INTEGER PROGRAMMING WITH MINOS

A User's Guide

Stanford Business Software, Inc.

INTEGER PROGRAMMING WITH MINOS

A User's Guide

INTEGER PROGRAMMING WITH MINOS

A User's Guide

First Edition

Stanford Business Software, Inc.
2680 Bayshore Parkway, Suite 304
Mountain View, CA 94043

INTEGER PROGRAMMING WITH MINOS

This manual was typeset using LaTeX macros with Knuth's TeX typesetting system.

Publisher: Stanford Business Software, Inc.
2680 Bayshore Parkway, Suite 304
Mountain View, CA 94043

Contents

List of Tables

Chapter 1

INTRODUCTION

BIPMINOS and MIPMINOS refer to simple branch-and-bound procedures which, together with MINOS (Version 5.4 or later), allow the MINOS user to solve integer programs. BIPMINOS solves binary integer linear programs; MIPMINOS is designed to solve mixed integer linear programs, or any linear program where some or all of the variables are integer multiples of given increments. The BIPMINOS routines can be found in the file BIPFUNS.FOR; the MIPMINOS routines are found in the file MIPFUNS.FOR.

The BIPMINOS and MIPMINOS routines are designed to execute a simple textbook version of the branch-and-bound technique for integer programming; thus they should work fine for problems with a few integer variables. However, because source code is provided, the sophisticated user may be able to modify BIPMINOS and MIPMINOS to create powerful and efficient versions of the branch-and-bound algorithm. MINOS users can easily modify these routines in order to experiment with different fathoming criteria, branching rules, and bound choices. Because MINOS is a powerful and efficient non-linear program solver, BIPMINOS and MIPMINOS could possibly be used to solve integer non-linear programs; however, this has not been tested and, it is conceivable that, should you decide to solve integer non-linear programs, you would have to modify the source code.

MIPMINOS, the mixed integer programming package, allows the user to solve integer programs where some or all of the variables are integer multiples of user-specified increments; these increments may or may not be integer-valued themselves. This approach may allow for more stability in the problems the user wishes to solve, since it will not be necessary to scale the problem constraints to deal with strictly integer-valued variables. This approach also helps the user to avoid some of the complications involved with bound selection in the branch-and-bound algorithm when some of the variables cannot take on all integer values.

In the chapters that follow, we describe how to set up and use BIPMINOS and MIPMINOS to solve binary and mixed integer linear programs. Although we will discuss every-

thing you need to know about BIPMINOS and MIPMINOS to solve mixed integer linear programs, we assume that the user has already used MINOS to solve linear programs. In any case, we recommend that you keep a copy of the MINOS User's Manual nearby for consultation.

Chapter 2

BINARY INTEGER PROGRAMMING (BIPMINOS)

BIPMINOS refers to a set of routines which, together with MINOS, allow you to solve binary integer linear programs. BIPMINOS uses a simple branch-and-bound algorithm, as specified by Dakin. (See Section A.1 in the Appendix for a detailed explanation of the branch-and-bound method used for integer programming.)

In this chapter, we describe how to set up the BIPMINOS executable program and the BIPMINOS input files in order to solve a binary integer linear program. A simple example of binary integer linear programming is presented to illustrate the setup process.

2.1 SETTING UP BIPMINOS

REQUIREMENTS

To set up BIPMINOS, you will need:

- a copy of the MINOS source code (Version 5.4 or later),
- a Fortran 77 compiler,
- the IPMINOS diskette which contains the BIPMINOS source code, and,
- a computer with sufficient memory (see the MINOS User's Manual for details on memory requirements).

Note: If you decide to set up BIPMINOS on a PC, we recommend that you use a Fortran 77 compiler that supports extended memory.

CREATING THE BIPMINOS EXECUTABLE PROGRAM

In this section we describe the steps to create the BIPMINOS executable program on your PC assuming that you have available the Lahey Fortran 77 F77L-EM/32 Version 5.01 compiler.[1]

1. Go to the MINOS directory.

   ```
   C:\> cd \minos
   ```

2. Transfer all files from the BIPMINOS diskette to the MINOS directory, or the directory where your MINOS files reside (we assume the directory is named MINOS).

   ```
   C:MINOS\> copy a:*.*
   ```

3. Because of the way MINOS is set up, you will have to make a small modification to the file MINOSL.FOR. Comment out the line

   ```
   entry matmod
   ```

   which is towards the end of MINOSL.FOR. Remember to save the file with this change. This modification is necessary because the mixed integer programming routines are called through subroutine MATMOD, which is in the file BIPFUNS.FOR.

4. The batch file BIPPCCOM.BAT is used to compile the source code for BIPMINOS and link the resulting .OBJ files to create an executable BIPMINOS.EXE. The batch file is invoked by typing:

   ```
   C:\MINOS> bippccom
   ```

5. A sample problem is included on your IPMINOS diskette. The example uses three files:

   - BIPEX.SPC,
   - BIPEX.MPS, and
   - BIPBIPEX.MPS.

   To try BIPMINOS on the sample problem, copy files as follows.

[1] Should you decide to use a different compiler (or a different platform) please refer to your compiler manual and modify the files BIPCCOM.BAT and BIPPCLP.LNK appropriately.

```
C:\MINOS> copy  BIPEX.*  MINOS.*
```

and

```
C:\MINOS> copy  BIPBIPEX.MPS  BIPMINOS.MPS
```

6. Execute BIPMINOS by issuing the following command.

```
C:\MINOS> BIPMINOS
```

You should get an objective function value of -28 for the above mentioned example. For details on creating your own input files please refer to Sections 2.2 and 2.3.

NOTES:

- The batch file BIPPCCOM.BAT uses the default compiler options for the Lahey F77L-EM/32 Version 5.01 compiler. This results in creation of files with a .SLD extension which are used by Lahey's symbolic online debugger. Add the /NS option to the batch file to suppress generation of these files.

- Lahey Fortran allows you to reduce the executable size of BIPMINOS by putting the workspace array Z in a blank common block; the compiler will create a smaller executable that allocates memory for Z during execution rather. If you wish to make this change, you will need to modify the file MINOSL.FOR.

- If you plan to use a compiler other than Lahey F77L-EM/32, check to see if it supports extended memory. If your compiler supports extended memory, you should be to able compile and link by modifying the batch file BIPPCCOM.BAT and link file BIPPCLP.LNK. If your compiler does not support extended memory, you will need to study the source files and create overlays. The creation of overlays is done during link time for most compilers.

- We have not tried to solve non-linear integer programs using the BIPMINOS routines. However, you should be able to create a non-linear version of BIPMINOS by modifying the compile and link files supplied with MINOS. The only source file you should have to modify is MI05FUNS.FOR, which contains the default MATMOD routine for the non-linear version of MINOS.

2.2 SETTING UP A BINARY INTEGER PROGRAM

The PC version of BIPMINOS uses the following files for input and output.

MINOS.SPC	SPECIFICATIONS File	(Input)
MINOS.MPS	MPS File	(Input)
BIPMINOS.MPS	INTEGER VARIABLE SPECIFICATION File	(Input)
MINOS.PRN	PRINT File	(Output)

The four files just listed are the only files you will really need for binary integer programming. The three files MINOS.SPC, MINOS.MPS, and MINOS.PRN, are the standard MINOS files; *the only new file is BIPMINOS.MPS* (described in Section 2.2.3). Note: on systems other than a PC, MINOS file names may automatically be supplied by the system; please refer to the MINOS User's Manual and/or the files with extension .DOC supplied on your MINOS diskette.

Because BIPMINOS adds on to MINOS, it can also be modified to read from or write to all the other files that MINOS can access. If you want to use the information in these files for integer programming purposes, you may need to work with the specification file, or even modify the integer programming routines. See the MINOS User's Guide for a description of these files.

2.2.1 SPECIFYING MINOS PARAMETERS

The MINOS specifications file should be set up as if you were solving an ordinary linear program, with two important exceptions and a suggestion for improving efficiency.

- *CYCLE LIMIT.* The number of branch-and-bound subproblems solved by BIPMINOS is controlled by the CYCLE LIMIT parameter found in the MINOS specifications file. For this reason, include the line

  `CYCLE LIMIT` l

  where l is one more than the maximum number of subproblems you are willing to solve. If $l-1$ subproblems have been solved, and not all subproblems have been fathomed, BIPMINOS will write the best solution found up to that point to the output file.

- *WORKSPACE (USER).* You will need to reserve a portion of the MINOS workspace for use by the binary integer programming routines. Include the line

  `WORKSPACE (USER)` *maxw*

where *maxw* is the number of spaces in the workspace array that you will allocate to the arrays needed for binary integer programming. A good rule of thumb for selecting *maxw* is to set

$$maxw \geq 6n + 2m + 3l,$$

where n is the number of variables, m is the number of constraints, and l is the CYCLE LIMIT.

- *SUMMARY FILE.* If you set the SUMMARY FILE option to 0 in order to turn off the MINOS screen output, BIPMINOS will run much faster.

2.2.2 SPECIFYING LINEAR PROBLEM DATA (MPS File)

All variables, and the constraints and bounds associated with the variables, are specified in the MPS data file required by MINOS for a linear program. However, you do *not* need to specify the lower and upper bounds on the (0,1)-valued variables; BIPMINOS will do this for you once you specify the binary integer variables as described in the next section.

2.2.3 SPECIFYING INTEGER VARIABLES

File BIPMINOS.MPS is used to specify which variables in the linear program described by the specifications file and the MPS data file are $(0, 1)$-valued. BIPMINOS.MPS uses a format similar to that found in an MPS file; there is a header card, an end-of-data card, and the lines in-between must have the following data format:

Columns 5-12
Contents Variable Name

In other words, the variable name is placed left justified in Columns 5-12 of the line.

For example, if we have a binary integer program with variables named X01, X02, and X03, with X01 and X03 being integer-valued, BIPMINOS.MPS would take the following form.

```
INTEGERS
    X01
    X03
ENDATA
```

2.2.4 BIPMINOS OUTPUT

The solution to the binary Integer Program is written to the PRINT file used by MINOS, if a solution has been found. If not all subproblems have been fathomed after the CYCLE LIMIT has been reached, the incumbent subproblem (i.e. the best solution found to date)

is written to the PRINT file. If no incumbent has been found, or if the problem has no feasible solution, the solution to the last subproblem solved and an error message is written to the PRINT file.

2.3 AN EXAMPLE

An example problem is described in this section to illustrate how you would set up the BIPMINOS input files.

The specifications file, MPS data file, and BIPMINOS.MPS file for this problem are shown in Tables 2.1, 2.2 and 2.3 respectively; they are stored in the three files BIPEX.SPC, BIPEX.MPS, and BIPBIPEX.MPS on the IPMINOS diskette.

$$
\begin{array}{lrrrrcr}
\text{maximize} & 9x_1 & +5x_2 & +6x_3 & +4x_4 & = & z \\
\text{subject to} & 6x_1 & +3x_2 & +5x_3 & +2x_4 & \leq & 10 \\
 & & & +x_3 & +x_4 & \leq & 1 \\
 & -1x_1 & & +x_3 & & \leq & 0 \\
 & & -x_2 & & +x_4 & \leq & 0 \\
 & x_1, & x_2, & x_3, & x_4 & \in & \{0,1\}
\end{array}
$$

The solution to the above problem is $x = (1,1,0,0)^T$, $z = 14$; it takes 11 iterations to fathom all subproblems using the BIPMINOS version of the branch-and-bound algorithm. An additional iteration is required to verify that the optimal solution satisfies all constraints.

```
Begin BIPEX         (Sample Binary Integer Program)
   Maximize
   Rows                    20
   Columns                 30
   Elements               100
   Cycle Limit             70  * Limits # of B&B problems solved
   MPS file                10

   Iterations             100
   Print level              1  * OK for small problems
   Print frequency          1
   Summary frequency        1
   Workspace (user)      1000  * Allocate workspace for B&B routines
End BIPEX
```

Table 2.1: Specification File (BIPMINOS Example)

```
* This is the .MPS file for the BIP described in the manual.
NAME          BIPEX
ROWS
 L  ROW1
 L  ROW2
 L  ROW3
 L  ROW4
 N  COST
COLUMNS
    COLUMN1   ROW1          6.0
    COLUMN1   ROW3         -1.0
    COLUMN1   COST          9.0
    COLUMN2   ROW1          3.0
    COLUMN2   ROW4         -1.0
    COLUMN2   COST          5.0
    COLUMN3   ROW1          5.0
    COLUMN3   ROW2          1.0
    COLUMN3   ROW3          1.0
    COLUMN3   COST          6.0
    COLUMN4   ROW1          2.0
    COLUMN4   ROW2          1.0
    COLUMN4   ROW4          1.0
    COLUMN4   COST          4.0
RHS
    DEMANDS   ROW1         10.0
    DEMANDS   ROW2          1.0
    DEMANDS   ROW3          0.0
    DEMANDS   ROW4          0.0
ENDATA
```

Table 2.2: MPS File (BIPMINOS Example)

```
INTEGERS   * BIPMINOS.MPS file for the BIP example presented in the manual.
    COLUMN1
    COLUMN2
    COLUMN3
    COLUMN4
ENDATA
```

Table 2.3: Integer Specification File (BIPMINOS Example)

Chapter 3

MIXED INTEGER PROGRAMMING (MIPMINOS)

MIPMINOS refers to a set of routines which, together with MINOS, allow you to solve mixed integer linear programs, or any linear program where some or all of the variables are integer multiples of given increments. MIPMINOS uses a simple branch-and-bound algorithm, as specified by Dakin. (See Section A.1 in Appendix A for a detailed explanation of the branch-and-bound method used for mixed integer programming.)

In this chapter, we describe how to set up the MIPMINOS executable program and the MIPMINOS input files in order to solve a mixed integer linear program. A simple example of mixed integer linear programming is presented to illustrate the setup process.

3.1 SETTING UP MIPMINOS

REQUIREMENTS

To set up MIPMINOS, you will need:

- a copy of the MINOS source code (Version 5.4 or later),
- a Fortran 77 compiler,
- the IPMINOS diskette which contains the MIPMINOS source code, and,
- a computer with sufficient memory (see the MINOS User's Manual for details on memory requirements).

Note: If you decide to set up IPMINOS on a PC, we recommend that you use a Fortran 77 compiler that supports extended memory.

CREATING THE MIPMINOS EXECUTABLE PROGRAM

In this section we describe the steps to create the MIPMINOS executable program on your PC assuming that you have available the Lahey Fortran 77 F77L-EM/32 Version 5.01 compiler.[1]

1. Go to the MINOS directory.

   ```
   C:\> cd \minos
   ```

2. Transfer all files from the MIPMINOS diskette to the MINOS directory, or the directory where your MINOS files reside (we assume the directory is named MINOS).

   ```
   C:MINOS\> copy a:*.*
   ```

3. Because of the way MINOS is set up, you will have to make a small modification to the file MINOSL.FOR. Comment out the line

   ```
   entry matmod
   ```

   which is towards the end of MINOSL.FOR. Remember to save the file with this change. This modification is necessary because the mixed integer programming routines are called through subroutine MATMOD, which is in the file MIPFUNS.FOR.

4. The batch file MIPPCCOM.BAT is used to compile the source code for MIPMINOS and link the resulting .OBJ files to create an executable MIPMINOS.EXE. The batch file is invoked by typing:

   ```
   C:\MINOS> mippccom
   ```

5. A sample problem is included on your IPMINOS diskette. The example uses three files:

   - MIPEX.SPC,
   - MIPEX.MPS, and
   - MIPMIPEX.MPS.

   To try MIPMINOS on the sample problem, copy files as follows.

[1] Should you decide to use a different compiler (or a different platform) please refer to your compiler manual and modify the files MIPCCOM.BAT and MIPPCLP.LNK appropriately.

```
C:\MINOS> copy  MIPEX.*  MINOS.*
```

and

```
C:\MINOS> copy  MIPMIPEX.MPS  MIPMINOS.MPS
```

6. Execute MIPMINOS by issuing the following command.

```
C:\MINOS> MIPMINOS
```

You should get an objective function value of -28 for the above mentioned example. For details on creating your own input files please refer to Sections 3.2 and 3.3.

NOTES:

- The batch file MIPPCCOM.BAT uses the default compiler options for the Lahey F77L-EM/32 Version 5.01 compiler. This results in creation of files with a .SLD extension which are used by Lahey's symbolic online debugger. Add the /NS option to the batch file to suppress generation of these files.

- Lahey Fortran allows you to reduce the executable size of MIPMINOS by putting the workspace array Z in a blank common block; the compiler will create a smaller executable that allocates memory for Z during execution rather. If you wish to make this change, you will need to modify the file MINOSL.FOR.

- If you plan to use a compiler other than Lahey F77L-EM/32, check to see if it supports extended memory. If your compiler supports extended memory, you should be to able compile and link by modifying the batch file MIPPCCOM.BAT and link file MIPPCLP.LNK. If your compiler does not support extended memory, you will need to study the source files and create overlays. The creation of overlays is done during link time for most compilers.

- We have not tried to solve non-linear integer programs using the MIPMINOS routines. However, you should be able to create a non-linear version of MIPMINOS by modifying the compile and link files supplied with MINOS. The only source file you should have to modify is MI05FUNS.FOR, which contains the default MATMOD routine for the non-linear version of MINOS.

3.2 SETTING UP A MIXED INTEGER PROGRAM

The PC version of MIPMINOS uses the following files for input and output.

MINOS.SPC	SPECIFICATIONS File	(Input)
MINOS.MPS	MPS File	(Input)
MIPMINOS.MPS	INTEGER VARIABLE SPECIFICATION File	(Input)
MINOS.PRN	PRINT File	(Output)

The four files just listed are the only files you will really need for mixed integer programming. MINOS.SPC, MINOS.MPS and MINOS.PRN are standard MINOS files; *the only new file is MIPMINOS.MPS* (described in Section 3.2.3). Note: on systems other than a PC, MINOS file names may automatically be supplied by the system; please refer to the MINOS User's Manual and/or the files with extension .DOC supplied on your MINOS diskette.

Because MIPMINOS adds on to MINOS, it can also be modified to read from or write to all the other files that MINOS can access. If you want to use the information in these files for integer programming purposes, you may need to work with the specification file, or even modify the integer programming routines. See the MINOS User's Guide for a description of these files.

3.2.1 SPECIFYING MINOS PARAMETERS

The MINOS specifications file should be set up as if you were solving an ordinary linear program, with two important exceptions and a suggestion for improving efficiency.

- *CYCLE LIMIT.* The number of branch-and-bound subproblems solved by MIPMINOS is controlled by the CYCLE LIMIT parameter found in the MINOS specifications file. For this reason, include the line

  `CYCLE LIMIT` l

  where l is one more than the maximum number of subproblems you are willing to solve. If $l-1$ subproblems have been solved, and not all subproblems have been fathomed, MIPMINOS will write the best solution found up to that point to the output file.

- *WORKSPACE (USER).* You will need to reserve a portion of the MINOS workspace for use by the mixed integer programming routines. Include the line

  `WORKSPACE (USER)` *maxw*

where *maxw* is the number of spaces in the workspace array that you will allocate to the arrays needed for mixed integer programming. A good rule of thumb for selecting *maxw* is to set

$$maxw \geq 6n + 2m + 4l,$$

where n is the number of variables, m is the number of constraints, and l is the CYCLE LIMIT.

- *SUMMARY FILE.* If you set the SUMMARY FILE option to 0 in order to turn off the MINOS screen output, MIPMINOS will run much faster.

3.2.2 SPECIFYING LINEAR PROBLEM DATA (MPS File)

All variables, and the constraints and bounds associated with the variables, are specified in the MPS data file required by MINOS for a linear program.

3.2.3 SPECIFYING INTEGER VARIABLES

File MIPMINOS.MPS is used to specify which variables in the linear program described by the specifications file and the MPS data file are either integer-valued or are integer multiples of given increments. MIPMINOS.MPS uses a format similiar to that found in an MPS file; there is a header card, an end-of-data card, and the lines in-between must have the following data format:

Columns	5-12	25-36
Contents	Variable Name	Increment Value

In other words, the variable name is placed left justified in Columns 5-12 of the line, and the increment value is placed in Columns 25-36. MIPMINOS uses free format to read the increment value, so the increment value may be entered in any form you like. For example, 321.0, 3.21D+02 and 3210.0E-01 represent the same number. If the variable is to accept all integer values, set the increment to 1.0. NOTE: because of the way that MIPMINOS handles the increment values, the increments *must* always be positive.

It should be noted that if you specify that

$$l_i \leq x_i \leq u_i, \quad x_i = kd_i, \quad k \text{ integer}, \quad d_i > 0,$$

MIPMINOS will attempt to find a solution where

$$\hat{l}_i d_i \leq x_i \leq \hat{u}_i d_i,$$
$$\hat{l}_i \text{ the smallest integer larger than } l_i/d_i,$$
$$\hat{u}_i \text{ the largest integer smaller than } u_i/d_i.$$

For example, if we have a mixed integer program with variables named X01, X02, X03 and X04, X01 and X02 are integer-valued, and X04 is an integer multiple of 2.5, MIPMINOS.MPS would take the following form.

```
INTEGERS
    X01                 1.0
    X02                 1.0
    X04                 2.5
ENDATA
```

You could also specify X04 as having an increment of 1.0, and multiply all coefficients of X04 by 2.5; we, however, recommend the former approach for possibly better scaling of the constraint matrix.

3.2.4 MIPMINOS OUTPUT

The solution to the Mixed Integer Program is written to the PRINT file used by MINOS, if a solution has been found. If not all subproblems have been fathomed after the CYCLE LIMIT has been reached, the incumbent subproblem (i.e. the best solution found to date) is written to the PRINT file. If no incumbent has been found, or if the problem has no feasible solution, the solution to the last subproblem solved and an error message is written to the PRINT file.

3.3 AN EXAMPLE

An example problem is described in this section to illustrate how you would set up the MIPMINOS input files.

The following problem is adapted from a mixed integer program presented by Nemhauser & Wolsey [1988]. The specifications file, MPS data file, and MIPMINOS.MPS file are shown in Tables 3.1, 3.2, and 3.3 respectively; they are stored in the three files MIPEX.SPC, MIPEX.MPS, and MIPMIPEX.MPS on the IPMINOS diskette.

$$
\begin{array}{lrrrcr}
\text{Minimize} & -7x_1 & -2x_2 & & = & z \\
\text{Subject to} & -x_1 & +2x_2 & +16x_3 & = & 4 \\
& 5x_1 & +x_2 & & \leq & 20 \\
& -2x_1 & -2x_2 & & \leq & -7 \\
& x_1, & x_2, & x_3, & \geq & 0 \\
& x_1, & x_2 & \text{integer} & &
\end{array}
$$

The solution to the above problem is $x = (4, 0, 0.5)^T$, $z = -28$; it takes 5 iterations to fathom all of the subproblems using the MIPMINOS version of the branch-and-bound

```
Begin MIPEX         (Sample Mixed Integer Program)
   Minimize
   Rows                     20
   Columns                  30
   Elements                100
   Cycle Limit              70  * Limits # of B&B problems solved
   MPS file                 10

   Iterations              100
   Print level               1  * OK for small problems
   Print frequency           1
   Summary frequency         1
   Workspace (user)       1000  * Allocate workspace for B&B routines
End MIPEX
```

Table 3.1: Specification File (MIPMINOS Example)

algorithm. An additional iteration is required to verify that the optimal solution satisfies all constraints.

```
* This example of a Mixed Integer Program was adapted from the
* problem on Page 360 of "Integer and Combinatorial Optimization",
* by G.L. Nemhauser and L.A. Wolsey.
NAME          MIPEX
ROWS
 E  ROW1
 L  ROW2
 L  ROW3
 N  COST
COLUMNS
    COLUMN1   ROW1         -1.0
    COLUMN1   ROW2          5.0
    COLUMN1   ROW3         -2.0
    COLUMN1   COST         -7.0
    COLUMN2   ROW1          2.0
    COLUMN2   ROW2          1.0
    COLUMN2   ROW3         -2.0
    COLUMN2   COST         -2.0
    COLUMN3   ROW1         16.0
RHS
    DEMANDS   ROW1          4.0
    DEMANDS   ROW2         20.0
    DEMANDS   ROW3         -7.0
ENDATA
```

Table 3.2: MPS File (MIPMINOS Example)

```
INTEGERS
    COLUMN1                  1.0
    COLUMN2                  1.0
ENDATA
```

Table 3.3: Integer Specification File (MIPMINOS Example)

Appendix A

TECHNICAL DETAILS

This section provides a description of the technical details of BIPMINOS and MIPMINOS. We will discuss the branch-and-bound method used by both BIPMINOS and MIPMINOS, describe our implementation, and provide a list of error codes.

A.1 ALGORITHM DESCRIPTION

The integer programming algorithm implemented here is Dakin's version of the branch-and-bound method, as described by Geoffrion & Marsten [1972], and Hillier & Lieberman [1986]. Hillier & Lieberman offer an overview of the branch-and-bound technique; more extensive treatments of integer programming and variants of the branch-and-bound method can be found in Geoffrion & Marsten [1972] or in Nemhauser & Wolsey [1988].

Branch-and-bound algorithms solve a sequence of linear programs, each of which can be considered as a node on a binary tree. The general approach allows for some flexibility with respect to the order in which it solves the sequence. (You could, if desired, modify the MIPMINOS and BIPMINOS routines to solve the linear programs in a different and perhaps more efficient order.) A description of the branch-and-bound procedure used by BIPMINOS and MIPMINOS follows.

1. *Initialization:* Set Z^* to ∞ if minimizing ($-\infty$ if maximizing). Z^* will contain the objective value of the best integer solution found during the course of the algorithm. Remove the integer restrictions and solve the resulting linear program.

2. *Fathoming:* Apply the fathoming criterion to the most recently solved linear program. The linear program is considered fathomed if any of the following conditions hold.

   - The linear program objective value $\geq Z^*$ if minimizing ($\leq Z^*$ if maximizing). Because any subproblem generated by adding additional constraints to the linear

program cannot yield a better integer solution, these subproblems are no longer of interest to the algorithm.

- The linear program is infeasible. Because any subproblems generated by adding additional constraints will also be infeasible, these subproblems need not be considered.
- The linear program has a solution satisfying the integer restrictions. If the associated objective value is less than Z^* if minimizing (greater than Z^* if maximizing), update Z^* and set the incumbent integer solution to the current linear program solution. Otherwise, any subproblems generated by adding additional constraints cannot yield a better integer solution, so they need not be considered.

If the linear program is fathomed, do not add it to the branch and bound tree. Mark the current node's left or right child as fathomed and go to step 3. Otherwise, determine the first integer restricted variable in the linear program with a nonintegral value. Let x_j be that variable. Add a node with x_j as the branching variable to the branch-and-bound tree.

3. *Branching:* Let x_j^* be the value of the branching variable associated with the node currently under consideration, and let $\lfloor x_j^* \rfloor$ be the integer part of x_j^*.

   - If neither of the node's children has been fathomed, solve the linear program associated with the node's left child by removing all integer restrictions but adding the constraint $x_j \leq \lfloor x_j^* \rfloor$. (If we are solving a binary integer program, we set x_j to 0.) Go to step 2.
   - If the node's left child has been fathomed but the right child has not been fathomed, solve the linear program associated with the right child by removing all integer restrictions but adding the constraint $x_j \geq \lfloor x_j^* \rfloor + 1$. (If we are solving a binary integer program, we set x_j to 1.) Go to step 2.
   - If both children have been fathomed, the current node has also been fathomed.

4. *Optimality:* If the fathomed current node has no parent node, stop. Either the incumbent solution is the optimal solution, or there are no solutions which satisfy the integer restrictions. Otherwise, move from the current node to its parent, marking the current node as fathomed, and go to step 3.

A.2 IMPLEMENTATION DESCRIPTION

In this section, we describe the implementation of our branch-and-bound algorithm. Critical issues include memory allocation, subroutine design, and data structures.

A.2.1 MEMORY ALLOCATION

MINOS allocates memory by declaring one large double precision array Z and then setting pointers to different locations in Z for the various arrays needed to solve linear and nonlinear programs. Therefore, any memory needed for arrays to be used in the integer programming algorithm must be allocated for. The length of the array Z is set in the program found in MINOSL.FOR.

MINOS allows its users to reserve memory for their own purposes at the beginning of Z through the SPECS file option WORKSPACE (USER). (See the MINOS User's Guide for more details on the SPECS file.) For example, the MIPMINOS subroutine **IPCORE** (described in Section A.2.2) divides up this chunk of contiguous memory among the various arrays required to store the data required for the branch-and-bound algorithm.

A.2.2 SUBROUTINES

MIPMINOS and BIPMINOS only differ from standard MINOS in subroutine MATMOD. The purpose of MATMOD is to allow MINOS users to access the various MINOS data structures before MINOS solves a problem. Thus, MINOS is well-suited to solving the sequence of problems encountered when using the branch-and- bound method for integer programming. The MIPMINOS and BIPMINOS versions of MATMOD were written so that they call various other subroutines to perform the fathoming, branching and optimality test steps. MATMOD is called by the MINOS routine MINOS3, which is found in the source file MI10*.FOR; in the PC version of MINOS, for example the source file is MI10PC.FOR.

The file MIPFUNS.FOR includes the MIPMINOS version of MATMOD, as well as all of the subroutines used in our implementation of the branch-and-bound technique. The file BIPFUNS.FOR includes the BIPMINOS version of MATMOD, as well as all of the subroutines used in our implementation of the branch-and-bound technique. A summary of each routine follows.

- **MATMOD** interfaces between standard MINOS and the integer programming subroutines.
- **IPITER** is the main integer programming routine. It calls other routines to perform the fathoming, branching and optimality tests described in section A.1.
- **IPINPT** and **GETINP** are used to read the input file MIPMINOS.MPS (BIPMINOS.MPS) for the names of the integer variables in the problem, and, in the case of MIPMINOS, the values of their associated increments.
- **IPINIT** performs the initialization step described in section A.1.

- **IPCORE** allocates $Z(1, ..., MAXW)$, the portion of Z reserved for the MINOS user, among the various arrays required for the branch-and-bound algorithm.
- **CHKSOL** examines a linear programming solution and determines whether the integer restricted variables are all integer multiples of their increments. If not, **CHKSOL** returns the index of the first variable that is not an integer multiple of its increment.
- **PRUNE** removes a node from the branch-and-bound tree after the associated subproblem has been fathomed.
- **FATHOM** applies the fathoming criteria to a linear program solution.
- **INCUMB** updates the incumbent solution.
- **GETSOL** reads the incumbent solution back into MINOS once optimality has been reached.
- **PUSH** adds a node to the branch-and-bound tree.

A.2.3 DATA STRUCTURES

The choice of data structures used to implement an algorithm can profoundly affect its reliability and efficiency. MIPMINOS and BIPMINOS require a few fixed length arrays which require very little memory. However, they also require a data structure for the branch-and-bound tree. The tree can grow rapidly as nodes are added, so the selection of a data structure to efficiently process the nodes is vital. Note that our branch-and-bound implementation as described in Section A.1 always processes the left child of a node before the right child. By processing the left child first, we can use a stack data structure to represent the tree. This would not be true if the algorithm could consider either the left child or the right child first; in this case a more intricate data structure would be require. After **IPCORE** allocates the fixed length arrays, the rest of $Z(1, ..., MAXW)$ is used for the stack, where each item in the stack corresponds to a node in the branch-and-bound tree. Each node contains the branching variable index, the branching value (in the MIPMINOS version), flags indicating whether the left and right children have been fathomed, and the values of the branching variable's bounds.

MINOS allows the user to modify any of its own data structures by calling MATMOD, and will solve whatever problem resides in those structures after MATMOD is finished. In the case of a branch-and-bound algorithm, the only parts of the problem that require a changed are the lower and upper bound arrays BL and BU; no other MINOS arrays need be altered.

A.3 ERROR CODES

Unmodified MINOS has many different exit conditions. The variable $IERR$ contains the exit condition upon completion of a run. Exit conditions include optimality ($IERR = 0$), infeasibility ($IERR = 1$), excessive iterations ($IERR = 3$), basis factorization errors ($IERR = 12$) and many others. (Refer to the MINOS User's Guide for a complete listing of exit conditions.) Some additional error conditions specific to MIPMINOS and BIPMINOS have been added.

- $IERR = -4$. The integer program is infeasible, although the linear programming relaxation has a solution.
- $IERR = -5$. There is insufficient memory for integer programming algorithm. You will have to specify a larger value of user workspace in the SPECS file. It may also be necessary to recompile MINOS with a larger Z (workspace) array.
- $IERR = -6$. There is an error in the integer programming input procedure. This means that there is an error in MIPMINOS.MPS or BIPMINOS.MPS, the files containing the names of the integer variables, and, in the case of MIPMINOS, the increment values. Check to ensure that the MIPMINOS.MPS or BIPMINOS.MPS file specifies the integer variables using the format discussed in Sections 3.2.3 and 2.2.3. If this fails, you should make sure that the hard disk of the computer running the software is in good working order.
- $IERR = -7$. There is a stack overflow during the integer programming algorithm. The branch-and-bound tree has grown too large to be stored in the available memory. Proceed in the same manner as if $IERR = -5$.
- $IERR = -8$. There is an error in the integer programming algorithm; the incumbent solution is not integer-valued. There is a bug in the integer programming algorithm; contact Stanford Business Software.
- $IERR = -9$. The CYCLE LIMIT (the number of problems that MINOS will solve) has been reached, not all subproblems have been fathomed, and no feasible solution has been found. Increase the CYCLE LIMIT in the SPECS file.
- $IERR = -10$. The CYCLE LIMIT has been reached, not all subproblems have been fathomed, but a feasible solution has been found. An error message will be written to the PRINT and SUMMARY files, and the incumbent solution will be reported as the best solution found. You may wish to rerun the problem with an increased CYCLE LIMIT in the SPECS file to confirm optimality.

REFERENCES

Dantzig, G.B. (1963)."Linear Programming and Extensions", *Princeton University Press*, Princeton, New Jersey.

Dantzig, G.B. and Thapa M.N. (1997). "Linear Programming 1: Introduction", *Springer-Verlag*, Berlin and New York.

Dantzig, G.B. and Thapa M.N. (1998a). "Linear Programming 2: Theory and Implementation", To be published by *Springer-Verlag*, Berlin and New York.

Dantzig, G.B. and Thapa M.N. (1998b). "Linear Programming 3: Extensions", To be published by *Springer-Verlag*, Berlin and New York.

Geoffrion, A.M., and Marsten, R.E. (1972). " Integer Programming Algorithms: A Framework and State-of-the-Art Survey", *Management Science*, Volume 18, pages 465-491.

Gill, P.E., Murray, W., and Wright, M.H. (1981). "Practical Optimization", *Academic Press*, London and New York.

Hillier F.S., and Lieberman, G.J. (1986). "Introduction to Operations Research", *Holden-Day*, Oakland, California.

Lahey Computer Systems, Inc. (1991). "F77L-EM/32 Fortran Reference Manual", *Lahey Computer Systems, Inc.*, Incline Village, Nevada.

Murtagh, B.A. and Saunders, M.A. (1993). "MINOS 5.4 User's Guide", *Report SOL 83-20R*, Department of Operations Research, Stanford University.

Nemhauser, G.L. and Wolsey, L.A. (1988). "Integer and Combinatorial Optimization," *John WIley & Sons*, New York.