 56 and 53, respectively.

**MOVE(XPBNAME,'SLGPDAT')**

This statement moves the problem name SLGPDAT into the communication region cell XPBNAME. SLGPDAT is now the name of the problem on PROBFILE.

**MOVE(XRHS,'B1')**

This statement moves the right hand side column name, B1, into the communication region cell XRHS.
MOVE(XDATA,'SLGPDAT')

This statement moves the input dataset name, SLGPDAT, into the communication region cell XDATA.

MVIND(XOBJ,ALL1,5)

This statement moves the next five elements of the constant OBJS, whose beginning address is in ALL1, into the communication region cell XOBJ.
This establishes the name of the current objective function.

LPLEVEL=COUNT+2

This statement uses the value of COUNT, supplied by the user, to update LPLEVEL to its appropriate value. LPLEVEL will now contain the value of the number of priority levels for the current problem.

CONVERT('FILE','ALLDAT')

CONVERT instructs MPSX/370 to read the input data found in the file assigned to ALLDAT and convert it to packed binary form on the PROBFILE.
This internal representation of the problem is written on PROBFILE with the name SLGPDAT, the name found in XPBNAME.

BCDOUT

BCDOUT is used to get a listing of the complete problem at its present stage.
**SETUP ('MIN')**

SETUP initiates a solution to the problem, SLGPDAT, minimizing the objective function.

**PRIMAL**

PRIMAL is one of MPSX/370's optimizing procedures. It first finds a feasible solution, then an optimal solution.

**SAVE**

SAVE stores the current optimal basis, bounded variable status, and part of the communication region on PROBFILE. The communication region cells saved are listed in the MPSX/370's user's guide (1979b).

**SOLUTION**

SOLUTION, in this case, will print the current solution in a tabled format.

```
SOLUTION('FILE','OUTDATA','NAME','VALUE','RMASKS','OBJ**',
' ','CMASKS',' ')
```

In this case, SOLUTION stores the specified portions of the current solution in the file OUTDATA. 'CMASKS',' ' instructs SOLUTION to omit the CSECTION array altogether. 'NAME','VALUE','RMASKS','OBJ**',' ' instructs SOLUTION to include only the row names beginning with OBJ and their associated activity levels from the RSECTION array in the file OUTDATA. The SOLUTION array is automatically stored in OUTDATA.
**FREECORE**

This statement releases core obtained by MPSX/370 procedures for work areas and I/O buffers in case the FORTRAN procedure GOAL should exceed the usual MPSX/370 procedure space requirement.

**GOAL(PLEVEL, LLEVEL)**

GOAL is the name of the FORTRAN procedure which writes the current solution onto the revise dataset for use by MPSX/370 in solving the problem at the next priority level. The values of the parameter PLEVEL and LLEVEL are passed to the FORTRAN subroutine. PLEVEL is the value of the current priority level while LLEVEL is equal to the number of priority levels.

**PLEVEL=PLEVEL+1**

This statement increments PLEVEL by one.

**ALL1=ALL1+5**

This statement increments ALL1 by five.

**GOTO(LOOP)**

This statement transfers program control to the line labelled LOOP.

**MORE TALLY(COUNT, LOOP)**

This statement causes MPSX/370 to decrement the constant COUNT by one and then compare the value of COUNT to zero. If it is not zero program
control is transferred to the line labelled LOOP, otherwise the program proceeds.

**GOTO(OUT)**
This statement transfers program control to the line labelled OUT.

**MOVE(XDATA,'REVN')**
This statement moves the revise dataset name, REVN, into the communication region cell XDATA.

**MOVE(XOLDNAME,'SLGPDAT')**
This statement moves the old dataset name, SLGPDAT, into the communication region cell XOLDNAME.

**REVISE('FILE','REVDAT')**
REVISE modifies the problem SLGPDAT on the PROBFILE according to the revisions recorded in the dataset associated with REVDAT. The problem is stored back on PROBFILE under the name SLPGDAT.

**RESTORE**
RESTORE brings back the last optimal basis from PROBFILE and reinverts this basis.

**GOTO(MORE)**
This statement transfers program control to the line labelled MORE.
OUT EXIT
This statement causes the program to exit MPSX/370.

LPLEVEL DC(0)
This statement initializes LPLEVEL to zero. LPLEVEL will be updated in the program to contain the total number of priority levels.

ALL1 DC(0)
This statement declares the variable ALL1 to have the value zero. ALL1 is used by the MPSX/370 control program when it changes objective function names at each iteration.

PLEVEL DC(1)
This statement declares the variable PLEVEL to have the value one. PLEVEL keeps track of the current priority level.

COUNT DC(5)
This statement declares the value of the variable COUNT, which should be set equal to the number of priority levels minus two. Therefore, before running the program the user should place their problem's appropriate value for COUNT within the parentheses. In this case, this listing was used to solve the reformulated transportation problem of Chapter IV which has seven priority levels, therefore COUNT has an initial value of five.
OBJS DC('OBJ1 ','OBJ2 ','OBJ3 ','OBJ4 ','OBJ5 ','OBJ6 ','OBJ7 ')

This statement declares the objective function names in the preemptive order in which they are to be solved, which must correspond with their order in the ROWS section of the input data. The user will need to update this line so it is correct for each problem solved. The objective function names must be five characters long, which accounts for the trailing blank character after each name in Exhibit B1, line 56. The names must be listed within the parentheses, each within the quotes, separated by commas. Line 56 gives the objective function names for the reformulated transportation problem of Chapter IV, where the linear programming problems are solved in an order corresponding to their objective function names.

PEND

This statement indicates the end of the MPSX/370 control program.

FORTRAN subroutine

INTEGER FILE, PLEVEL, LLEVEL

This statement declares the following variables to be integer valued.

FILE is the variable that will define the file containing the SOLUTION information.

PLEVEL is the variable that will contain the current priority level. It is passed from the MPSX/370 control program to the FORTRAN subroutine.
LLEVEL is the variable which will contain the number of priority levels. It, too, is passed from the MPSX/370 control program to the FORTRAN subroutine.

```
INTEGER*2 Y(12)
```
This statement declares an array, Y, of size 12 to be integer valued with 2 bytes each. Y will contain character data.

```
DOUBLE PRECISION COLUMN(2),X
```
This statement declares the following variables to be double precision.

- COLUMN(2) is an array of two variables. COLUMN(1) will contain the activity level of the row being transferred from OUTDATA while COLUMN(2) will contain the row name.
- X is a variable which will be used by the FORTRAN program to temporarily store the correct COLUMN(1) value.

```
FILE=4
```
This statement initializes the variable FILE to the value 4.

```
CALL GETARG(PLEVEL, LLEVEL)
```
This statement transfers the values of PLEVEL and LLEVEL from the control program to the FORTRAN subroutine.

```
DO 30 IJK=1,PLEVEL
```
This statement initiates a DO loop which will iterate through PLEVEL sets of solutions before reaching the set associated with the current priority level.
DO 10 I=1,9
READ(FILE)
10 CONTINUE

This loop reads through the beginning information for the current set of solutions. This information is not needed in this program, thus, nothing is stored.

DO 20 I=1,LLEVEL

This statement initiates a DO loop which will iterate through the row names for the current set of solutions. Recall that the only information stored in OUTDATA is for rows whose names begin with OBJ, thus, there is information on LLEVEL rows stored in OUTDATA for each priority level.

IF (IJK.NE.PLEVEL)GO TO 20
IF (I.NE.PLEVEL)GO TO 20

These IF statements check to see if the FORTRAN subroutine is at the correct set of solutions and at the correct OBJ row in that set. If not, control is transferred to line labelled 20. If so, control is passed to the WRITE statements.

WRITE(13,601) COLUMN(2), COLUMN(2), COLUMN(1)
601 FORMAT('NAME',10X,'REVN'/ROWS'/2X,'MODIFY'/IX,'E',2X,A8,
_1/'RHS'/2X,'MODIFY'/4X,'B1',8X,A8,2X,E12.6,'ENDATA')

These statements write out the revise file on disk. COLUMN(2) contains the current objective function name and COLUMN(1) contains the current objective function optimum. Exhibit B4, is an example of REV370 as written by these statements.
WRITE(6,602)PLEVEL, COLUMN(1)
602 FORMAT(' THE SOLUTION TO PRIORITY LEVEL',I5,' IS',E12.6)

These statements write the priority level and optimum for the current objective function on hard copy.

\[ \text{X=COLUMN(1)} \]

The optimum to the current priority level is stored for later use in the variable X.

\[ \text{REWIND 13} \]

This statement repositions the revise file REV370 to its beginning so that it can be used again by the FORTRAN subroutine or the MPSX/370 control program.

\[ \text{20 CONTINUE READ(FILE) READ(FILE) 30 CONTINUE} \]

The CONTINUE statements simply end the DO loops while the READ(FILE) statements finish the iteration through a particular set of solutions.

\[ \text{IF (X.EQ.0.00) GO TO 40} \]

If X is zero the revise file is left as originally written and the subroutine is ended at line 40. Otherwise, control is transferred to the following READ statement.
READ(13,501)COLUMN(2),COLUMN(2),Y
501 FORMAT(//,//,4X,A8,//,//,14X,A8,2X,12A1,/)  

This statement reads the revise file. COLUMN(2) contains the current
objective function name. The array Y contains the current objective
function optimum in character format. This is done so the 'D' used by
FORTRAN in E12.6 format can be replaced by an 'E' which can be
recognized by MPSX/370.

WRITE(13,603)COLUMN(2),COLUMN(2),Y(1),Y(2),Y(3),Y(4),Y(5),
       Y(6),Y(7),Y(8),Y(10),Y(11),Y(12)
603 FORMAT('NAME',10X,'REVN'/2X,'ROWS'/2X,'MODIFY'/1X,'E',2X,A8,
       1/'RHS'/2X,'MODIFY'/4X,'B1',8X,A8,2X,8A1,'E',3A1/'ENDDATA')

These statements rewrite the revise file on disk with the appropriate
format corrections made, as noted above.

40 RETURN
   END

These statements signal the end of the FORTRAN subroutine.

INSERT READCOMF

This statement causes the Linkage Editor to insert READCOMF into the
load module.

ENTRY MAIN

This statement identifies the entry point of the FORTRAN load module.
NAME GOAL(R)

This statement identifies GOAL as the FORTRAN procedure name. The parameter R causes any old GOALS to be replaced by the new one being created.
Section B3
1. //LEEANN JOB U9229,LAC,MSGLEVEL=1
2. */JOBPARM BIN=515
3. /** PROGRAM NAME LEE#GOAL370
4. //STEP1 EXEC MPSX370,TIME,MPSCOMP=(,30),REGION.MPSEXEC=512K,
5. // TIME.MPSEXEC=3
6. //MPSCOMP.SYSIN DD *
7. PROGRAM('ND')
8. INITIALZ
9. ASSIGN('FILE','OUTDATA','JCL','FT04F001')
10. ASSIGN('FILE','ALLDAT','JCL','FT12F001')
11. ASSIGN('FILE','REVDAT','JCL','FT13F001')
12. MVADR(ALL1,OBJJS)
13. MOVE(XPBNAME,'SLGPDAT')
14. MOVE(XRHS,'BL')
15. MOVE(XDATA,'SLGPDAT')
16. MVIND(XOBJ,ALL1,5)
17. LPLEVEL=COUNT+2
18. CONVERT('FILE','ALLDAT')
19. BCDOUT
20. SETUP('MIN')
21. PRIMAL
22. SAVE
23. SOLUTION
24. SOLUTION('FILE','OUTDATA','NAME','VALUE', X
25. 'RMASKS','OBJ**',' ','CMASKS',' ')  
26. FREECORE
27. GOAL(PLEVEL,LPLEVEL)
28. PLEVEL=PLEVEL+1
29. ALL1=ALL1+5
30. GOTO(LOOP)
31. MORE TALLY(COUNT,LOOP)
32. GOTO(OUT)
33. LOOP MVIND(XOBJ,ALL1,5)
34. MOVE(XDATA,'REVN')
35. MOVE(XOLDNAME,'SLGPDAT')
36. MOVE(XPBNAME,'SLGPDAT')
37. REVISE('FILE','REVDAT')
38. BCDOUT
39. SETUP('MIN')
40. RESTORE
41. PRIMAL
42. SAVE
43. SOLUTION
44. SOLUTION('FILE','OUTDATA','NAME','VALUE', X
45. 'RMASKS','OBJ**',' ','CMASKS',' ')  
46. FREECORE
47. GOAL(PLEVEL,LPLEVEL)

Exhibit Bl. SLGP program using MPSX/370
143

Exhibit B1. (Continued)

Exhibit B2. FORTRAN subroutine augmented to MPSX/370
DO 20 I=1,LLEVEL.
23. READ(FILE)COLUMN
24. IF(IJK.NE.PLEVEL) GO TO 20
25. IF (I.NE.PLEVEL) GO TO 20
26. WRITE(13,601)COLUMN(2),COLUMN(2),COLUMN(1)
27. 601 FORMAT('NAME',10X,'REVN'/'ROWS'/2X,'MODIFY'/1X,'E',2X,A8,
28. 1 '/RHS'/2X,'MODIFY'/4X,'BL',8X,A8,2X,E12.6,/'ENDATA')
29. WRITE(6,602)PLEVEL,COLUMN(1)
30. 602 FORMAT(' THE SOLUTION TO PRIORITY LEVEL',15,' IS',E12.6)
31. X=COLUMN(1)
32. REWIND 13
33. 20 CONTINUE
34. READ(FILE)
35. READ(FILE)
36. CONTINUE
37. IF (X.EQ.0.00) GO TO 40
38. READ(13,501)COLUMN(2),COLUMN(2),Y
39. 501 FORMAT(/,/,/,4X,A8,/,/,/,14X,A8,2X,12A1,/) 
40. REWIND 13
41. WRITE(13,603)COLUMN(2),COLUMN(2),Y(1),Y(2),Y(3),Y(4),Y(5),
42. 1Y(6),Y(7),Y(8),Y(10),Y(11),Y(12)
43. 603 FORMAT('NAME',10X,'REVN'/'ROWS'/2X,'MODIFY'/1X,'E',2X,A8,
44. 1'/RHS'/2X,'MODIFY'/4X,'BL',8X,A8,2X,8A1,'E',3A1,/'ENDATA')
45. REWIND 13
46. 40 RETURN
47. END
48. //* OVERRIDE ON SYSLIB NEEDED TO AVOID SUBROUTINE "ARRAY" CONFLICT
49. //FT13FO01 DD DSN=L.U9229.REV370,UNIT=DISK,DISP=(MOD,KEEP),
50. // DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(5,1),RLSE)
51. /*
52. //LKED.SYSLIB DD 
53. // DD DSNAME=SYS2.MPSX.MPSXSUB,DISP=SHR
54. //LKED.SYSLMOD DD DSNAME=L.U9229.LAC370,UNIT=DISK,
55. // VOLUME=SER=UCC001,LABEL=EXPD1=99350,
56. // DISP=(NEW,CATLG),
57. // SPACE=(800,(10,10,1),RLSE)
58. //LKED.SYSIN DD *
59. /*
60. ENTRY MAIN
61. NAME GOAL(R)
62. /*
| 1. NAME | SLGPDAT |
| 2. * TRANSPORTATION PROBLEM, REWRITTEN |
| 3. ROWS |
| 4. N OBJ1 |
| 5. N OBJ2 |
| 6. N OBJ3 |
| 7. N OBJ4 |
| 8. N OBJ5 |
| 9. N OBJ6 |
| 10. N OBJ7 |
| 11. L ROW1 |
| 12. L ROW2 |
| 13. L ROW3 |
| 14. L ROW4 |
| 15. L ROW5 |
| 16. L ROW6 |
| 17. E ROW7 |
| 18. E ROW8 |
| 19. E ROW9 |
| 20. E ROW10 |
| 21. E ROW11 |
| 22. E ROW12 |
| 23. E ROW13 |
| 24. E ROW14 |
| 25. E ROW15 |
| 26. E ROW16 |
| 27. COLUMNS |
| 28. C1 ROW1 1.0 |
| 29. C1 ROW4 1.0 |
| 30. C1 ROW9 1.0 |
| 31. C1 ROW13 5.0 |
| 32. C1 ROW15 1.0 |
| 33. C1 ROW16 5.0 |
| 34. C1 OBJ3 -1.0 |
| 35. C1 OBJ6 -1.0 |
| 36. C1 OBJ7 -5.0 |
| 37. C2 ROW1 1.0 |
| 38. C2 ROW5 1.0 |
| 39. C2 ROW10 1.0 |
| 40. C2 ROW13 2.0 |
| 41. C2 ROW16 2.0 |
| 42. C2 OBJ3 -1.0 |
| 43. C2 OBJ7 -2.0 |
| 44. C3 ROW1 1.0 |
| 45. C3 ROW6 1.0 |
| 46. C3 ROW11 1.0 |
| 47. C3 ROW13 6.0 |

Exhibit B3. Input to MPSX/370 for the reformulated transportation problem. This data is on disk under L.U9229.LDATA(TRANSDAT2)
Exhibit B3. (Continued)
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>95</td>
<td>C8</td>
<td>OBJ7</td>
<td>-6.0</td>
</tr>
<tr>
<td>96</td>
<td>C9</td>
<td>ROW3</td>
<td>1.0</td>
</tr>
<tr>
<td>97</td>
<td>C9</td>
<td>ROW4</td>
<td>1.0</td>
</tr>
<tr>
<td>98</td>
<td>C9</td>
<td>ROW8</td>
<td>1.0</td>
</tr>
<tr>
<td>99</td>
<td>C9</td>
<td>ROW9</td>
<td>1.0</td>
</tr>
<tr>
<td>100</td>
<td>C9</td>
<td>ROW13</td>
<td>4.0</td>
</tr>
<tr>
<td>101</td>
<td>C9</td>
<td>ROW15</td>
<td>1.0</td>
</tr>
<tr>
<td>102</td>
<td>C9</td>
<td>ROW16</td>
<td>4.0</td>
</tr>
<tr>
<td>103</td>
<td>C9</td>
<td>OBJ2</td>
<td>-1.0</td>
</tr>
<tr>
<td>104</td>
<td>C9</td>
<td>OBJ3</td>
<td>-1.0</td>
</tr>
<tr>
<td>105</td>
<td>C9</td>
<td>OBJ6</td>
<td>-1.0</td>
</tr>
<tr>
<td>106</td>
<td>C9</td>
<td>OBJ7</td>
<td>-4.0</td>
</tr>
<tr>
<td>107</td>
<td>C10</td>
<td>ROW3</td>
<td>1.0</td>
</tr>
<tr>
<td>108</td>
<td>C10</td>
<td>ROW5</td>
<td>1.0</td>
</tr>
<tr>
<td>109</td>
<td>C10</td>
<td>ROW10</td>
<td>1.0</td>
</tr>
<tr>
<td>110</td>
<td>C10</td>
<td>ROW13</td>
<td>5.0</td>
</tr>
<tr>
<td>111</td>
<td>C10</td>
<td>ROW16</td>
<td>5.0</td>
</tr>
<tr>
<td>112</td>
<td>C10</td>
<td>OBJ3</td>
<td>-1.0</td>
</tr>
<tr>
<td>113</td>
<td>C10</td>
<td>OBJ7</td>
<td>-5.0</td>
</tr>
<tr>
<td>114</td>
<td>C11</td>
<td>ROW3</td>
<td>1.0</td>
</tr>
<tr>
<td>115</td>
<td>C11</td>
<td>ROW6</td>
<td>1.0</td>
</tr>
<tr>
<td>116</td>
<td>C11</td>
<td>ROW11</td>
<td>1.0</td>
</tr>
<tr>
<td>117</td>
<td>C11</td>
<td>ROW13</td>
<td>2.0</td>
</tr>
<tr>
<td>118</td>
<td>C11</td>
<td>ROW15</td>
<td>-44444444</td>
</tr>
<tr>
<td>119</td>
<td>C11</td>
<td>ROW16</td>
<td>2.0</td>
</tr>
<tr>
<td>120</td>
<td>C11</td>
<td>OBJ3</td>
<td>-1.000</td>
</tr>
<tr>
<td>121</td>
<td>C11</td>
<td>OBJ6</td>
<td>.44444444</td>
</tr>
<tr>
<td>122</td>
<td>C11</td>
<td>OBJ7</td>
<td>-2.0</td>
</tr>
<tr>
<td>123</td>
<td>C12</td>
<td>ROW3</td>
<td>1.0</td>
</tr>
<tr>
<td>124</td>
<td>C12</td>
<td>ROW7</td>
<td>1.0</td>
</tr>
<tr>
<td>125</td>
<td>C12</td>
<td>ROW12</td>
<td>1.0</td>
</tr>
<tr>
<td>126</td>
<td>C12</td>
<td>ROW13</td>
<td>3.0</td>
</tr>
<tr>
<td>127</td>
<td>C12</td>
<td>ROW16</td>
<td>3.0</td>
</tr>
<tr>
<td>128</td>
<td>C12</td>
<td>OBJ1</td>
<td>-1.0</td>
</tr>
<tr>
<td>129</td>
<td>C12</td>
<td>OBJ3</td>
<td>-1.0</td>
</tr>
<tr>
<td>130</td>
<td>C12</td>
<td>OBJ7</td>
<td>-3.0</td>
</tr>
<tr>
<td>131</td>
<td>C13</td>
<td>ROW7</td>
<td>1.0</td>
</tr>
<tr>
<td>132</td>
<td>C14</td>
<td>ROW8</td>
<td>1.0</td>
</tr>
<tr>
<td>133</td>
<td>C15</td>
<td>ROW9</td>
<td>1.0</td>
</tr>
<tr>
<td>134</td>
<td>C16</td>
<td>ROW10</td>
<td>1.0</td>
</tr>
<tr>
<td>135</td>
<td>C17</td>
<td>ROW11</td>
<td>1.0</td>
</tr>
<tr>
<td>136</td>
<td>C18</td>
<td>ROW12</td>
<td>1.0</td>
</tr>
<tr>
<td>137</td>
<td>C19</td>
<td>ROW13</td>
<td>1.0</td>
</tr>
<tr>
<td>138</td>
<td>C20</td>
<td>ROW14</td>
<td>1.0</td>
</tr>
<tr>
<td>139</td>
<td>C21</td>
<td>ROW15</td>
<td>1.0</td>
</tr>
<tr>
<td>140</td>
<td>C22</td>
<td>ROW16</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Exhibit B3. (Continued)
<table>
<thead>
<tr>
<th>Row</th>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>C23</td>
<td>ROW7</td>
</tr>
<tr>
<td>142</td>
<td>C23</td>
<td>OBJ1</td>
</tr>
<tr>
<td>143</td>
<td>C24</td>
<td>ROW8</td>
</tr>
<tr>
<td>144</td>
<td>C24</td>
<td>OBJ2</td>
</tr>
<tr>
<td>145</td>
<td>C25</td>
<td>ROW9</td>
</tr>
<tr>
<td>146</td>
<td>C25</td>
<td>OBJ3</td>
</tr>
<tr>
<td>147</td>
<td>C26</td>
<td>ROW10</td>
</tr>
<tr>
<td>148</td>
<td>C26</td>
<td>OBJ3</td>
</tr>
<tr>
<td>149</td>
<td>C27</td>
<td>ROW11</td>
</tr>
<tr>
<td>150</td>
<td>C27</td>
<td>OBJ3</td>
</tr>
<tr>
<td>151</td>
<td>C28</td>
<td>ROW12</td>
</tr>
<tr>
<td>152</td>
<td>C28</td>
<td>OBJ3</td>
</tr>
<tr>
<td>153</td>
<td>C29</td>
<td>ROW13</td>
</tr>
<tr>
<td>154</td>
<td>C29</td>
<td>OBJ4</td>
</tr>
<tr>
<td>155</td>
<td>C30</td>
<td>ROW14</td>
</tr>
<tr>
<td>156</td>
<td>C30</td>
<td>OBJ5</td>
</tr>
<tr>
<td>157</td>
<td>C31</td>
<td>ROW15</td>
</tr>
<tr>
<td>158</td>
<td>C31</td>
<td>OBJ6</td>
</tr>
<tr>
<td>159</td>
<td>C32</td>
<td>ROW16</td>
</tr>
<tr>
<td>160</td>
<td>C32</td>
<td>OBJ7</td>
</tr>
<tr>
<td>161</td>
<td>RHS</td>
<td>ROW1</td>
</tr>
<tr>
<td>162</td>
<td>B1</td>
<td>ROW2</td>
</tr>
<tr>
<td>163</td>
<td>B1</td>
<td>ROW3</td>
</tr>
<tr>
<td>164</td>
<td>B1</td>
<td>ROW4</td>
</tr>
<tr>
<td>165</td>
<td>B1</td>
<td>ROW5</td>
</tr>
<tr>
<td>166</td>
<td>B1</td>
<td>ROW6</td>
</tr>
<tr>
<td>167</td>
<td>B1</td>
<td>ROW7</td>
</tr>
<tr>
<td>168</td>
<td>B1</td>
<td>ROW8</td>
</tr>
<tr>
<td>169</td>
<td>B1</td>
<td>ROW9</td>
</tr>
<tr>
<td>170</td>
<td>B1</td>
<td>ROW10</td>
</tr>
<tr>
<td>171</td>
<td>B1</td>
<td>ROW11</td>
</tr>
<tr>
<td>172</td>
<td>B1</td>
<td>ROW12</td>
</tr>
<tr>
<td>173</td>
<td>B1</td>
<td>ROW13</td>
</tr>
<tr>
<td>174</td>
<td>B1</td>
<td>ROW14</td>
</tr>
<tr>
<td>175</td>
<td>B1</td>
<td>ROW15</td>
</tr>
<tr>
<td>176</td>
<td>B1</td>
<td>ROW16</td>
</tr>
<tr>
<td>177</td>
<td>ENDATA</td>
<td></td>
</tr>
</tbody>
</table>

Exhibit B3. (Continued)
Exhibit B4. FORTRAN output/revised MPSX/370 input for reformulated transportation problem. This file is on disk under L.U9229.REV370.
APPENDIX C

Section Cl
A third version of the program that solves the Sequential Linear Goal Programming (SLGP) problem was created using the Extended Control Language (ECL) version of IBM's Mathematical Programming System Extended 370 (MPSX/370). Based on the high-level programming language, PL/I, ECL allows greater flexibility in data input and manipulation than MPSX or MPS, the control language specific to MPSX/370. Since ECL includes the facilities of the PL/I language as a part of MPSX/370, a PL/I compiler is needed to compile ECL programs. Appendix C, Section C3 contains a listing of the MPSX/370 ECL SLGP program, an example of an input data set which resides on disk, and various tables which will help explain how the program solves the SLGP problem. The characteristics of the SLGP problem and of the program that can solve it were discussed in Chapter II and Appendix A, Section A1, respectively.

The data file for the input data is set up in roughly the same manner as the MPSX SLGP program's input data in Appendix A. The data should have a data set NAME line, sections for ROWS, COLUMNS, and RHS and an ENDATA line, with all information within each section in appropriate file columns as described in the MPSX/370 user's guide (1976b). In general, all names within the sections must be eight alphameric characters or less, and special characters may be used except the '$' which may not be used as the first character in a name. In the ROWS section the objective function names must be placed first, and they must be of the form OBJn where n is an integer greater than zero but less than one hundred. The order in which they are placed does not matter. More on the order of the objective function names is
explained in Section C2. Naturally, no two objective function names should be exactly the same. The data for all of the objective functions needs to be included in the COLUMNS sections. An example of an input data set is Example A in Appendix A, Exhibit A3, where the problem has three objective functions, four rows and ten columns. All objective function names, OBJ1, OBJ2, OBJ3, are listed first and their data is recorded in the COLUMNS section. Unlike the input for the MPSX or MPSX/370 SLGP program, the user need not name the data 'SLGPDAT' nor name the right hand side 'Bl'. An ENDATA line completes the file.

A listing of this program that solves the SLGP problem can be found in Exhibit C1, while a line by line detailed program description can be found in Appendix C, Section C2. This particular listing in Exhibit C1 is for the program that was used when solving Example A. In this case, as will be explained in detail later, OBJ3 is associated with priority level one, OBJ1 is associated with priority level 2, and OBJ2 with priority level three. The beginning of the program is spent defining the main program, initializing communication region cells, and defining variables and structures. The ASSIGN statement, line 9, indicates that the input data found where indicated on the INPUT DD card, line 63, will be referred to by the program as ALLDAT. Next, the first objective function name, FOBJ, the right hand side name, RHSNAME, and the data's name, DNAME, are declared as character variables. Some indicator variables as well as an array, ORDER, of size ten are then declared to be integer valued. The structures REVDATA and OUTDATA are
defined. REVDATA and OUTDATA identify arrays which possess logical relationships with one another. OUTDATA is the structure that will contain the output from the SOLUTION procedure after each iteration through the SLGP problem. The array SNAME will contain the row names and the array ACTIVITY will contain the row activity levels. The structure OUTDATA is dimensioned to size 10 because only the information in the objective function rows from the ROWS section of SOLUTION's output is transferred to OUTDATA, and the problem currently handles a maximum of ten objective functions. The output obtained from SOLUTION after solving Example A's first priority level is listed in Exhibit C2. Exhibit C3 shows the subsequent values stored in the structure OUTDATA, which are the values found in SECTION 1 - ROWS in the OBJ1, OBJ2, and OBJ3 rows under the column headings ...ROW... and ...ACTIVITY... (note that a '.' in MPSX is an exact '0'). Since there are three objective functions there are three variables filled for each array in OUTDATA, and since these will be the only variables referred to by the program at this iteration it was unnecessary to initialize variables 4 through 10 in any of the arrays. REVDATA is the structure which will contain the revisions which are used to modify the current problem before solving the linear programming problem associated with the next priority level. The form of REVDATA is similar to that of input data but the arrays IND, NAME1, NAME2, and VALUE do the job that input fields previously accomplished. IND contains the indicator and control card information normally found in field one truncated to the first two characters. Thus,
"RO" is used in place of "ROWS", "MO" in place of "MODIFY", and "RH" in place of "RHS". Arrays NAME1 and NAME2 will contain the information normally placed in field 2 and field 3, while the numeric information normally found in field 4 is placed in the array VALUE. In the ROWS section of REVDATA the current objective function is changed to an equality, allowing it to enter the problem as a new row in the problem. The current optimum is assigned as the new row's right hand side in REVDATA's RHS section. An example of REVDATA after solving priority one for Example A can be found in Exhibit C4. OBJ3, the objective function used in priority one, is modified to an equality with a right hand side value of zero, the optimum for priority level one. B1 is Example A's right hand side name. REVDATA will use all six values in each array at each iteration through the SLGP problem. As mentioned earlier, this program currently handles a maximum of ten objective functions (i.e., ten priority levels). To allow for more than ten priority levels the user would need to adjust the dimension of ORDER in Exhibit C1 line 14 and the dimension of OUTDATA in Exhibit C1 line 20 to an appropriate desired value.

Next, many of the variables and some of the communication region cells are initialized. The two GET LIST statements read in information the user will need to supply at the end of the program (line 65 and 66). The first GET LIST reads in the values of the number of priority levels, the name of the objective function to be solved in priority level one, the right hand side name and the data set's name. The next line read
in by the second GET LIST statement should contain the integer values which determine the order in which the objective functions listed in the ROWS section of the input data are to be solved by the SLGP program. A more detailed description of these variables and the form they should have when being read in can be found in Appendix C Section C2. After reading in user supplied information, the MPSX/370 ECL program initializes various variables in arrays IND, NAME1, NAME2 and the variable PLEVEL. The name of the input data, the problem's name on the PROBFILE, the right hand side name and the first objective function name are then assigned to the appropriate communication region cells. The MPSX/370 ECL program is now ready to begin solving the SLGP problem.

After the data are converted to an internal representation on the PROBFILE and an initial solution is found, the program enters an iterative process of solving the current linear programming problem, checking to see if it is the last problem in the SLGP problem, and if not revising the data file before moving on to solve the next problem. The label CONT in line 42 defines the point that the program control returns to after each iteration. PRIMAL is the optimizing procedure used to obtain an optimal solution. The optimal basis is saved so that it can be used in the next priority level's linear programming problem to find an initial feasible solution. The results from the procedure PRIMAL are printed in the first call to SOLUTION and are partially stored in the structure OUTDATA in the second call to
SOLUTION. At this point a check is made to see if the linear programming problem for the last priority level has been solved, and if so the program exits at FINISH, line 60. Otherwise, some variables from the structure REVDATA are initialized to their appropriate values and the priority level is increased by one. Recall that a description of REVDATA was provided earlier and Exhibit C4 gives a representative example of REVDATA and the values of its variables. Appropriate communication region cells are initialized or updated to their necessary values and in particular the next objective function name is moved into CR cell XOBJ. REVISE is called to modify the data in preparation for solving the next priority level's problem. A call to SETUP initiates a feasible solution while a call to RESTORE reinstates and reinverts the last priority level's optimal basis, providing a near optimal feasible solution. Program control is then transferred back to the line labelled CONT and continues in this fashion until problems for all priority levels have been solved.

To successfully solve a SLGP problem the user must provide a minimal amount of information. The JCL for the program in Exhibit C1 (lines 1 through 6 and 62 through 64) is that which is compatible with Iowa State University's NAS9160 computer and with the author's computer account, therefore users may need to modify the JCL statements. At ISU the user needs to supply their own computer account number, bin number, and data set name for their input data stored on disk (lines 1, 2, and 63, respectively). Two lines of information pertinent to solving the
user's SLGP problem, lines 65 and 66, must be supplied after the GO.SYSIN statement. This information, read by the GET LIST statements, is explained in detail in Section C2.
Section C2
CONTROL: PROCEDURE OPTIONS (MAIN);
This statement specifies that the PL/1 procedure's control program is named CONTROL. The option MAIN specifies that this procedure should be given control when the program is executed.

%INCLUDE DPLINIT;
This statement appears once at the beginning of a program. DPLINIT is a system macro that initializes tolerances and communication region cells and defines the course of action when errors and special conditions occur.

CALL ASSIGN ('FILE', 'ALLDAT', 'JCL','INPUT');
This statement associates the control program filename ALLDAT with the JCL ddname for the DD statement INPUT. This DD statement (Exhibit C1 line 63) reveals the name the user has assigned to the input data stored on disk. ALLDAT is the name used by the control program to reference this data. Exhibit C1 line 63 shows that the input data for Example A is on disk with the name L.U9229.LDATA(ORIG408).

DCL FOBJ $CHAR;
This statement declares FOBJ to be a character variable of length eight. FOBJ will contain the name for the first objective function to be solved.

DCL RHSNAME $CHAR;
This statement declares RHSNAME to be a character variable of length eight. RHSNAME will contain the name for the problem's right hand side.
DCL DNAME $CHAR;

This statement declares DNAME to be a character variable of length eight. DNAME will contain the data's name, which is the same name found on the NAME line in the input data.

DCL LPREFIX,I,PREFIX $INTEGER;

This statement declares the following variables to be integer valued. LPREFIX contains the number of priority levels in the problem. I is used as an index when solving the SLGP problem. PREFIX is the variable that contains the current priority level. Its value changes with each iteration through each new linear programming problem.

DCL ORDER(10) $INTEGER;

This statement declares an array of size 10, ORDER, to be one of integer variables. This array contains the order in which the objective functions are to be solved, with ten being the maximum number of objective functions that this program can currently manage.

DCL REVDATA(6),
   2 IND CHAR(2),
   2 NAME1 $CHAR,
   2 NAME2 $CHAR,
   2 VALUE $REAL2;

This statement declares a structure, REVDATA, of four arrays of size 6 each. IND is an array of character variables of length two. NAME1 and NAME2 are arrays of character variables of length eight. VALUE is an
array of double precision variables. This structure will contain the revisions to the problem associated with the next priority level after the solution to each current problem is determined.

DCL 1 OUTDATA(IO)
   2 SNAME $CHAR,
   2 ACTIVITY $REAL2;

This statement declares a structure, OUTDATA, of two arrays of size 10 each. SNAME is an array of character variables of length eight. ACTIVITY is an array of double precision real variables. This structure will contain the ROWS section of SOLUTION's output after each iteration.

GET LIST(LPLEVEL,FOBJ,RHSNAME,DNAME);

This statement reads in the values of LPLEVEL, FOBJ, RHSNAME and DNAME in that order. This information must be defined and entered as the first data entry after GO.SYSIN DD *. Since FOBJ, RHSNAME and DNAME are character variables their values must be enclosed in quotation marks and all values must be separated by a comma or a space. Exhibit C1 line 65 gives the proper input for Example A. Recall from Appendix C Section C1 that Example A had three priority levels, thus LPLEVEL=3. Line 65 of Exhibit C1 indicates that OBJ3 will be the first objective function solved. The value of the right hand side name, RHSNAME, is B1, which is the name used in Example A. DNAME is the name found on the NAME card in Exhibit A3 Appendix A, SLGPDAT.
GET LIST(ORDER);

This statement reads in the values of the array ORDER. The integer values for ORDER associated with the current problem must be the second data entry after GO.SYSIN DD *. The integer values should indicate the order in which the objective functions listed in the ROWS section of the input data are to be solved in the SLGP problem. Each value should be separated from adjacent values by a comma or a blank and the user should fill ORDER with zeros after the true values have been listed. Thus, ORDER(1) indicates which objective function in the list is solved in priority level one, ORDER(2) the function solved in priority level two, and so on. Exhibit Cl line 66 gives an example of this data line for Example A. Recall from Exhibit A3 Appendix A that the three objective functions are listed in the ROWS section of the input data as OBJ1, OBJ2, and OBJ3. The values 3, 1, 2 in line 66, Exhibit Cl indicate that OBJ3 will be solved at priority level one, OBJ1 will be solved at priority level two and OBJ2 will be solved at priority level three. Notice the remaining values for array ORDER are set to zero.

PLEVEL=1;

This statement initiates the value of PLEVEL to 1.

IND(1)='RO';  IND(2) = 'MO';
IND(3)='E ';  IND(4) = 'RH';
IND(5)='MO';  IND(6) = ' ';  

These statements initialize the values of the array IND from the structures REVDATA to their appropriate values. See Exhibit C4 for an example of the form of REVDATA.
DO I=1 to 2;
   NAME1(I)=''; NAME1(I+3)='';
END;

This DO loop initializes some variables in the array NAME1 from the structure REVDATA to blank values. See Exhibit C4.

NAME1(6)=RHSNAME;

This statement initializes the last variable in the array NAME1 to the value found in the variable RHSNAME.

DO I=1 to 5;
   NAME2(I)='';
END;

This DO loop initializes all but the last variable in the array NAME2 from the structure REVDATA to blank values. See Exhibit C4.

XDATA=DNAME;

This statement moves the input data set name contained in the variable DNAME into the communication region cell XDATA.

XPBNAME='SLGPDAT';

This statement moves the problem name, SLGPDAT, into the communication region cell, XPBNAME. SLGPDAT will be the name of the problem on PROBFILE.
XRHS=RHSNAME;
This statement moves the right hand side name found in the variable RHSNAME into the communication region cell XRHS.

XOBJ=FOBJ;
This statement moves the objective function name found in the variable FOBJ into the communication region cell XOBJ.

CALL CONVERT('FILE','ALLDAT');
CONVERT instructs MPSX/370 ECL to read the input data found in the file assigned to ALLDAT and convert it to packed binary form on the PROBFILE. This internal representation of the problem is written on PROBFILE with the name SLGPDAT, the name found in XPBNAME.

CALL SETUP('MIN');
SETUP initiates a solution to the problem SLGPDAT, minimizing the objective function.

CONT; CALL BCDOUT;
BCDOUT is used to get a listing of the complete problem at its present state.

CALL PRIMAL;
PRIMAL is one of MPSX/370 ECL's optimizing procedures. It first finds a feasible solution, then an optimal solution.
CALL SAVE;
SAVE stores the current optimal basis, bounded variable status, and part of the communication region on PROBFILE.

CALL SOLUTION;
In this case, SOLUTION prints the current MPSX/370 ECL solution in a tabled format.

CALL SOLUTION('STRUCTURE',OUTDATA,'NAME','VALUE','RMASKS','OBJ**','CMASKS');
In this case, SOLUTION stores the specified portions of the current solution in the structure OUTDATA. 'CMASKS', ' ', instructs SOLUTION to bypass the results given in the COLUMNS section (CSECTION array). 'NAME','VALUE','RMASKS','OBJ**',' ' instructs SOLUTION to include in OUTDATA only the row names and activity levels from the objective function rows given in the ROWS section (RSECTION array) of the SOLUTION output. To understand the correspondence between the structure and the solution results, see Exhibits C2 and C3.

IF PLEVEL = LPLEVEL THEN GOTO FINISH;
This statement checks to see if the current priority level is the same as the last priority level. If so, the SLGP problem has been solved and the program exits at FINISH.
NAME1(3)=SNAME(ORDER(PLEVEL));
NAME2(6)=SNAME(ORDER(PLEVEL));

These statements initialize a variable in arrays NAME1 and NAME2 to the value found in the variable SNAME(ORDER(PLEVEL)).

VALUE(6)=ACTIVITY(ORDER(PLEVEL));

This statement initializes the last variable in the array VALUE from the structure REVDATA to the value found in the variable ACTIVITY(ORDER(PLEVEL)).

PLEVEL=PLEVEL+1;

This statement increments PLEVEL by one.

XOBJ=SNAME(ORDER(PLEVEL));

This statement moves the objective function name found in the variable SNAME(ORDER(PLEVEL)) into the communication region cell XOBJ.

XOLDNAME='SLGPDAT';

This statement moves the old name of the problem on PROBFILE, SLGPDAT, into the communication region cell XOLDNAME.

CALL REVISE('STRUCTURE',REVDATA);

REVISE modifies the problem SLGPDAT on the PROBFILE according to the revisions recorded in the structure REVDATA. The problem is stored back on PROBFILE under the name SLGPDAT.
CALL RESTORE;
RESTORE brings back and reinverts the last optimal basis from PROBFILE.

GOTO CONT;
This statement transfers the program control to the line labelled CONT.

FINISH: STOP;
This statement signals the program's end. FINISH is the line's label.

END CONTROL;
This statement signals the end of the main program named CONTROL.
Section C3
1. //LEEANN JOB U9274,LAC
2. J0BPARM BIN=515
3. /// PROGRAM NAME LEE#GOALECL
4. /// EXEC MPSX370E,REGION.PLI=320K,REGION.GO=384K,
5. /// TIME.GO=1
6. ///PLI.SYSIN DD *
7. CONTROL: PROCEDURE OPTIONS (MAIN);
8. %INCLUDE DPLINIT;
9. CALL ASSIGN('FILE','ALLDAT','JCL','INPUT');
10. DCL FOBJ $CHAR;
11. DCL RHSNAME $CHAR;
12. DCL DNAME $CHAR;
13. DCL LPLEVEL,I,PLEVEL $INTEGER;
14. DCL ORDER(10) $INTEGER;
15. DCL 1 REVDATA(6),
16. 2 IND CHAR(2),
17. 2 NAME1 $CHAR,
18. 2 NAME2 $CHAR,
19. 2 VALUE $CHAR2;
20. DCL 1 OUTDATA(IO),
21. 2 SNAME $CHAR,
22. 2 ACTIVITY $REAL2;
23. GET LIST(LPLEVEL,FOBJ,RHSNAME,DNAME);
24. GET LIST(ORDER);
25. PLEVEL=1;
26. IND(1)='R0'; IND(2)='MO';
27. IND(3)='E '; IND(4)='RH';
28. IND(5)='MO'; IND(6)=' ';
29. DO I=1 TO 2;
30. NAME1(I)=''; NAME1(I+3)=';
31. END;
32. NAME1(6)=RHSNAME;
33. DO I=1 TO 5;
34. NAME2(I)='';
35. END;
36. XDATA=DNAME;
37. XPBNAME='SLGPDAT';
38. XRHS=RHSNAME;
39. XOBJ=FOBJ;
40. CALL CONVERT('FILE','ALLDAT');
41. CALL SETUP('MIN');
42. CONT: CALL BCDOUT;
43. CALL PRIMAL;
44. CALL SAVE;
45. CALL SOLUTION
46. CALL SOLUTION('STRUCTURE',OUTDATA,'NAME','VALUE',
47. 'RMASKS','OBJ**',',''CMASKS','');

Exhibit Cl. SLGP program using MPSX/370 ECL
IF PLEVEL=LPLEVEL THEN GOTO FINISH;

NAME1(3)=SNAME(ORDER(PLEVEL));
NAME2(6)=SNAME(ORDER(PLEVEL));
VALUE(6)=ACTIVITY(ORDER(PLEVEL));
PLEVEL=PLEVEL+1;
XOBJ=SNAME(ORDER(PLEVEL));
XOLDNAME='SLGPDAT';
XPBNAME='SLGPDAT';
CALL REVISE('STRUCTURE',REVDATA);
CALL SETUP('MIN');
CALL RESTORE;
GOTO CONT;

FINISH: STOP;
END CONTROL;

/*

//GO.INPUT DD DSN=L.U9229.LDATA(ORIG408),UNIT=DISK,DISP=SHR
//GO.SYSIN DD *
3 'OBJ3' 'B1' 'SLGPDAT'
3 1 2 0 0 0 0 0 0
*/

Exhibit Cl. (Continued)
SOLUTION (OPTIMAL)

<table>
<thead>
<tr>
<th>...NAME...</th>
<th>...ACTIVITY...</th>
<th>DEFINED AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTIONAL</td>
<td></td>
<td>OBJ3</td>
</tr>
<tr>
<td>RESTRAINTS</td>
<td></td>
<td>B1</td>
</tr>
</tbody>
</table>

SECTION 1 - ROWS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>...ROW...</th>
<th>AT ...ACTIVITY...</th>
<th>SLACK ACTIVITY</th>
<th>DUAL ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBJ1</td>
<td>BS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>OBJ2</td>
<td>BS 3.00000</td>
<td>3.00000-</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>OBJ3</td>
<td>BS</td>
<td></td>
<td>1.00000</td>
</tr>
<tr>
<td>A 4</td>
<td>ROW1</td>
<td>EQ 12.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 5</td>
<td>ROW2</td>
<td>EQ 10.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 6</td>
<td>ROW3</td>
<td>EQ 7.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 7</td>
<td>ROW4</td>
<td>EQ 4.00000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SECTION 2 - COLUMNS

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>...COLUMNS</th>
<th>AT ...ACTIVITY...</th>
<th>...INPUT COST...</th>
<th>...REDUCED COST.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>C1</td>
<td>BS 4.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 9</td>
<td>C2</td>
<td>LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>C3</td>
<td>BS 4.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>C4</td>
<td>BS 6.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C5</td>
<td>BS 3.00000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 13</td>
<td>C6</td>
<td>LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 14</td>
<td>C7</td>
<td>LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 15</td>
<td>C8</td>
<td>LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 16</td>
<td>C9</td>
<td>LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>C10</td>
<td>LL 1.00000</td>
<td>1.00000</td>
<td></td>
</tr>
</tbody>
</table>

*LOWER LIMIT and UPPER LIMIT columns were omitted.

Exhibit C2. Solution output*, example A, priority level one
### Exhibit C3. Structure OUTDATA, example A, priority level one

<table>
<thead>
<tr>
<th>VARIABLE IN ARRAY</th>
<th>SNAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBJ1</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>OBJ2</td>
<td>3.0</td>
</tr>
<tr>
<td>3</td>
<td>OBJ3</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Exhibit C4. Structure REVDATA, example A, priority level one

<table>
<thead>
<tr>
<th>VARIABLE IN ARRAY</th>
<th>IND</th>
<th>NAME1</th>
<th>NAME2</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>OBJ3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>B1</td>
<td>OBJ3</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>
The Extended Control Language (ECL) version of IBM's Mathematical Programming System Extended 370 (MPSX/370) was used to solve the sequential dual linear goal programming problem. ECL is based on PL/I, a high-level programming language, and as a result it has greater capabilities than the control language specific to MPSX/370, MPS Control Language. ECL allows the user more flexibility than MPS or MPSX in retrieving problem data and results, so one can use this information to alter the course of future computations. ECL includes the facilities of the PL/I language as a part of MPSX/370, therefore, a PL/I compiler is needed to compile ECL programs. Appendix D, Section D3 contains a listing of the MPSX/370 ECL control program, an example of an input data file which resides on disk, and various tables which will be used to explain how the program solves the sequential dual LGP problem.

Recall, as described in Chapter III, that the multidimensional dual problem consists of a linear programming problem with multiple, prioritized right hand sides. Each right hand side corresponds to an objective function in the primal linear goal programming problem, one for each priority level. Any program that intends to solve the dual LGP problem sequentially will need a device to keep track of which right hand side is being used in the problem at any particular iteration. After solving the linear programming problem associated with a right hand side, certain steps must be taken before solving the linear programming problem associated with the next right hand side. As explained in Chapter III, rows with slacks in them are redundant and
thus, are no longer needed when solving the next problem. These rows with slacks must be identified and removed from the problem's work matrix. Then, to acknowledge the priority of the right hand side in the problem just solved over the priority of the right hand side for the next problem, a new variable is created. This is done by adding the most recent right hand side as a new column to the work matrix with the most recent objective function optimum as this column's coefficient in the objective function. Lastly, the program must be able to identify what is to be the linear programming problem's new right hand side vector.

The data file is set up in a manner that simplifies some of the modifications that will take place after each iteration through the dual LGP problem. The data file should be in traditional MPSX form with a data set NAME, sections for ROWS, COLUMNS, RHS and BOUNDS, and an ENDATA line. Within each section, the data should be placed in appropriate file columns as described in any MPSX/370 user's guide (1979a). In the ROWS section the objective function name must be listed first, followed by the row names. Later, when the program iterates through the results of a solution to find the rows with slacks it assumes the objective function has been listed in the ROWS section first. Names must be eight alphameric characters or less. Special characters may also be used except the dollar sign '$' may not be used as the first character in a name. All right hand sides to be used in the dual LGP problem must be recorded in the RHS section. The
right hand side names must be seven alphabetic characters or less (special characters allowed as before), and naturally no two right hand side names can be exactly the same. All data recorded in the RHS section, except that for the last right hand side, must be duplicated in the COLUMNS section using as the column name the right hand side name with the letter 'C' in front of it. These extra columns must then be declared in the BOUNDS section as variables fixed to zero. This setup is required so that a new variable can be created after each iteration through the dual LGP problem with minimal effort. Thus, all right hand sides that will eventually become columns (which is all but the last) are declared as both right hand sides and columns in the input data, the only difference being the letter 'C' in front of the column names. As the program iterates through the dual LGP problem, the bounds on these extra variables are removed and the columns are added to the problem's work matrix.

An example of an input data set is Example D listed in Exhibit D2. This is the multidimensional dual of Example A. There are three priority levels, thus, three right hand sides in this problem, as well as six rows and four columns. The first line contains the name of the problem, DUALDAT. Next, is the ROWS section where the objective function name, OBJ1, is listed first followed by the row names. The COLUMNS section contains the usual information for the problem's four columns C1, C2, C3 and C4 as well as duplicate information for the first two right hand sides, B1 and B2, under the column names CBl and
CB2. Data for all three right hand sides B1, B2, and B3 is found in the
RHS section. Vector Z1 fixes CB1 and CB2 to zero in the BOUNDS section.
An ENDATA line completes this file.

A listing of the program that will solve the dual LGP problem can
be found in Exhibit D1. Specifically, this listing represents the
program used when solving Example D. The beginning of the program, line
7 to line 31, consists of instructions that define the main program,
initialize communication region cells, define variables and structures
and initialize or read in values for some of the variables. The user
will need to specify some information at the end of the program (lines
93 and 94) for each problem solved, but it is here that MPSX/370 ECL
acknowledges this information. The ASSIGN statement, line 9, indicates
that the data will be referred to as ALLDAT in the program, and this
data can be found where the user has indicated on the INPUT DD card,
line 91. Next, the bound vector name, the objective function name, the
data's name and the vector of right hand side names, BDNAME, OBJNAME,
DNAME, and BRHS respectively, are declared as character variables. Some
indicator variables are then declared as integer valued. Next, the
two GET LIST statements read in values of these variables. The first
line read in contains the values of the number of priority levels, the
number of rows, the objective function name, the bound vector name and
the data set's name. The next line read in contains the ten right hand
side names. For a more detailed description of these variables and the
form they should have when being read in see Appendix D, Section D2. In
general, Section D2 contains a detailed line by line program description. Next, the structures OUTDATD and REVDATD are defined. These structures identify arrays that possess logical relationships with one another.

OUTDATD is the structure that will contain the output from the SOLUTION procedure after each iteration. Row names will be stored in the array SNAME. Activity levels for the rows will be stored in the array ACTIVITY and slack values for the rows will be stored in the array SLACK. Exhibit D3 illustrates the output obtained from SOLUTION after solving Example D with right hand side B1. Exhibit D4 shows the subsequent values that are stored in structure OUTDATD, which are the values found in SECTION 1 - ROWS under the column heading ...ROW..., ...ACTIVITY..., and SLACK ACTIVITY (please note that in MPSX/370 ECL a '.' represents an exact '0'). There are seven variables filled for each array, one for each row. Notice that ROW7 and ROW8 are the only rows with any slack in them. The program will refer only to these seven variables in each array, thus, it was not necessary to initialize variables 8 through 1000 in any of the arrays. REVDATD is the structure that contains the information used to modify the current problem before the linear programming problem for the next right hand side is solved. The general form of REVDATD, illustrated in Exhibit D5, is similar to that of the input data but with arrays doing the job that input fields, previously accomplished. IND contains the indicator and control card information normally found in field one, truncated to the first two characters. Thus, "ROWS" becomes "RO", "MODIFY" becomes "MO", "COLUMNS" becomes "CO"
and "BOUNDS" becomes "BO". Arrays NAME1 and NAME2 will contain the information normally placed in field 2 and field 3 while the array VALUE contains the numeric information normally placed in field 4. The number of variables initialized in each array at any particular iteration through the dual LGP problem depends on the value of COUNT, which equals the number of rows with slack in them, but the form of REVDATD is the same for every iteration. First, rows with slack in them are modified in the ROWS section so they will not be in the next right hand side's linear programming problem (see variables 1 through COUNT+2). Next, in the COLUMNS section, the current optimum is assigned as the coefficient of the objective function for the right hand side entering the problem as a new column (see variables COUNT+3 through COUNT+5). In the BOUNDS section, then, this right hand side is entered as a new column by freeing the bound originally placed on it in the input data. Recall the new column name is the same as the current right hand side name with the letter 'C' attached to the front of it. Lastly, 'EA' is the value of IND in the last variable initialized. This signifies the end of action for this structure and any information in the remaining variables for each array of the structure is ignored. The structure REVDATD for Example D after having solved the linear programming problem using the first right hand side is listed in Exhibit D6. Recall from Exhibit D4 that rows ROW7 and ROW8 had slack in them, therefore, the value of COUNT is two and REVDATD will have eleven variables initialized for each array at this particular iteration. ROW7 and ROW8 will be
modified in the ROWS section so they are no longer in the problem. From Exhibit D2 we learned that the first right hand side was named B1, thus, column CBl has the objective function value 0.0 assigned as its coefficient in objective function OBJ1. Next, the bound is freed from column CBl, allowing it to enter the problem as a new variable. 'EA' then ends the action for this revise structure. Currently, the program is dimensioned for 1000 rows, including the objective function. The user could modify this so the program would accommodate, for example, 1200 rows by changing the 1000 to 1200 in lines 14, 18, and 22. To increase the number of rows above 1200, one would need to increase the REGION values in their JCL, line 4.

In much of the remainder of the code, Exhibit D1, lines 32 to 88, the program follows an iterative process of solving the linear programming problem for a given right hand side, checking to see if this is the last right hand side or if all rows have slack, and if not modifying the data file before the next linear programming problem is solved. Before the problem is solved with the first right hand side, the appropriate communication region cells need to contain the data name, the objective function name and the right hand side name. The data is then converted to an internal representation on the PROBFILE before an initial solution is found in the procedure SETUP, line 37. The label CONT is attached to the next line, line 38, to define the point that the program control returns to after each iteration. If the number of variables in the problem is less than the number of rows, the optimizing procedure DUAL is used to first
find a feasible solution and possibly an optimal solution before PRIMAL is called to assure an optimal solution. This allows MPSX/370 ECL to make use of the most efficient optimizing procedure initially, while it is working toward a feasible solution and may not yet be near the optimal solution. The optimal basis is stored and will be used in the next priority level's problem to help find an initial feasible solution. Two calls to SOLUTION cause the results to be printed and also the ROWS section of the results to be stored in the structure OUTDATD, as described earlier. The program then checks to see if the problem just solved used the last right hand side and if so it exists at FINISH, line 88. Otherwise, it iterates through the row slack values stored by SOLUTION in the array SLACK to determine how many rows have a value different from zero. Recall that this value, stored in the variable COUNT, determines the number of variables that will be initialized in this iteration in the structure REVDATD. After the value of COUNT is found, the appropriate variables in REVDATD are assigned the necessary values. The values that the variables in the arrays IND, NAME1, NAME2, and VALUE will take was explained earlier in the description of the structure REVDATD. The data are now ready to be revised, so the necessary communication region cells must be updated to their appropriate values. In particular, the right hand side name is changed to its next value before REVISE and SETUP are called to alter the data and find an initial feasible solution for the linear programming problem with the new right hand side. The previous solution's basis is reinstated
to provide a hopefully near optimal feasible solution. At this point program control is transferred back to the line labelled CONT and continues in this fashion until all right hand sides have been used in the dual LGP problem or all rows have been removed because of slack.

Minimal input is required from the user to successfully solve a dual LGP problem with this program. The JCL for the program in Exhibit D1 (lines 1 through 6 and 90 through 92) is that which is compatible with Iowa State University's NAS9160 computer and the author's computer account. For other users, modifications may be necessary in any JCL statements but specifically, at ISU, the user needs to supply their own computer account number, bin number, and data set name for their data properly stored on disk (lines 1, 2 and 91 respectively). Some information pertinent to the solving of the user's dual LGP problem must be supplied after the GO.SYSIN statement, lines 93 and 94. This is the information read by the GET LIST statements and is explained further in Section D2.
Section D2
CONTROL: PROCEDURE OPTIONS (MAIN);
This statement specifies that the PL/1 procedure's control program is named CONTROL. The option MAIN specifies that this procedure should be given control when the program is executed.

%INCLUDE DPLINIT;
This statement appears once at the beginning of a program. DPLINIT is a system macro that initializes tolerances and communication region cells and defines the course of action when errors and special conditions occur.

CALL ASSIGN('FILE','ALLDAT','JCL','INPUT');
This statement associates the control program filename ALLDAT with the JCL ddname for the DD statement INPUT. This DD statement reveals the name the user has assigned to the input data stored on disk. ALLDAT is the name used by the control program to reference this data. Exhibit D1 line 91 shows that the input data for Example D is on disk with the name L U9229.ALLDATD.

DCL BDNAME $CHAR;
This statement declares BDNAME to be a character variable of length eight. BDNAME will contain the name for the vector of bounds on the variables.
DCL OBJNAME $CHAR;
This statement declares OBJNAME to be a character variable of length eight. OBJNAME must contain the name for the problem's objective function.

DCL DNAME $CHAR;
This statement declares DNAME to be a character variable of length eight. DNAME must contain the data's name, which is the same name found on the NAME card in the input data.

DCL BRHS(10) CHAR(7);
This statement declares an array of size 10, BRHS, to be character variables of length seven. This array contains the name of the right hand sides. Currently, this program can solve dual goal programming problems with a maximum of ten right hand sides.

DCL FLAG(1000) $INTEGER;
This statement declares an array of size 1000, FLAG, to be integer valued. This array will contain a 0 if a particular row is still being used in each problem and a 1 if a particular row has been removed because it has slack in it.

DCL COUNT, BLEVEL, LLEVEL, NROWS, I, J $INTEGER;
This statement declares the following variables to be integer valued.
COUNT is the variable used by the program to store the number of slack rows found in the solution at each iteration. Its value changes after the problem is solved for each new right hand side.

BLEVEL is the variable that contains the priority level associated with the current right hand side. Its value changes with each iteration through each new right hand side.

LBLEVEL contains the number of right hand sides in the problem, which is the same as the number of priority levels in the corresponding primal linear goal programming problem.

NROWS contains the number of rows in the problem, including the objective function.

I and J are variables used as indices when solving the dual goal programming problem.

GET LIST (LBLEVEL,NROWS,OBJNAME,BDNAME,DNAME);

This statement reads in the values of LBLEVEL, NROWS, OBJNAME, BDNAME, and DNAME in that order. This information must be defined and entered as the first data entry after GO.SYSIN DD *. Since OBJNAME, BDNAME, and DNAME are character variables their values must be enclosed in quotation marks and all values must be separated by a comma or a space. Exhibit D1, line 93 gives the proper input line for Example D. Recall from Section D1 that Example D had three right hand sides and six rows, therefore, LBLEVEL=3 and NROWS=7. As indicated in Exhibit D2, the objective function
name, OBJNAME, is OBJ1, the bound vector's name, BDNAME, is Z1 and the
data's name, DNAME, is DUALDAT.

GET LIST(BRHS);
This statement reads in the values of the array BRHS. The names of the
right hand sides must be the second data entry line after the GO.SYSIN JCL
line, each enclosed in quotation marks and separated by a comma or a
blank. The names should be listed in the order in which the right hand
sides are to be solved in the problem. After the true right hand side
names have been listed, fill the remaining values of BRHS with blank
names, ' ', so that there are ten values being read into BRHS. Exhibit
D1, line 94 gives an example of this line for Example D. Recall from
Exhibit D2 that the three right hand sides are named B1, B2, and B3.
Line 94 indicates that the dual LGP problem is to be solved with the
right hand sides in that order. After the right hand side names have
been listed seven blank "names" are listed to finish the array BRHS.

DCL 1 OUTDATD(IOOO),
    2 SNAME $CHAR,
    2 ACTIVITY $REAL2,
    2 SLACK $REAL2;
This statement declares a structure, OUTDATD, of three arrays of size
1000 each. SNAME is an array of character variables of length eight.
ACTIVITY and SLACK are arrays of double precision real variables. This
structure will contain the rows section of SOLUTION's output after each
iteration.
DCL 1 REVDATD(1000),
   2 IND CHAR(2),
   2 NAME1 $CHAR,
   2 NAME2 $CHAR,
   2 VALUE $REAL2;

This statement declares a structure, REVDATD, of four arrays of size 1000 each. IND is an array of character variables of length two. NAME1 and NAME2 are arrays of character variables of length eight. VALUE is an array of double precision variables. This structure will contain the revisions to the problem associated with the next priority level after the solution to each current problem is determined.

DO  I=1 TO NROWS-1;
   FLAG(I)=0;
END;

This DO loop iterates through the array FLAG, initializing as many variables to zero as there are rows in the problem.

IND(1)='RO'; IND(2)='MO';

These statements initialize the first two values in the array, IND, to the values 'RO' and 'MO' respectively. See Exhibit D5 for the general form of REVDATD.

BLEVEL=1;

This statement initiates the value of BLEVEL to 1.
XDATA=DNAME;
This statement moves the input data set name contained in the variable DNAME into the communication region cell XDATA.

XPBNAME='DUALDAT';
This statement moves the problem name, DUALDAT, into the communication region cell, XPBNAME. DUALDAT will be the name of the problem on PROBFILE.

XOBJ=OBJNAME;
This statement moves the objective function name found in the variable OBJNAME into the communication region cell, XOBJ.

XRHS=BRHS(BLEVEL);
This statement moves the right hand side name found in the variable BRHS(BLEVEL) into the communication region cell XRHS.

CALL CONVERT('FILE','ALLDAT');
CONVERT instructs MPSX/370 ECL to read the input data found in the file assigned to ALLDAT and convert it to packed binary form on the PROBFILE. This internal representation of the problem is written on PROBFILE with the name DUALDAT; the name found in XPBNAME.

CALL SETUP('MAX','BOUNDS',BDNAME,'SKIP');
SETUP initiates a solution to the problem DUALDAT, maximizing the
objective function. The BOUNDS parameter indicates that the vector whose name is stored in the variable BDNAME defines the bounds on some variables in the input data. SKIP instructs SETUP to bypass the variables that are fixed to zero in the BOUNDS section, possibly allowing the optimization procedure to be shortened.

CONT; CALL BCDOUT;

BCDOUT is used to get a listing of the complete problem at its present stage.

IF XJ<XM THEN CALL DUAL;

This statement tells MPSX/370 ECL to use the optimization procedure DUAL if XJ, the number of logical and structural variables in the problem, is less than XM, the number of rows in the problem. DUAL works toward a feasible solution but may not reach an optimal solution.

CALL PRIMAL;

PRIMAL is one of MPSX/370 ECL's optimizing procedures. It first finds a feasible solution, then an optimal solution.

CALL SAVE;

SAVE stores the current optimal basis, bounded variable status, and part of the communication region on PROBFILE.

CALL SOLUTION;

In this case, SOLUTION prints the current solution in a tabled format.
CALL SOLUTION('STRUCTURE',OUTDATD,'NAME','VALUE','COST','CMASKS',');

In this case, SOLUTION stores the specified portions of the current solution in the structure OUTDATD. 'CMASKS', ' ', instructs SOLUTION to omit the COLUMNS section, (CSECTION), of the results. 'NAME', 'VALUE', and 'COST' instructs SOLUTION to include only the row names, activity levels and row slack levels from the ROWS section (RSECTION) in the structure.

To understand the correspondence between the structure and the solution results, see Exhibits D3 and D4.

IF BLEVEL=LBLEVEL THEN GOTO FINISH;

This statement checks to see if the current priority level, (right hand side), is the same as the last priority level (right hand side). If so the dual goal programming problem has been solved and the program exits at FINISH.

COUNT=0;

This statement initializes the variable COUNT to zero.

DO I=2 TO NROWS;
IF SLACK(I)~=0 THEN COUNT=COUNT+1;
END;

This DO loop iterates through the array of variables SLACK, containing row slack levels for a given problem, checking each one to see if a given row had slack levels different than zero at the optimum solution. If so, the variable COUNT is incremented by one and at the end of the
DO loop COUNT contains the number of rows associated with the current solution that have slack in them.

IND(COUNT+3)='CO'; IND(COUNT+4)='MO';
IND(COUNT+5)=''; IND(COUNT+6)='BO';
IND(COUNT+7)='MO'; IND(COUNT+8)='FR';

These statements initialize more values in the array, IND, to their proper values. The position of these particular values in the array depends on the value of COUNT. See Exhibit D5 for the general form of REVDATD.

DO I=1 TO 2;
   NAME1(I)=''; NAME2(I)='';
   NAME1(COUNT+I+2)=''; NAME2(COUNT+I+2)='';
   NAME1(COUNT+I+5)=''; NAME2(COUNT+I+5)='';
END;

The statements within the DO loop initialize values in the arrays NAME1 and NAME2 to their proper values. The position of four of these values in their array depends on the value of COUNT. See Exhibit D5.

NAME1(COUNT+8)=BDNAME;

This statement initializes another variable in the array NAME1 to the value found in the variable BDNAME.

NAME1(COUNT+5)=C\|BRHS(LEVEL);
NAME2(COUNT+8)=C\|BRHS(LEVEL);

These two statements initialize two more variables in the arrays NAME1 and NAME2. The letter 'C' is concatenated with the proper right hand side name to give the name of the column to be modified. See Exhibit D5.
NAME2(COUNT+5)=OBJNAME;
This statement initializes the appropriate variable in the array NAME2 to the name of the objective function found in the variable OBJNAME.

VALUE(COUNT+5)=ACTIVITY(1);
This statement initializes the appropriate variable in the array VALUE to the optimum value of the previous solution's objective function.

J=3;
This statement initializes the index variable J to three.

DO  I=2 TO NROWS;
   IF SLACK(I)<=0.00 THEN DO;
      IND(J)='N';
      NAME1(J)=SNAME(I);
      NAME2(J)='J=J+1;
      FLAG(I-1)=1;
   END;
END;

This DO loop iterates through the array of variables SLACK, containing row slack levels for a given problem, checking each one to see if a given row had slack levels different than zero at the optimum. Each time a row with slack in it is found, the name of that row is recorded in the appropriate variable in the array NAME1, appropriate values are assigned to variables in arrays IND and NAME2, and J is incremented by one. Also, the appropriate variable in the array FLAG is set to the value of 1.
IND(COUNT+9)='EA';
This statement initializes the last variable in the array IND for this iteration to the value EA, which signals the end of the action for the structure REVDATD.

NAME1(COUNT+9)=';
NAME2(COUNT+9)=';
These statements initialize the last variables in arrays NAME1 and NAME2 for this iteration to their appropriate values.

DO  I=1 TO NROWS-1;
   IF FLAG(I)=0 THEN GOTO NEXT;
END;
This DO loop iterates through the previously initialized variables in the array FLAG, checking to see if there are any rows left in the problem which did not have slack in them. If so, control is transferred to the line labelled NEXT which allows the program to continue. Otherwise, control is passed to the next line.

GOTO FINISH;
This line transfers program control to the line labelled FINISH.

NEXT: BLEVEL=BLEVEL+1;
This statement increments BLEVEL by one.
XOLDNAME='DUALDAT';
This statement moves the old name of the problem on PROBFILE, DUALDAT, into the communication region cell XOLDNAME.

CALL REVISE('STRUCTURE',REVDATD);
REVISE modifies the problem DUALDAT on the PROBFILE according to the revisions recorded in the structure REVDATD. The problem is stored back on PROBFILE under the name DUALDAT.

CALL RESTORE;
RESTORE brings back the last optimal basis from PROBFILE and reinverts this basis.

GOTO CONT;
This statement transfers the program control to the line labelled CONT.

FINISH: STOP;
This statement signals the program's end. FINISH is the line's label.

END CONTROL;
This statement signals the end of the main program named CONTROL.
Section D3
CONTROL: PROCEDURE OPTIONS (MAIN);

INCLUDE DPLINIT;

CALL ASSIGN('FILE', 'ALLDAT', 'JCL', 'INPUT');

DCL BDNAME $CHAR;
DCL OBJNAME $CHAR;
DCL DNAME $CHAR;
DCL BRHS(IO) CHAR(7);
DCL FLAG(IOO) $INTEGER;
DCL COUNT,BLEVEL,LBLLEVEL,NROWS,I,J $INTEGER;

GET LIST(LBLEVEL,NROWS,OBJNAME,BDNAME,DNAME);
GET LIST(BRHS);

DCL OUTDATD(IOO0),
  2 SNAME $CHAR,
  2 ACTIVITY $REAL2,
  2 SLACK $REAL2;

DCL REVDATD(IOO0),
  2 IND CHAR(2),
  2 NAME1 $CHAR,
  2 NAME2 $CHAR,
  2 VALUE $REAL2;

DO I=1 TO NROWS-1;
  FLAG(I)=0;
END;
IND(1)='R0'; IND(2)='M0';
BLEVEL=1;
XDATA=DNAME;
XPBNAME='DUALDAT';
XOBJ=OBJNAME;
XRHS=BRHS(BLEVEL);
CALL CONVERT('FILE', 'ALLDAT');
CALL SETUP('MAX', 'BOUNDS', BDNAME, 'SKIP');

CONT: CALL BCDOUT;
IF XJ<XM THEN CALL DUAL;
CALL PRIMAL;
CALL SAVE;
CALL SOLUTION;
CALL SOLUTION('STRUCTURE', OUTDATD, 'NAME', 'VALUE', 'COST',
  'CMASKS', '');
IF BLEVEL=LBLLEVEL THEN GOTO FINISH;
COUNT=0;

Exhibit D1. Sequential dual LGP program using MPSX/370 ECL
DO I=2 TO NROWS ;
    IF SLACK(I) ^= 0.000 THEN COUNT = COUNT + 1;
END;
IND(COUNT + 3) = 'CO'; IND(COUNT + 4) = 'MO';
IND(COUNT + 5) = ' '; IND(COUNT + 6) = 'BO';
IND(COUNT + 7) = 'MO'; IND(COUNT + 8) = 'FR';
DO I=1 TO 2;
    NAME1(I) = ' '; NAME2(I) = ' ';
    NAME1(COUNT + I + 2) = ' '; NAME2(COUNT + I + 2) = ' ';
    NAME1(COUNT + I + 5) = ' '; NAME2(COUNT + I + 5) = ' ';
END;
NAME1(COUNT + 8) = BDNAME;
NAME1(COUNT + 9) = ' ';
NAME2(COUNT + 8) = 'C1' | | BRHS(BLEVEL);
NAME2(COUNT + 9) = OBJNAME;
VALUE(COUNT + 5) = ACTIVITY(1);
J = 3;
DO I=2 TO NROWS ;
    IF SLACK(I) ^= 0.000 THEN DO;
        IND(J) = 'N';
        NAME1(J) = SNAME(I);
        NAME2(J) = ' ';
        J = J + 1;
        FLAG(I - 1) = 1;
    END;
END;
IND(COUNT + 9) = 'EA';
NAME1(COUNT + 9) = ' ';
NAME2(COUNT + 9) = ' ';
DO I=1 TO NROWS - 1;
    IF FLAG(I) = 0 THEN GOTO NEXT;
END;
GOTO FINISH;
NEXT: BLEVEL = BLEVEL + 1;
XOLDNAME = 'DUALDAT';
XPBNAME = 'DUALDAT';
XRHS = BRHS(BLEVEL);
CALL REVISE('STRUCTURE', REVDATD);
CALL SETUP('MAX', 'BOUNDS', BDNAME, 'SKIP');
CALL RESTORE;
GOTO CONT;
FINISH: STOP;
END CONTROL;
/*
//GO.INPUT DD DSN=L. U9229.ALLDATD,UNIT=DISK,DISP=SHR
//GO.SYSIN DD *
3 7 'OBJ1' 'Z1' 'DUALDAT'
'B1' 'B2' 'B3' , ' ' , ' ' , ' ' , ' ' , ' ' , ' '
//
Exhibit D1. (Continued)
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>NAME</td>
<td>DUALDAT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>ROWS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>N OBJ1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>L ROW1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>L ROW2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>L ROW7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>L ROW8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>L ROW9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>L ROW10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>COLUMNS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>C1 ROW1</td>
<td>-2.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>C1 ROW2</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>C1 ROW7</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>C1 OBJ1</td>
<td>-12.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>C2 ROW1</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>C2 ROW2</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>C2 ROW8</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.</td>
<td>C2 OBJ1</td>
<td>-10.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19.</td>
<td>C3 ROW1</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.</td>
<td>C3 ROW9</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21.</td>
<td>C3 OBJ1</td>
<td>-7.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.</td>
<td>C4 ROW1</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>C4 ROW2</td>
<td>-4.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24.</td>
<td>C4 ROW10</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.</td>
<td>C4 OBJ1</td>
<td>-4.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.</td>
<td>CB1 ROW1</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27.</td>
<td>CB1 ROW2</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28.</td>
<td>CB1 ROW7</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29.</td>
<td>CB1 ROW8</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30.</td>
<td>CB1 ROW9</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31.</td>
<td>CB1 ROW10</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32.</td>
<td>CB2 ROW1</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33.</td>
<td>CB2 ROW2</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34.</td>
<td>CB2 ROW7</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.</td>
<td>CB2 ROW8</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36.</td>
<td>CB2 ROW9</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37.</td>
<td>CB2 ROW10</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38.</td>
<td>RHS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39.</td>
<td>B1 ROW1</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40.</td>
<td>B1 ROW2</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41.</td>
<td>B1 ROW7</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42.</td>
<td>B1 ROW8</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43.</td>
<td>B1 ROW9</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44.</td>
<td>B1 ROW10</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45.</td>
<td>B2 ROW1</td>
<td>-1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46.</td>
<td>B2 ROW2</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47.</td>
<td>B2 ROW7</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exhibit D2. Input to MPSX/370 ECL for example D
<table>
<thead>
<tr>
<th>Number</th>
<th>Activity</th>
<th>Slack Activity</th>
<th>Dual Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OBJ1</td>
<td>.</td>
<td>1.00000</td>
</tr>
<tr>
<td>2</td>
<td>ROW1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>ROW2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>ROW7</td>
<td>.</td>
<td>1.00000</td>
</tr>
<tr>
<td>5</td>
<td>ROW8</td>
<td>.</td>
<td>1.00000</td>
</tr>
<tr>
<td>6</td>
<td>ROW9</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>ROW10</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

**SECTION 2 - COLUMNS**

<table>
<thead>
<tr>
<th>Number</th>
<th>Column</th>
<th>Activity</th>
<th>Input Cost</th>
<th>Reduced Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>C1</td>
<td>LL</td>
<td>12.00000-</td>
<td>12.00000-</td>
</tr>
<tr>
<td>9</td>
<td>C2</td>
<td>LL</td>
<td>10.00000-</td>
<td>10.00000-</td>
</tr>
<tr>
<td>10</td>
<td>C3</td>
<td>LL</td>
<td>7.000000-</td>
<td>7.000000-</td>
</tr>
<tr>
<td>11</td>
<td>C4</td>
<td>LL</td>
<td>4.000000-</td>
<td>4.000000-</td>
</tr>
</tbody>
</table>

*LOWER LIMIT and UPPER LIMIT columns are omitted*
Variable in array | Array
---|---
| SNAME | Activity | Slack
---|---|---
1 | OBJ1 | 0.0 | 0.0
2 | ROW1 | 0.0 | 0.0
3 | ROW2 | 0.0 | 0.0
4 | ROW7 | 0.0 | 1.0
5 | ROW8 | 0.0 | 1.0
6 | ROW9 | 0.0 | 0.0
7 | ROW10 | 0.0 | 0.0

Exhibit D4. Structure OUTDATD. Example D

Variable in array | Array
---|---
| IND | NAME1 | NAME2 | Value
---|---|---|---
1 | RO | | |
2 | MO | | |
3 | N | \(\text{ROW}(\text{slack}(1))\) |
| | \(\vdots\) | \(\vdots\) |
COUNT+2 | N | \(\text{ROW}(\text{slack}(\text{count}))\) |
COUNT+3 | CO | | |
COUNT+4 | MO | | |
COUNT+5 | | \(\text{C}(\text{r.h.s. } J)\) |
COUNT+6 | BO | \{obj.fn.\} | \{obj.fn. value\}
COUNT+7 | MO | | |
COUNT+8 | FR | \{bound vector\} | \(\text{C}(\text{r.h.s. } J)\)
COUNT+9 | EA | | |
COUNT+10 | | | |
| | \(\vdots\) | | |
1000 | | | |

r.h.s.=right hand side
obj.fn.=objective function

All terms in \{ \} are to be replaced by appropriate names, otherwise terms are to be replaced by appropriate values.

Exhibit D5. General form of structure REVDATD. Priority level J (>1)
<table>
<thead>
<tr>
<th>Variable in array</th>
<th>IND</th>
<th>NAME1</th>
<th>NAME2</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td></td>
<td>ROW7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td></td>
<td>ROW8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CB1</td>
<td>OBJ1</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>FR</td>
<td>Z1</td>
<td>CB1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>EA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exhibit D6. Structure REVDATD. Example D