MATHEMATICAL PROGRAMMING LANGUAGE

An Appraisal Based on Practical Experiments

BY

PIERRE E. BONZON

STAN-CS-72-267

MARCH 1972

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

MPL

MATHEMATICAL PROGRAMMING LANGUAGE

AN APPRAISAL BASED ON PRACTICAL EXPERIMENTS

by

Pierre E. Bonzon

Technical Report No. 72-6

February 1972

DEPARTMENT OF OPERATIONS RESEARCH
Stanford University
Stanford, California

This work was performed while the author was a visitor at Stanford University, under a Grant from the Swiss National Research Council.

Partial research and reproduction of this report was supported under the auspices of National Science Foundation Grant GJ 30408X

MATHEMATICAL PROGRAMMING LANGUAGE
AN APPRAISAL BASED ON PRACTICAL EXPERIMENTS

by

Pierre E. Bonzon

Abstract

The newly proposed Mathematical Programming Language is approached from the user's point of view. To demonstrate its facility of use, three programs are presented which solve large scale linear programming problems with the generalized upper-bounding structure.

Key words:

Mathematical Programming Language

Large-scale systems

Generalized upper-bounding

Introduction.

The purpose of the work reported here was to investigate the applicability and usefulness of the newly proposed Mathematical Programming Language, or MPL [1].

We were particularly interested in finding answers to such questions as:

- can MPL be regarded as a highly readable communication language for mathematical algorithms?
- are mathematical algorithms significantly easier to write and modify. in MPL than in any other currently available language?
- what performance, in solving large scale systems, can be expected from MPL programs?

To date, only a subset of MPL, called MPL/71, has been implemented, using PL/1 as a translator [7]. In particular, MPL/71 does not yet allow for partitioned data structures, thus preventing us from dealing efficiently with block angular structures. Therefore, our experimentation had to be limited in scope.

From the half a dozen programs that we have been writing and testing for a period of 3 months, we have selected three, that should enable us to bring convincing answers to our questions.

We shall first present the revised simplex algorithm in its simplest form, that is when an initial feasible basis is readily available through the slack variables. This program shows clearly how both pricing and column-generation can be accomplished simply by two matrix multiplications. This, we believe, is sufficient in itself to make up for our first point.

Two different versions of the generalized upper-bounding algorithm are then given. It will be thus demonstrated how easily specialized algorithms can be implemented, and in particular how few changes are required to modify an existing program (our second point).

These two programs having been run with the data of a large problem that was at hand, this will help us giving some elements of an answer for the third point also.

The Revised Simplex Algorithm [2].

Consider the following linear programming problem. Find

$$\min z = cx$$

such that $x \geq 0$

$Ax \leq b$, with $b > 0$

where $c = (c_j)$

$x = (x_j)$ $\quad i = 1, \ldots, m$

$A = ((a_{ij}))$ $\quad j = 1, \ldots, n$

$b = (b_i)$

An initial feasible basis, as well as its inverse, are readily available through the addition of slack variables.

Let (x_j), $j = n+1, \ldots, n+m$, be the slack variables, with coefficients (c_j) equal to zero and $((a_{ij}))$ forming an m by m identity matrix for $j = n+1, \ldots, n+m$.

At the beginning of some iteration, assume that the current basis inverse, B^{-1}, the associated basic solution, $x_B = B^{-1}b = \bar{b}$, and the data of the original problem, A, $\bar{c}$, are available.

One step of the revised simplex algorithm proceeds then as follows:

I Pricing: By definition, the simplex multipliers, or prices, are

$$\pi = (\pi_1, \ldots, \pi_m) = c_B B^{-1} .$$

The relative costs are then given by

$$\bar{c}_j = c_j - \sum_{i=1}^{m} \pi_i a_{ij}, \qquad j = 1, \ldots, n+m .$$

II Pivot column selection: The pivot column s is given by

$$c_s = \min_j \bar{c}_j .$$

If $\bar{c}_s \geq 0$, the current basic solution is optimal.

III Column generation: If $P_j = (a_{ij})$, i = 1, . . . , m, represents the jth column of the original matrix A, then the entering column is given by

$$\bar{P}_s = B^{-1} P_s = (\bar{a}_{is}), \qquad i = 1, \ldots, m ,$$

IV Pivot row selection: The pivot row r is given by

$$\frac{\bar{b}_r}{a_{rs}} = \min_{\bar{a}_{is} > 0} \frac{\bar{b}_i}{\bar{a}_{is}} = \theta \quad .$$

If all $\bar{a}_{is} < 0$, the optimum is unbounded.

V Update the inverse: The updated inverse is given by pivoting the first m columns of $[B^{-1} \vdots \bar{P}_s]$ on $\bar{a}_{rs}$.

VI Update the basic solution: The updated values of the basic variables are

$$\bar{b}_i \leftarrow \bar{b}_i - \theta \bar{a}_{is} , \qquad i = 1, \ldots , m, \; i \neq r$$

$$\bar{b}_r \leftarrow \theta \quad .$$

Now, it has been observed that by considering instead the following augmented matrices and vectors

$$A \leftarrow \left[\begin{array}{c|c} A & I \\ \hline -c_1 \cdots -c_n & 0 \end{array}\right] \qquad c \leftarrow [c \vdots 0]$$

$$B^{-1} \leftarrow \left|\begin{array}{c|c} B^{-1} & 0 \\ \hline \pi_1 \cdots \pi_m & 1 \end{array}\right| \qquad \pi \leftarrow [\pi \; 3 \; 1]$$

the pricing operation becomes equivalent to a matrix multiplication, i.e.,

$$c = -\pi A \quad .$$

Furthermore, if

$$P_j \longleftarrow \begin{bmatrix} P_j \\ --- \\ -c_j \end{bmatrix} \quad ,$$

then, as before

$$\bar{P}_j = B^{-1} P_j \qquad \text{and} \qquad \bar{c}_j = -\pi P_j \quad .$$

To complete the algorithm, the updated inverse will be given by pivoting the first m+1 columns of $[B^{-1} \vdots \bar{P}_s]$ on $\bar{a}_{rs}$, and the m+1th row of the updated inverse will contain the updated simplex multipliers for the next iteration (initially they are all zero, except $\pi_{m+1} = 1$).

The Program Revised-Simplex.

This MPL program reproduces exactly the 6 steps procedure described in the previous section, introducing the augmented matrices and the corresponding simplifications in the pricing operation.

The following notation is used, due to the non-availability of lower-case and special characters:

B for the right-hand side b and $\bar{b}$

INVERSE for the augmented inverse B^{-1}

COST for the augmented $\bar{c}$

PI for the prices π^T

AC for the augmented matrix A (to distinguish it from the original A).

Some of the interesting features of the MPL are well illustrated in this program. To be convinced of the advantage of **MPL** notations, as a communication tool, over other common languages, just compare the following lines of instructions:

I Declaration:

MPL: DEFINE INVERSE MATRIX **M+1** BY **M+1**

ALGOL-W: REAL ARRAY INVERSE (1 :: **M+1, 1** :: **M+1)**

PL/1: DECLARE INVERSE **(M+1,** M+1) BINARY FLOAT

II Set Notation:

MPL: FOR 1 IN (1, . . . , **M)** : P(I) > 0, **[···];**

ALGOL-W: FOR I : = 1 UNTIL M DO IF P(1) > 0 THEN BEGIN . **..END;**

. **PL/1:** DO I= 1 TO M WHILE (P(1) > 0);

...

END;

III Matrix Operations:

MPL: COST = -PI * AC;

ALGOL-W, **PL/1:** several **lines** of iterated statements.

```
PROGRAM REVISED_SIMPLEX;

"PROBLEM DESCRIPTION

FIND MIN Z=C*X  SUCH THAT
                X>=0
                A*X<=B, WHERE A MATRIX M BY N
IT IS ASSUMED B>=0"


"DATA"
GIVEN (M,N) INTEGER,
      A MATRIX M BY N,
      B COLUMN M,
      C ROW N;
$LET ZERO=1E-6;
$LET INFINITY=1E6;

"CONSTRUCT AUGMENTED MATRIX WITH SLACKS AND COST ROW"
DEFINE AC "AUGMENTED" MATRIX M+1 BY M+N,(I,J) "SUBSCRIPTS" INTEGER;
AC=0;
FOR I IN (1,...,M),|_FOR J IN (1,...,N),AC(I,J)=A(I,J)_|;
FOR I IN (1,...,M),AC(I,N+I)=1 "SLACKS";
FOR J IN (1,...,N),AC(M+1,J)=-C(J) "COSTS";

"INITIAL BASIS"
DEFINE BASIC INTEGER COLUMN M "INDEX FOR BASIC VARIABLES";
FOR I IN (1,...,M),BASIC(I)=N+I "INITIAL BASIC ARE SLACKS";

"INITIAL INVERSE"
DEFINE INVERSE MATRIX M+1 BY M+1;
INVERSE=0;FOR I IN (1,...,M+1),INVERSE(I,I)=1 "IDENTITY MATRIX";

"INITIAL VALUE FOR OBJECTIVE Z"
DEFINE Z=0.0;

"SIMPLEX ITERATION"

RECYCLE:"PRICING"
        DEFINE PI ROW M+1 "PRICES OR SIMPLEX MULTIPLIERS";
        PI=INVERSE(M+1,*);
        DEFINE COST ROW M+N "RELATIVE COSTS";
        COST=-PI*AC;

        "PIVOT COLUMN SELECTION"
        DEFINE S "PIVOT COLUMN" INTEGER,DELTA=INFINITY;
        FOR J IN (1,...,M+N),IF COST(J)<DELTA,|_S=J;DELTA=COST(J)_|;
        IF DELTA>-ZERO, GO TO BOUNDED;

        "COLUMN GENERATION"
        DEFINE P "ENTERING" COLUMN M+1;
        P=INVERSE*AC(*,S);

        "PIVOT ROW SELECTION"
        DEFINE R "PIVOT ROW" INTEGER,THETA=INFINITY;
        FOR I IN (1,...,M):P(I)>ZERO,
        IF B(I)/P(I)<THETA,|_R=I;THETA=B(I)/P(I)_|;
        IF THETA=INFINITY, GO TO UNBOUNDED;
```

```
            " UPDATE INVERSE"
            INVERSE(R,*)=INVERSE(R,*)/P(R);
            FOR I IN (1,...,M+1):I¬=R,
            INVERSE(I,*)=INVERSE(I,*)-P(I)*INVERSE(R,*);

            "UPDATE BASIC VARIABLES AND OBJECTIVE"
            BASIC(R)=S;B(R)=THETA;
            FOR I IN (1,...,M):I¬=R,B(I)=B(I)-P(I)*THETA;
            Z=Z+P(M+1)*THETA;

            " RECYCLE"
            GO TO RECYCLE;


"SOLUTION"
 BOUNDED:DEFINE X "SOLUTION" ROW N;
           X=0;FOR I IN (1,...,M):BASIC(I)<=N,X(BASIC(I))=B(I);
           PUT SKIP DATA(X,Z);GO TO OUT;
UNBOUNDED:PUT SKIP LIST(<<UNBOUNDED>>);
OUT:END
```

It must be noticed here that several more features of MPL, not yet implemented in MPL/71, would stillenhancethe clarity of programs by introducing more standard notations for common operations. As an example, the concatenation of matrices would readily produce the augmented matrices.

in conclusion, the extensive use of standard mathematical notation in a programming language allows the mathematician-programmer to write compact statements, directly meaningful to him. As such, MPL programs could well serve as a communication language for mathematical algorithms.

The Generalized Upper-Bounding Algorithm [3].

Consider the following linear programming problem. Find

$$\min z = cx$$

such that $x > 0$

$$ax \leq b$$

$$\sum_{j=r_k}^{r_{k+1}-1} x_j = d_k, \qquad k = 1, \ldots, \ell$$

with $r_k < r_{k+1}$

where

$c = (c_j)$

$x = (x_j)$

$b = (b_j)$

$A = ((a_{ij}))$

$d = (d_k)$

Here, in addition to the usual matrix inequality constraint, the sums of disjoint sets of variables are fixed. Transportation problems, for example, exhibit such a structure.

Problems of this kind can, of course, be solved by the simplex method by incorporationg the sums into the augmented matrix constraint.

Such an approach, however, would prove infeasible in large problems, where the number of supplementary rows thus introduced could be too large to be dealt with efficiently.

The procedure about to be described here is a specialization of the revised simplex method, which solves such problems while maintaining a "working" basis of dimension m+1 by m+1.

Let us refer to the sets of variables defined by

$$\{x_k : r_k \leq k < r_{k+1}\} \quad , \qquad k = 1, \ldots, \ell$$

as the successive gub sets k.

The technique is developed from the following observation: "each set of basic variables for this problem must include at least one variable from each gub set k".

As the result of this property, it is then possible to identify in each basis a set of ℓ variables, called the key variables, representative of each gub set. The remaining basic variables, after proper elimination of the key variables, form the working basis. The method then follows the usual steps of the revised simplex as applied to this reduced system. Special attention must be given in maintaining a

complete set of key variables, since some of them may be candidates for leaving the basis of the original system at each iteration.

At the beginning of some iteration, assume that the current working basis inverse, B^{-1}, augmented as in the previous section, the set of key variables, x_{j_k}, the current solution, $\bar{b}_i$ and $\bar{d}_{k'}$ and the data of the original problem, are available. One step of the generalized upper-bounding algorithm works then as follows:

I Pricing: The revised costs for the entire system, $\bar{c}_j$, $j = 1,\ldots,n+m$, are given by

$$c_j = -\pi(P_j - P_{j_k}) \qquad \text{for } x_j \text{ in gubset } k$$

or

$$c_j = -\pi P_j - \mu_k$$

where $\mu_k = -\pi P_{j_k}$ is the simplex multiplier associated with the sum constraint over gubset k.

II Pivot column selection: The pivot column s is given by the usual criterion

$$\bar{c}_s = \min_j c_j$$

If $\bar{c}_s > 0$, the current solution is optimal.

III Column generation: The entering column from the augmented matrix A is given by

$$\bar{P}_s = B^{-1}(P_s - P_{j_\sigma}) \qquad \text{for } x_s \text{ in gubset } \sigma .$$

The entering column from the matrix constraint defined by sums is given by

$$\bar{Q}_s = (\bar{q}_{ks}), \qquad k = 1, \ldots, \ell$$

where

$$q_{ks} = - \sum_{i:x_i \text{ in gub set } k} \bar{a}_{is} \qquad \text{if } k \neq \sigma$$

$$\bar{q}_{\sigma s} = 1 - \sum_{i:x_i \text{ in gubset } k} \bar{a}_{is} .$$

IV Pivot row selection.

The pivot row **r** is given by performing

$$\theta = \min_{\substack{\bar{a}_{is} > 0 \\ \bar{q}_{ks} > 0}} \left(\frac{\bar{b}_i}{\bar{a}_{is}}, \frac{\bar{d}_k}{\bar{q}_{ks}} \right) .$$

If all $\bar{a}_{is}, \bar{q}_{ks} \leq 0$, the optimum is unbounded.

V Update the values of the basic variables (excluding the pivot).

The updated values are

$$\bar{b}_i \leftarrow \bar{b}_i - \theta \bar{a}_{is} , \qquad i,k \neq r$$

$$\bar{d}_k \leftarrow \bar{d}_k - \theta \bar{q}_{ks} .$$

VI Swapping, in necessary.

In the case where $\theta = \bar{d}_r/\bar{q}_{rs'}$ i.e., when the pivot variable is a key variable, another variable from the same gub set ρ must be made key. If a non-key basic variable, say the ith of working basis, can be found in gub set ρ, then swapping occurs, i.e., $r \leftarrow i$. The working inverse is then updated by the formula

$$B^{-1}_{ij} \leftarrow B^{-1}_{ij} \qquad i \neq r$$

$$B^{-1}_{rj} \leftarrow B^{-1}_{rj} - \sum_{i:x_i \text{ in gub set } \rho} B^{-1}_{ij}$$

If there is no non-key basic variable in gub set p, then the entering variable necessarily can be made key for gub set ρ. No pivoting is needed in this case.

VII Update working inverse and pivot variable.

The updated working inverse is given by pivoting the first $m+1$ columns of $[B^{-1} \vdots \bar{P}_s]$ on $\bar{a}_{rs'}$ and $\bar{b}_r = \theta$.

The Program COMPOSITE-GUB.

A complete implementation of the generalized upper-bounding algorithm just described must deal explicitly with the problem of finding an initial feasible working basis and its corresponding set of

key variables. The first method that we have considered is an extension of the usual two-phase method, known as the composite simplex method [5].

In short, whereas the two-phase method introduces artificial variables, the composite method allows the slack variables to take negative values. In the two phase method, feasibility and optimality are dealt with in two separate phases. In the composite method, feasibility and optimality can be worked toward simultaneously by a pricing procedure combining, with appropriate weights, the objective function and the sum of the negative slacks. However, the optimality criterion can be attained before feasibility is achieved, in which case new weights must be assigned. It is possible to ensure a successful termination of the process after at most two weight combinations, i.e., (1,0) and (0,1). In this case the procedure becomes equivalent to the two-phase method without artificial variables. This is the particular set-up that we have actually programmed in COMPOSITE-GUB.

The initial working basis is given by theslacks variables, and the key variables are selected with the smallest coefficient in each gub set. The corresponding basic solution will in general be infeasible, i.e., some $\bar{b}_i$ will be negative.

In the simplex iterations, the pivot row selection criterion must be extended to allow for negative slacks to become positive. Following [5], we take $\theta = \min(\theta_1, \theta_2)$, where θ_1 is given by the usual criterion restricted to positive $\bar{b}_i$, and

$$\theta_2 = \min_{\substack{\bar{b}_i < 0 \\ a_{is} < 0}} \frac{\bar{b}_i}{\bar{a}_{is}}$$

```
PROGRAM COMPOSITE_GUB;


"PROBLEM DESCRIPTION
FIND MIN Z=C*X SUCH THAT
                X>=0
                A*X<=B, WHERE A MATRIX MB YN
                SUM X(I)=D(K) FOR RANGE(K)<=I<RANGE(K+1), K=1,...,L
IT IS ASSUMED RANGE(1)=1"


"DATA"
GIVEN(M,N,L)INTEGER,
      "A MATRIX MBYN, THIS STATEMENT REPLACED BELOW"
       B COLUMN M,
       C ROW N,
       D COLUMN L,
       RANGE INTEGER ROW L+1;
$LET GUB_K=(RANGE(K),...,RANGE(K+1)-1);
$LET ZERO=1E-4;
$LET INFINITY=1E6;


"INPUT SPARSE A AND CONSTRUCT AUGMENTED MATRIX"
DEFINE AC "AUGMENTED" MATRIX M+1 BY M+N,(I,J) "SUBSCRIPTS" INTEGER,A;
AC=0;IN:GET LIST(J);IF J>0,|_GET LIST(I,A);AC(I,J)=A;GO TO IN_|;
FOR I IN (1,...,M),AC(I,N+I)=1 "SLACKS";
FOR J IN (1,...,N),AC(M+1,J)=-C(J) "COSTS";


"INITIAL WORKING BASIS"
DEFINE BASIC INTEGER COLUMN M "INDEX FOR UON-KEY BASIC VARIABLES";
FOR I IN (1,...,M),BASIC(I)=N+I "INITIAL NON-KEY BASIC ARE SLACKS";
DEFINE SET INTEGER COLUMN M "SET INDEX FOR NON-KEY BASIC";
SET=0 "SLACKS NOT IN ANY SET";


"INITIAL KEY VARIABLES, CHOSEN WITH SMALLEST COEFFICIENTS"
DEFINE KEY INTEGER COLUMN L "INDEX FOR KEY BASIC",K "SUBSCRIPT" INTEGER;
FOR K IN (1,...,L),|_DEFINE TEST=INFINITY;FOR J IN GUB_K,
                    IF C(J)<TEST,|_KEY(K)=J;TEST=C(J)_|_|;


"INITIAL INVERSE"
DEFINE INVERSE MATRIX M+1 BY M+1;
INVERSE=0;FOR I IN( 1,...,M+1),INVERSE(I,I)=1 "IDENTITY MATRIX";


"INITIAL VALUES FOR NON-KEY BASIC AND OBJECTIVE"
DEFINE Z=0.0 "OBJECTIVE";
FOR K IN (1,...,L),|_FOR I IN (1,...,M),B(I)=B(I)-AC(I,KEY(K))*D(K);
                    Z=Z+C(KEY(K))*D(K)_|;


"CHECK FEASIBILITY"
DEFINE PHASE INTEGER "INDEX FOR PHASE";
DEFINE INFEASIBILITY ROW M+1 "INDEX FOR INFEASIBLE BASIC";
CHECK:PHASE=2;INFEASIBILITY=0 "ASSUMING FEASIBILITY";
      FOR I IN (1,...,M):B(I)<-ZERO,|_PHASE=1;INFEASIBILITY(I)=1
```

```
"SIMPLEX ITERATION"

ITERATE:"PRICING"
        DEFINE PI ROW M+1 "PRICES FOR INEQUALITIES";
        IF PHASE=1,PI=-INFEASIBILITY*INVERSE;
        IF PHASE=2,PI=INVERSE(M+1,*);
        DEFINE COST ROW M+N "RELATIVE COSTS";
        COST=-PI*AC;
        DEFINE MU POW L "PR ICES FOR SUMS":
        FOR K IN (1,...,L),|_MU(K)=-PI*AC(*,KEY(K));
                           FOR J IN GUB_K,COST(J)=COST(J)-MU(K)_|;


        "PIVOT COLUMN SELECTION"
        DEFINE S "PIVOT COLUMN" INTEGER, DELTA=INFINITY;
        FOR J IN (1,...,M+N),IF COST(J)<DELTA,|_S=J;DELTA=COST(J)_|;
        IF DELTA>-ZERO, |_IF PHASE=1, GO TO INFEASIBLE;
                          I   FPHASE=2, GO TO BOUNDED _|;


        "COLUMN GENERAT ION"
        DEFINE SIGMA=0 "SET INDEX F  O  R ENTERING VARIABLE";
        FOR K IN (1 v-i'..,L),IF RANGE(K)<=S AND S<RANGE(K+1),SIGMA=K;
        DEFINE P "ENTER ING" COLUMN M+1 " F R O M INEQUALITIES";
        P=INVERSE*AC(*,S);
        I fSIGMA>0,P=P-INVERSE*AC(*,KEY(SIGMA));
        DEFINE Q "ENTER ING" COLUMN L "FROM SUMS";
        FOR K IN (1,...,L),|_Q(K)=0;FOR I I  N(1,...,M):SET(I)=K,
                           Q(K)=Q(K)-P(I)_|;
        IF SIGMA>0, Q(SIGMA)=Q(SIGMA)+1;


        "PIVOT ROW SELECTION"
        DEFINE R "PIVOT R O W" INTEGER, THETA=INFINITY;
        FOR I IN(1,...,M):P(I)>ZERO A N D B(I)>-ZERO,
        I FB(I)/P(I)<THETA,|_R=I;THETA=B(I)/P(I)_|;
        FOR K IN (1 ,. . . ,L):Q(K)>ZERO,
        IF D(K)/Q(K)<THETA,|_R=M+K;THETA=D(K)/Q(K)_|;
        IF PHASE=1, |_DEF INE R2 "ALTERNATE" INTEGER,THETA2=-INFINITY;
                    FOR I IN (1,...,M):P(I)<-ZERO AND B(I)<-ZERO,
                    I FB(I)/P(I)>THETA2,|_R2=I;THETA2=B(I)/P(I)_|;
                    I FTHETA2¬=-INFINITY AND THETA2<THETA,
                    |_R=R2;THETA=THETA2 _|_| ;
        IF THETA=INFINITY, GO TO UNBOUNDED ;


        "UPDATE BASIC VARIABLES (EXCLUDING PIVOT) AND OBJECTIVE"
        FOR I IN (1,...,M):I¬=R,B(I)=B(I)-P(I)*THETA;
        FOR K IN (1,. . .,L):K¬=R-M,D(K)=D(K)-Q(K)*THETA;
        Z=Z+P(M+1)*THETA;


        "SWAPPING, IF NECESSARY"
        IF R>M "PIVOT IS KEY",
              |_DEFINE RHO=R-M "SET INDEX FOR PIVOT";
                FOR I IN (1,...,M),IF SET(I)=RHO "NON-KEY IN SET RHO",
                |_R=I;P(R)=Q(RHO) "PI VOT I S MADE NON-KEY";
                  KEY(RHO)=BASIC(R);D(RHO)=B(R)"NON-KEY MADE K E Y " ;
```

```
                    INVERSE(R,*)=-INVERSE(R,*);
                    FOR I IN (1,...,M):SET(I)=RHO AND I¬=R,
                    INVERSE(R,*)=INVERSE(R,*)-INVERSE( I,*);GO TO PIVOT _| ;

                  "NO NON-KEY FROM SET RHO"
                  D( RHO)=THETA;KEY(RHO)=S "ENTERING VARIABLE MADE KEY";
                  GO TO RECYCLE "PIVOTING NOT NEEDED"_|;


        "UPDATE INVERSE"
  PIVOT:INVERSE(R,*)=INVERSE(R,*)/P(R);
        FOR I IN (1,...,M+1):I¬=R,
        INVERSE(I,*)=INVERSE(I,*)-P(I)*INVERSE(R,*);
        "UPDATE PIVOT VARIABLE"
        B(R)=THETA;BASIC(R)=S;SET(R)=SIGMA;


        "RECYCLE"
RECYCLE:IF PHASE=1, GO TO CHECK;
        I F   PHASE=2,GOTOITERATE;


"SOLUTION"
BOUNDED:DEFINE X "SOLUTION" ROW N;X=0;
        FOR I IN (1,...,M):BASIC(I)<=N,X(BASIC(I))=B(I);
        FOR K IN (1,...,L),X(KEY(K))=D(K);
        PUT SKIP DATA(X,Z);GO TO OUT;
INFEASIBLE:PUT SKIP LIST(<<INFEASIBLE>>);GO TO OUT;
UNBOUNDED:PUT SKIP LIST(<<UNBOUNDED>>);

OUT:END
```

V The program DUAL-GUB [6].

The initial basis defined by the variables with smallest coefficient in each gub set and completed with the slack variables is dual feasible, i.e., all $\bar{c}_j \geq 0$. Furthermore $\bar{c}_i = 0$ for i basic (complementary slackness) and some $\bar{b}_i < 0$ (primal feasibility relaxed). This is the setting required for the dual simplex method, which works toward primal feasibility while maintaining dual feasibility. The algorithm differs from the previous one essentially in the choice of the pivot only. No phase I is needed.

The following changes are required:

1) The pivot row r is given by performing

$$\theta = \min_{\substack{\bar{b}_i < 0 \\ \bar{d}_k < 0}} (\bar{b}_i, \bar{d}_k)$$

If all $\bar{b}_i, \bar{d}_k \geq 0$, the current solution is optimal.

2) Swapping, if necessary: If $\theta = \bar{d}_r < 0$, another basic variable is necessarily to be found in the same gub set (because of $d_r \geq 0$) and swapping can always take place.

3) Row generation: The transformed row $(\bar{a}_{rj})$, j = 1, . . . , n+m, is defined by

$$a_{rj} = \pi P_j - \mu_k \qquad \text{for } x_j \text{ in gub set } k$$

where

$$\mu_k = \pi P_{jk} \ .$$

4) Pivot column selection: The pivot column s is given by

$$\frac{c_s}{-a_{rs}} = \min_{a_{rj} < 0} \frac{-\bar{c}_j}{-a_{rj}} \quad .$$

if all $\bar{a}_{rj} \geq 0$, the primal is infeasible. We can then proceed with the usual column-generation, pivoting and updating.

VI Results and Conclusions.

It has been an easy matter to get the program DUAL-GUB by just deleting and inserting a few lines in the previous program COMPOSITE-GUB.

The two programs have been first tested on small examples (e.g., m = 2, n = 8, ℓ = 5). By taking advantage of the availability of a free dual feasible basis, the dual algorithm always needed a few less iterations to provide the solution. But it was expected that, in larger problems, the extra computation required by the row generation, which amounts to another pricing operation, would prove disadvantageous.

This was confirmed by our last experiment. We run each program for 5 minutes on a problem with m = 26, n = 527, ℓ = 100. COMPOSITE-GUB provided a solution after some 180 iterations were executed, at the rate of 45 per minute. DUAL-GUB carried about 120 iterations, at the rate of 30 per minute, but primal feasibility was not achieved.

```
PROGRAM DUAL,GUB;

"PROBLEM DESCRIPTION
FIND MIN Z=C*X SUCH THAT
                X>=0
                A*X<=B, WHERE A MATRIX M BY N
                SUM X(I)=D(K) FOR     RANGE(K)<=I<RANGE(K+1), K=1,...,L
IT IS ASSUMED RANGE(1)=1"


"DATA"
GIVEN (M,N,L) INTEGER,
      "A MATRIX MBYN, THIS STATEMENT REPLACED BELOW"
       B COLUMN M,
       C ROW N,
       D COLUMN L,
       RANGE INTEGER ROW L+1;
$LET GUB_K=(RANGE(K),...,RANGE(K+1)-1);
$LET ZERO=1E-3;
$LET INFINITY=1E6;


"INPUT SPARSE A AND CONSTRUCT AUGMENTED MATRIX"
DEFINE AC "AUGMENTED" MATRIX M+1 BY M+N,(I,J) "SUBSCRIPTS" INTEGER,A;
AC=0;IN:GETLIST(J);IF J>0,|_GET LIST(I,A);AC(I,J)=A;GO TO IN _|;
FOR I IN (1,... ,M),AC(I,N+I)=1 "SLACKS";
FOR J IN (1,...,N),AC(M+1,J)=-C(J) "COSTS";


"INITIAL WORKING BASIS"
DEFINE BASIC INTEGER COLUMN M "INDEX FOR YON-KEY BASIC VARIABLES";
FOR I IN (1,...,M),BASIC(I)=N+I "INITIAL NON-KEY BASIC ARE SLACKS";
DEFINE SET INTEGER COLUMN M "SET INDEX FOR NON-KEY BASIC";
SET=0 "SLACKS NOT IN ANY SET";


"INITIAL KEY VARIABLES, CHOSEN WITH SMALLEST COEFFICIENTS"
DEFINE KEY INTEGER COLUMN L "INDEX FOR KEY BASIC",K "SUBSCRIPT" INTEGER;
FOR K IN (1,...,L),|_DEFINE TEST=INFINITY;FOR J IN GUB_K,
                    IF C(J)<TEST,|_KEY(K)=J;TEST=C(J)_|_|;


"INITIAL INVERSE"
DEFINE INVERSE MATRIX M+1 BY M+1;
INVERSE=0;FOR I IN (1,...,M+1),INVERSE(I,I)=1 "IDENTITY MATRIX";


"INITIAL VALUES FOR NON-KEY BASIC AND OBJECTIVE"
DEFINE Z=0.0 "OBJECTIVE":
FOR K IN (1,...,L),|_FOR I IN (1,...,M),B(I)=B(I)-AC(I,KEY(K))*D(K);
                    Z=Z+C(KEY(K))*D(K)_|;
```

```
"SIMPLEX ITERATION"

RECYCLE:"PRICING"
        DEFINE PI ROW M+1 '@PRICES FOR INEQUALITIES";
        PI=INVERSE(M+1,*);
        OEFINE COST ROW M+N '@RELATIVE COSTS",;
        COST=-PI*AC;
        DEFINE MU ROW L "PRICES FOR SUMS";
        FOR K IN(1,...,L),|_MU(K)=-PI*AC(*,KEY(K));
                          FOR J IN GUB_K,COST(J)=COST(J)-MU(K)_|;


        "PIVOT ROW SELECTION"
        DEFINE R "PIVOT ROW" INTEGER, THETA=INFINITY;
        FOR I IN(1,...,M),IF B(I)<THETA,|_R=I;THETA=B(I)_|;
        FOR K IN(1,...,L),IF D(K)<THETA,|_R=M+K;THETA=D(K)_|;
        IF THETA>-ZERO, GO TO FEASIBLE;


        "SWAPPING, IF NECESSARY"
        IF R>M "PIVOT IS KEY",
              |_DEFINE RHO=R-M  "SET  INDEX FOR PIVOT";
                FOR I IN (1,...,M),IF SET(I)=RHO "NON-KEY IN SET RHO",
                |_R=I "PIVOT MADE NON-KEY";
                  KEY(RHO)=BASIC(R);D(RHO)=B(R) "NON-KEY MADE KEY";
                  INVERSE(R,*)=-INVERSE(R,*);
                  FOR I IN (1,...,M):SET(I)=RHO AND I~=R,
                  INVERSE(R,*)=INVERSE(R,*)-INVERSE(I,*)_|_|;


        "ROW GENERATION"
        DEFINE T "ENTERING" ROW M+N;
        PI=INVERSE(R,*);
        T=PI*AC;
        FOR K IN(1,...,L),|_MU(K)=PI*AC(*,KEY(K));
                          FOR J IN GUB_K,T(J)=T(J)-MU(K)_|;


        "PIVOT COLUMN SELECTION"
        DEFINE S "PIVOT COLUMN" INTEGER, DELTA=INFINITY;
        FOR J IN (1,...,M+N):T(J)<-ZERO,
        IF COST(J)/(-T(J))<DELTA,|_S=J;DELTA=COST(J)/(-T(J))_|;
        IF DELTA=INFINITY, GO TO INFEASIBLE;


        "COLUMN GENERATION"
        DEFINE SIGMA=0 "SET INDEX FOR ENTERING VARIABLE";
        FOR K IN (1,...,L),IF RANGE(K)<=S AND S<RANGE(K+1),SIGMA=K;
        DEFINE P "ENTERING" COLUMN M+1 "FROM INEQUALITIES";
        P=INVERSE*AC(*,S);
        IF SIGMA>0,P=P-INVERSE*AC(*,KEY(SIGMA));
        DEFINE Q "ENTERING" COLUMN L "FROM SUMS";
        FOR K IN (1,...,L),|_Q(K)=0;FOR I IN(1,...,M):SET(I)=K,
                           Q(K)=Q(K)-P(I)_|;
        IF SIGMA>0,Q(SIGMA)=Q(SIGMA)+1;
        THETA=THETA/P(R);


        "UPDATE BASIC VARIABLES AND OBJECTIVE"
        B(R)=THETA;BASIC(R)=S;SET(R)=SIGMA;
```

```
        FOR I IN (1,...,M):I¬=R,B(I)=B(I)-P(I)*THETA;
        FORKIN(1,...,L),D(K)=D(K)-Q(K)*THETA;
        Z=Z+P(M+1)*THETA;


        "UPDATE INVERSE"
PIVOT:INVERSE(R,*)=INVERSE(R,*)/P(R);
        FOR I IN(1,...,M+1):I¬=R,
        INVERSE(I,*)=INVERSE(I,*)-P(I)*INVERSE(R,*);


        "RECYCLE"
        GO TO RECYCLE;



"SOLUTION"
FEASIBLE:DEFINE X "SOLUTION" ROW N;
        X=0;FOR I IN(1,...,M):BASIC(I)<=N,X(BASIC( I))=B(I);
        FORKIN(1,. . o ,L),X(KEY(K))=D(K);
        PUT SKIP DATA(X,Z);GO TO OUT;
INFEASIBLE:PUT SKIP LIST(<<INFEASIBLE>>);

OUT: END
```

The performance of these programs, while not measuring with commercial codes, is good enough to view the present implementation as capable of supporting the kind of experiments usually attempted when testing mathematical algorithms.

The full credit that MPL deserves will of course be greatly enhanced by the introduction of additional standard mathematical notations, as announced in the specification manual. A more efficient compiler also will prove necesary before MPL will possibly be used for production applications.

REFERENCES

[1] Mathematical Programming Language Specification Manual for Committee Review, Technical Report STAN-CS-70-187, Stanford University, November 1970.

[2] Dantzig, G. B., Linear Programming and Extensions, Princeton University Press, 1963.

[3] Dantzig, G. B. and Van Slyke, R. M., Generalized Upper Bounding Techniques, J. Comp. Syst. Sc., vol. 1, 1967 (213-226).

[4] Lasdon, L. S., Optimization Theory for Large Systems, The MacMillan Company, 1970.

[5] Wolfe, P., The Composite Simplex Algorithm, SIAM Review, vol. 7, No. 1, 1965 (42-54).

[6] Grigoriadis, M. D., A Dual Generalized Upper Bounding Technique, Management Science, Vol. 17, No. 5, 1971 (269-284).

[7] McGrath, M., Documentation for MPL 71 Translator, Dept. of Computer Science, Stanford University, October 1971.