USER'S MANUAL FOR: ZOOM / XMP

ZOOM = Zero/One Optimization Methods

- - - -

RELEASE 4.0

Roy Marsten

School of Industrial and Systems Engineering

Georgia Institute of Technology

Atlanta, GA 30332

USA

Phone: (404) 894-3983

Jaya Singhal

Department of Information and Quantitative Sciences

University of Baltimore

Baltimore, MD 21201

see: "Fixed Order Branch-and-Bound Methods for

Mixed-Integer Programming: The ZOOM System"

Jaya Singhal, Roy Marsten, and Thomas Morin,

ORSA JOURNAL ON COMPUTING, Vol. 1, No. 1,

Winter 1989, pp. 44-51.

CONTENTS

--------

CHAPTER 1. INTRODUCTION

CHAPTER 2. METHODOLOGY

A. Initial Linear Program

B. Pivot & Complement Heuristic

C. Branch-and-Bound

D. Fixed Order Branching

E. Multi-level Expansion

F. Diving vs. Best Bound

G. The Resource Space Tour

H. Locking Zero/One Variables

I. The Algorithm

J. Cheating

CHAPTER 3. MODEL INPUT

A. The SPECS file

B. The MPS file

CHAPTER 4. SAMPLE MODEL

A. The SPECS file

B. The XML file

C. The MPS file

D. The Output

CHAPTER 5. "EXPERT" ADVICE

REFERENCES

CHAPTER 1. INTRODUCTION

------------

ZOOM is a computer code for solving zero/one mixed-integer

programming problems. It is intended for medium-sized problems

with no special structure and up to about 200 zero/one variables.

The mathematical form of the problem is:

(MIP) maximize c(1)\*x(1) + ... + c(N)\*x(N)

subject to

a(1,1)\*x(1) + ... + a(1,N)\*x(N) <= b(1)

. . .

. . .

. . .

a(M,1)\*x(1) + ... + a(M,N)\*x(N) <= b(M)

L(j) <= x(j) <= U(j) for j=1,...,N

L(j)=0, U(j)=1, and x(j) integer for j in BINARY.

(In the Ax <= b constraints, the "<=" may be replaced by ">="

or "=", as necessary.)

A problem with no zero/one variables is a pure linear program;

a problem with all zero/one variables is a pure integer program;

these are both special cases of the mixed-integer program.

This manual is written with the mixed-integer problem in mind,

but it is both easy and efficient to use ZOOM to solve pure

linear programs as well. There is no extra work done or storage

allocated when the problem is a pure LP. ZOOM can solve pure

linear programs with many thousands of variables and equations.

This manual assumes some familiarity with linear and integer

programming, in particular the simplex method with bounded

variables and the branch-and-bound method.

The data for a given problem consists of the a(i,j), b(i),

c(j), L(j), and U(j). This data is placed in an MPS file

for input to ZOOM (see 3 B.). General information about the

problem and choices for the algorithmic parameters and options

are placed in a SPECS file (see 3 A.). Alternatively, ZOOM can

be used with the GAMS modelling language [9], or the XML modeling

language [10].

ZOOM is based on the XMP linear programming library [1],

which in turn uses the LA05 basis factorization routines [2].

ZOOM begins by solving the problem as a linear program, and

then uses the Pivot & Complement heuristic [3] to find an initial

integer feasible solution. It then uses a branch-and-bound

search to find improved solutions and to verify optimality.

The B&B procedure uses linear programming to compute bounds.

It also has several novel features not found in other MIP codes.

These are: fixed order branching, multi-level expansion,

the resource space tour, and "cheating". These are

explained below.

The algorithmic ideas that are incorporated in ZOOM have the

following origins. The Pivot & Complement heuristic is from

Balas and Martin [3]. The fixed-order branching and the

resource space tour idea are from Marsten and Morin [4].

The multi-level expansion idea is from Marsten and Singhal [5]

and Singhal [6]. The "cheating" idea is from Morin and

Marsten [7,8].

Only zero/one integer variables are allowed. Integer variables

with upper bounds larger than one can be converted to zero/one

variables. For example if x(j) can take on values 0,1,...,U(j),

then we can define K new zero/one variables, where K is chosen

so that:

2\*\*(K-1) <= U(j) <= 2\*\*K

Then the integer variable x(j) can be represented as:

x(j) = y(j,1) + 2\*y(j,2) + 4\*y(j,3) + 8\*y(j,4) + ...

+ 2\*\*(K-2)\*y(j,K-1) + (U(j)-2\*\*(K-1)+1)\*y(j,K)

where each y(j,k) variable is a zero/one variable.

ZOOM is not intended for very large specially structured

problems. These generally require special algorithms which

exploit problem structure.

Throughout these notes we shall assume that we are

maximizing, in order to simplify the discussion.

CHAPTER 2. METHODOLOGY

-----------

2 A. Initial Linear Program

----------------------

The first step in solving a zero/one mixed-integer program is

solving the linear programming relaxation. This is obtained

by relaxing the x(j) = 0 or 1 conditions to 0 <= x(j) <= 1.

ZOOM solves this LP relaxation by calling XMP. If the LP

relaxation is infeasible, then so is the original mixed-integer

program, and we are finished. If the LP relaxation has a natural

integer optimal solution, then this solution is also optimal

for the original mixed-integer program, and we are finished.

Notice that this is the case if all of the zero/one variables

are non-basic (or basic and degenerate) in the LP solution.

If neither of the above events occurs, we still get a bound

on the optimal value of the mixed-integer program. Let v(LP)

denote the optimal value of the LP relaxation, and let v(MIP)

denote the optimal value of the mixed-integer program.

Then v(LP) >= v(MIP), assuming maximization.

ZOOM will invoke the primal simplex method unless the user

specifies the dual simplex method in the SPECS file (see 3 A.).

Many integer programming models have extremely degenerate LP

relaxations (e.g. set covering problems). The dual simplex

method is generally more effective on these problems.

If the dual simplex method is chosen for an integer program,

then it is used during the heuristic and the branch-and-bound

search as well as for the initial LP.

2 B. Pivot & Complement Heuristic

----------------------------

We will not discuss the Pivot & Complement heuristic in detail

here, since the reader can consult [3]. Some obvious

modifications have been made to the procedure of [3] to extend

the method from pure integer to general mixed-integer problems.

In general terms, the method searches in the vicinity of the

optimal solution of the LP relaxation for an integer feasible

solution. It conducts this search by trying to force any

basic integer variable (which is necessarily fractional unless

it is degenerate) out of the basis. This is done by performing

pivots. Non-basic integer variables can also be complemented,

i.e. flipped from zero to one or one to zero. Both one-at-a-

time and two-at-a-time complements are attempted. If a feasible

solution is found, then additional complements are performed in

an attempt to improve it.

This Pivot & Complement heuristic has worked very well in our

experience to date, and it is quite fast (the pairwise complements

are not done on more than 500 zero/one variables).

Good solutions can be found for many large mixed-integer problems

by doing just the initial linear program and the P&C heuristic.

Doing branch-and-bound generally takes far longer than the

initial LP + P&C heuristic combination.

If the P&C heuristic finds a feasible integer solution with

value INCUMBENT, and if the relative gap between INCUMBENT and

v(LP) is less than or equal to a user specified TOLERANCE:

(v(LP)-INCUMBENT) / (1+abs(v(LP))) <= TOLERANCE

then we are finished. TOLERANCE is specified in the SPECS file.

2 C. Branch-and-Bound

----------------

If the LP relaxation is feasible but does not have a natural

integer solution, and if the heuristic does not find an integer

feasible solution that satisfies the TOLERANCE, then we have to

resort to branch-and-bound.

B&B can be described in general terms in the following way.

Assume that we are maximizing, and that INCUMBENT is the

objective function value of the best integer feasible solution

that has been found so far. We begin by creating a search tree

consisting of a single node that represents the original problem,

(MIP). This node is the permanent root of the tree and is

temporarily a leaf as well. The value of the LP relaxation,

v(LP), is attached to this node as a label. One "iteration"

of the branch-and-bound method is to:

(BB1) select some leaf node in the current search tree.

This node represents a problem which we denote (CP)

for "candidate problem". Initially, (CP)=(MIP).

(If the search tree is empty, we are done.)

(BB2) choose a zero/one variable, say x(k), and split (CP)

into two new problems (NP1) and (NP2) by appending the

mutually exclusive constraints: x(k)=0 and x(k)=1.

Add two new leaf nodes to the tree for (NP1) and (NP2)

(see figure below). The node for (CP) is no longer a leaf.

(BB3) Solve the LP relaxation (LP1) of (NP1). Eliminate the

node for (NP1) if (LP1) is infeasible, or if

v(LP1) <= INCUMBENT, or if (LP1) has a natural integer

solution. In the latter case, update the value of the

INCUMBENT. If the (NP1) node survives, then attach

v(LP1) to it as a label.

Do the same for (NP2). If (NP1) and (NP2) are both

eliminated, then (CP) may be eliminated as well.

--------

l l

l (CP) l

l l

--------

. .

. .

x(k)=0 . . x(k)=1

. .

. .

--------- ---------

l l l l

l (NP1) l l (NP2) l

l l l l

--------- ---------

This cycle (select, split, solve) is then repeated.

The B&B search is finished when the search tree is empty,

or the INCUMBENT value satisfies the TOLERANCE, whichever

comes first. The largest LP-value label over all leaves in

the current tree is the best available upper bound on v(MIP),

and we call this bound GLOBAL.

INCUMBENT <= v(MIP) <= GLOBAL.

Initially, GLOBAL = v(LP). The search is halted if

(GLOBAL-INCUMBENT) / (1+abs(GLOBAL)) <= TOLERANCE.

If the search tree is empty, then the INCUMBENT value is

guaranteed to be optimal. (Assuming that we really have a

solution with value INCUMBENT and are not bluffing.)

Thus B&B is a race between creating nodes at step (BB2) and

eliminating them at step (BB3).

2 D. Fixed Order Branching

---------------------

In the general B&B procedure described above, we can use any

zero/one variable to split node (CP) into two new nodes.

(More precisely, any zero/one variable that has not already

been fixed on the path from the root of the tree to the node

for (CP).) In ZOOM we restrict this freedom of choice.

Any node in the search tree has a "level", which is the number

of arcs on the path from that node up to the root of the tree.

(Or the number of zero/one variables already fixed in the

problem associated with that node.) We shall designate a

specific zero/one variable for each level in the tree.

For level d, we shall denote the index of this variable as

SORT(d). Then whenever any node at level d is split into

two new nodes, this will be done with the constraints:

x(SORT(d+1))=0 and x(SORT(d+1))=1.

We shall also use "depth" to mean "level".

The reason for imposing a fixed branching order will be

explained in section (3 G.) below. The fixed order is

determined in the following way. Consider the LP relaxation

of the original problem (MIP). The optimal LP solution

partitions the zero/one variables into three groups:

basic (BS), non-basic at upper bound (UB), and non-basic at

lower bound (LB). The BS variables are sorted according

to their values, the UB variables are sorted according to

their relative profits, and the LB variables are sorted

according to the negatives of their relative profits.

The relative profit (or reduced cost) for variable x(j) is:

d(j) = c(j) - u(1)\*a(1,j) -...- u(M)\*a(M,j)

where the u(1),...,u(M) are the shadow prices. Then if ORDER=1

in the SPECS file, the branching order is taken as first

the BS variables as sorted, then the UB variables as sorted,

and finally the LB variables as sorted. If ORDER=2, then

the UB variables are placed before the BS variables.

If ORDER=3, then the user will specify his own branching order.

2 E. Multi-level Expansion

---------------------

In the B&B procedure described in (3 C.) above, we always

select one node and split it into two nodes at the next level.

In ZOOM we employ a more general strategy. We may

select several nodes which are at the same level in the

current tree and split them several times to create new

nodes which are several levels below the selected nodes.

The user may specify two parameters, SELECT and EXPAND in the

SPECS file. ZOOM will then take SELECT nodes which are at

the same level, say level d, and split them EXPAND times

to create SELECT\*(2\*\*EXPAND) nodes at level d+EXPAND.

For example, the default values are SELECT=2 and EXPAND=3.

This causes 2 nodes at some level d to be expanded into 16

nodes at level d+3. The standard strategy is obtained by

setting SELECT=1 and EXPAND=1.

We have found empirically that it is advantageous to use

values greater than 1 for SELECT and EXPAND in an algorithm

based on fixed order branching (3 D.) and resource space

tours (3 G.).

The question of what level the SELECT nodes should be taken

from is addressed in section (2 F.). As to which nodes are

actually selected, we always take the ones with the SELECT

best LP-value labels. (If there are fewer than SELECT nodes

then we take all of them.)

2 F. Diving vs. Best Bound

---------------------

The first step in each B&B iteration is deciding on the level

in the tree from which the nodes to be expanded are to be

selected. ZOOM permits two alternative strategies: diving

and best bound. In the diving strategy we always choose the

deepest level in the current tree. This tends to produce natural

integer solutions as soon as possible, which is useful if the

INCUMBENT value is far from the optimum.

In the best bound strategy, we always choose the level that has

the leaf node with the largest LP-value label in the current tree.

That is, the leaf node with LP-value = GLOBAL. This tends to

reduce GLOBAL as fast as possible. This is useful when

we have an optimal or near-optimal solution and are trying to

satisfy the TOLERANCE.

The choice of search strategy, diving vs. best bound, is made in

the SPECS file. If ZOOM is diving and it finds an improved integer

feasible solution (i.e. new incumbent), it will automatically

switch to the best bound strategy. If you are running ZOOM

interactively under IMP, then IMP will ask you to choose between

diving and best bound after you have seen the outcome of the

heuristic.

2 G. The Resource Space Tour

-----------------------

We will now explain the motivation for the fixed order branching.

This section requires a knowledge of LP duality theory and

parametric linear programming.

Suppose that SELECT=2 and EXPAND=3. Then we will select the 2

best nodes from some level d and expand them into 16 nodes at

level d+3. These 16 nodes have 16 associated linear programs:

(LP1),..., (LP16). We are going apply bounding tests to these

16 nodes and eliminate as many as we can. The bounding tests

will be based on LP generated information, but we will not be

solving the 16 LP's completely separately from one another.

We will be making a "resource space tour".

Because of the fixed order branching, the 16 linear programs

are identical except for their right-hand-sides. They all

have the same set of fixed zero/one variables: SORT(1),...,

SORT(d+2). The fixed variables are moved from the left-hand-

side of the constraints to the right-hand-side, giving 16

different reduced right-hand-side vectors.

We imagine the 16 nodes as represented by 16 points in the M-

dimensional space of right-hand-side vectors. Starting at

the point for (LP1), we shall make a tour that visits some or

all of these 16 points. We begin by solving (LP1). Then we

use parametric programming to obtain the solution of (LP2).

The 16 linear programs have the same dual feasible region,

hence any dual feasible solution can be used to compute an

upper bound on v(LPq) for every q, and hence to perform a

bounding test on all 16 nodes. Since we are doing parametric

programming on the right-hand-side, every pivot produces a new

dual feasible solution that can be used to perform a bounding

test on all of the still surviving nodes.

If we are travelling, by parametric programming, from the

point for (LPq) to the point for (LPr) and we succeed in

eliminating some other point, say (LPs) (q<r<s), we call it

an "indirect hit". If we eliminate the current destination,

(LPr), we call it a "direct hit". If (LPr) is eliminated,

we immediately change direction and head toward the next still

surviving point.

If (LPr) survives, then we check to see if its LP solution is

naturally integer and then strike off for the next still

surviving node (which will be (LPr+1) if it still exists).

At the end of the tour we have eliminated some of the nodes

(points) and obtained the LP solutions of the others ( some of

which may be naturally integer and hence new incumbents).

For a more complete description, see [4].

We have discovered empirically that with fixed order branching

and the resource space tour, the standard SELECT=1, EXPAND=1

strategy gives poor results. It is much more effective to

spread the LP computational overhead among more nodes and take

advantage of indirect hits. In one series of experiments [6],

the best strategy was SELECT=5, EXPAND=4, giving 80 nodes for

each tour.

If EXPAND=3, then we will only be performing tours at levels

3,6,9,... in the search tree. ZOOM will save one "hot" LP basis

for each of these levels. A tour at level 9 will start with

the hot basis for level 9 if one is available, otherwise with

the hot basis for level 6, or level 3, or level 0 (root) in

that order of preference. The "MAX SAVE" option in the SPECS

file allows the user to decide how many hot bases should be

saved. If "MAX SAVE 5" is specified, then the root basis and

four other hot bases will be held at any one time.

Saving more hot bases should speed up the tours, but each

one saved takes up COLUMNS+2\*ROWS memory locations.

2 H. Locking Zero/One Variables

--------------------------

The user may lock zero/one variables by using the FX code in the

BOUNDS section of the MPS file (see 3 B.)

ZOOM will lock zero/one variables automatically, based on the

solution to the initial LP relaxation. After solving the initial

LP we compute the relative profits d(j) for each variable x(j),

j in BINARY. Assuming a maximization problem, if x(j) is at its

upper bound (UB) and

d(j) >= v(LP) - INCUMBENT

then x(j) can be locked permanently at one. On the other hand,

if x(j) is at its lower bound (LB) and

-d(j) >= v(LP) - INCUMBENT

then x(j) can be locked permanently at zero. The d(j) and v(LP)

values are saved and whenever a new INCUMBENT value is found during

the heuristic or the branch-and-bound search we attempt to lock

additional variables.

2 I. The Algorithm

-------------

We summarize the above discussion in a concise statement of the

main steps in the algorithm. Capitalized names refer to options

and parameters from the SPECS file or to program variables.

We assume maximization, and an initial INCUMBENT value (see the

INCUMBENT and GAP keywords in 3 A.).

Step 1. Solve the initial linear program.

If it is infeasible, stop. If it has a natural integer

solution, stop. Otherwise initialize the search

tree by creating a node with LP-value label = v(LP).

(If the problem is a pure LP, stop.)

Step2. Compute the relative profits, d(j), of all the zero/one

variables and lock as many of them as possible using

v(LP) and INCUMBENT.

Step 3. If HEURISTIC='YES', then use the Pivot & Complement

heuristic to attempt to find an integer feasible

solution.

If P&C finds an improved INCUMBENT, try to lock

additional variables.

Step 4. If INCUMBENT satisfies the TOLERANCE, stop.

(v(LP)-INCUMBENT) / (1+abs(v(LP))) <= TOLERANCE

Step 5. If BRANCH='NO', stop. Otherwise use ORDER to determine

the branching order. Set IMPROV=.FALSE..

Step 6. If the search tree is empty, stop. The INCUMBENT

value is optimal (unless it is a bluff).

Step 7. Set GLOBAL = the largest LP-value label over all leaves

in the current tree. Stop if

(GLOBAL-INCUMBENT) / (1+abs(GLOBAL)) <= TOLERANCE.

Step 8. Stop if IMPROV=.TRUE. and QUIT='YES'.

Step 9. Stop if the LIMIT on LP iterations has been exceeded.

Step 10. If DIVE='YES', choose d = the deepest level in the

current tree. Otherwise, choose d = the level that

has the leaf with LP-value label = GLOBAL.

Step 11. Take the SELECT nodes at level d that have the best

LP-value labels, and expand them into

SELECT\*(2\*\*EXPAND) nodes at level d+EXPAND.

If there are fewer than SELECT nodes at level d,

then just take all of them.

Step 12. Do a resource space tour at level d+EXPAND.

If a natural integer solution is found that gives an

improved INCUMBENT value, then set IMPROV=.TRUE..

Keep track of all LP iterations for comparison with

LIMIT. Label each surviving node with its LP-value.

Step 13. If IMPROV=.TRUE., purge the tree. That is, eliminate

any leaf that has LP-value label <= INCUMBENT.

Step 14. If IMPROV=.TRUE., use the saved d(j) and v(LP) values

to attempt to lock additional variables.

Step 15. Go to Step 6.

2 J. Cheating

--------

In this section we examine the bounding test that is used

in the branch-and-bound procedure. Consider any node in the

search tree. Suppose it represents sub-problem (CPq) with LP

relaxation (LPq). We may eliminate (discard) this node if

UB(q) <= INCUMBENT

where UB(q) is an upper bound on v(CPq), the optimal value of

(CPq). UB(q) may be v(LPq), or it may be an upper bound on

v(LPq) computed using a dual feasible solution of (LPq) during

a resource space tour. It may happen that

v(CPq) <= INCUMBENT <= UB(q)

i.e. (CPq) does not contain any solution better than the current

incumbent, but we cannot discard it since UB(q) is too loose an

estimate of v(CPq). (Remember that v(CPq) is unknown.)

It may also happen that

INCUMBENT < UB(q) < v(MIP)

i.e. (CPq) does not contain an optimal solution of (MIP),

but we cannot discard it because it may contain a solution

that is better than the current (poor) incumbent.

Because of the error involved in making the LP relaxation,

UB(q) is too big. And INCUMBENT, because it is not, in general,

optimal, is too small. We would like to decrease UB(q) and

increase INCUMBENT to correct for these errors. Of course the

amount of correction needed is unknown, but the form of the

correction would be:

UB(q)\*(1-A) <= INCUMBENT\*(1+B)

where 0<=A<1 and B>0. This is equivalent to:

UB(q) <= INCUMBENT\*(1+EPS).

If EPS > 0.0, then we may discard nodes that would not normally

(EPS = 0.0) be discarded. Clearly if EPS is large, we will

eliminate too many nodes and almost surely miss the optimal

solution. We can, however, determine the maximum possible error

in our final answer.

During the B&B search we simply keep track of the largest UB(q)

value over all discarded nodes. That is, the best bound that

gets thrown away. We call this value DISCARD. During the

search, we modify the definition of GLOBAL to be either DISCARD

or the largest LP-value label over all the leaves in the tree,

whichever is larger. At the end of the search,

GLOBAL = max [DISCARD, INCUMBENT].

Thus the incumbent is optimal if DISCARD <= INCUMBENT, and

otherwise the relative error is no greater than

(GLOBAL-INCUMBENT)/GLOBAL.

So we can determine the consequences of cheating (i.e. using

EPS > 0.0) simply by keeping track of the single number DISCARD.

In deciding on the numerical value to use for EPS, our first

observation is that EPS should be smaller when we are deep in

the tree than when we are near the top of the tree. We would

also like EPS to depend on the apparent relative gap between the

MIP and LP optimal values for the problem at hand. After

considerable experimentation that is described in [6], ZOOM

uses the following method for computing EPS values when

CHEAT='YES' is specified in the SPECS file.

Let NBR be the number of zero/one variables that have to be

branched on, i.e. the original number minus

any that are locked. Let SHRINK and EXPAND be as given in the

SPECS file, or by default. Suppose we are performing a resource

space tour at level d+EXPAND. Then we use the bounding test:

UB(q) <= INCUMBENT\*(1+EPS)

where

EPS = gap(d)\*(1 - (SHRINK+EXPAND-1)/(NBR-d))

if SHRINK+EXPAND-1 < NBR-d; or

= 0.0 otherwise

and

gap(d) = (BESTUB(d)-INCUMBENT)/INCUMBENT

and

BESTUB(d) = the largest LP-value label obtained so far

at depth d.

Thus we cheat more if the estimate gap(d) of the relaxation error

at depth d is large. The SHRINK parameter is required to be at

least 1.0, and the larger it is, the less we are cheating.

If CHEAT='YES', then the EPS value will be printed out at the

beginning of each resource space tour (and again during the

tour if a new INCUMBENT value causes EPS to be revised).

The value of DISCARD will also be printed.

CHAPTER 3. MODEL INPUT

-----------

3 A. The SPECS file

--------------

The SPECS file contains general information about the problem

to be solved, and the user's choices for the many algorithmic

parameters and options.

The first line of the SPECS file must contain the keyword

'BEGIN', and the last line must contain the keyword 'END'.

Every line between the 'BEGIN' and 'END' must either be a

comment line (first non-blank character '\*' or '!') or contain

one of the keywords listed below as its first non-blank characters.

For each of the valid keywords there may be a modifier and/or

a value. The lines are free form in that there may be any

number of blanks or '=' signs separating items. Every line

must look like:

keyword

or : keyword modifier

or : keyword value

or : keyword modifier value

Any line may end with a comment whose first character is '\*' or '!'.

The ordering of the lines (i.e. of the keywords) is not important.

Only the first 3 characters of each keyword are significant.

Only the first 4 characters of each modifier are significant.

All of the keywords have default values that are intended to

reduce the number of values that the user has to specify.

Numeric values for the keywords may consist of up to 8 characters,

the first of which must be numeric. The number may be either

an integer or a real constant. A number will be treated as

real if any character other than the first is '.', 'E', or 'D'.

Real numbers may be expressed in F, E, or D FORTRAN formats.

Any other numbers will be treated as integers.

(Note that .001 is invalid, but 0.001 is valid.)

EXAMPLES

To maximize a linear program with 86 constraints and 91 variables,

whose objective function is called 'PROFIT', we would use:

BEGIN

ROWS 86

COLUMNS 91

OBJECTIVE PROFIT

END

In this case we would be accepting several default values,

for example re-factoring the basis every 50 iterations.

The values specified as ROWS and COLUMNS have to be upper limits

on the true values, which are determined from the MPS file.

Therefore, to use the dual simplex method on the same problem,

assuming that all variables have lower bound zero and upper

bound +infinity, we could use:

BEGIN

ROWS 100

COLUMNS 100

DUAL YES

BOUNDS NONE

OBJECTIVE PROFIT

END

To solve a mixed-integer problem with no more than 20 zero/

one variables, using the heuristic, branch-and-bound

with the best bound strategy, and a stopping tolerance

of .005, we could use:

BEGIN

ROWS 100

COLUMNS 100

OBJECTIVE PROFIT

! THE REST OF THE LINES ARE FOR ZERO/ONE PART

INTEGER 20

HEURISTIC YES

BRANCH YES

DIVE NO

TOLERANCE 0.005

END

The following is the list of valid keywords with their associated

modifiers, values, and default values.

KEYWORD: BRANCH VALUES: 'YES', 'NO' DEFAULT: 'NO'

Specifies whether or not a branch-and-bound search is to be

performed.

KEYWORD: BOUNDS VALUES: 'BOTH', 'UPPER', 'NONE'

DEFAULT: 'BOTH'

'BOTH' means that both upper and lower bounds will be found

in the BOUNDS section of the MPS file. 'UPPER' means that

all variables (except free variables) have a lower bound of

zero, so only upper bounds will be present in the MPS file.

'NONE' means that all variables (except free and artificial

variables) have lower bound zero and upper bound +infinity.

Depending on the option specified, ZOOM will allocate

2\*COLUMNS, COLUMNS, or zero words of memory for the bounds.

Note: you must use 'BOTH' if there are any integer variables.

KEYWORD: CHEAT VALUES: 'YES', 'NO' DEFAULT: 'NO'

Specifies whether or not the cheating strategy is to be used

during the branch-and-bound search.

KEYWORD: COEFFICIENTS see: ELEMENTS

KEYWORD: COLUMNS VALUES: positive integer DEFAULT: 400

SYNONYM: VARIABLES

An upper limit on the number of structural columns in the model

to be solved (do not count slacks). It is best to use the exact

number if this is available (it is not obvious in the MPS file),

since it is used to allocate working storage space.

KEYWORD: CONSTRAINTS see: ROWS

KEYWORD: DIVE VALUES: 'YES', 'NO' DEFAULT: 'NO'

Specifies whether or not to use the diving strategy in the

branch-and-bound search. Diving is recommended as a first

attack on the problem, especially if the heuristic fails or

finds a relatively poor solution. 'NO' means use the best-bound

strategy. In the diving strategy, we always select nodes from

the deepest level in the current search tree. In the best-bound

strategy, we choose the level in the tree that has the best

associated LP bound. If ZOOM is diving and it finds an improved

integer feasible solution (i.e. new incumbent), it will

automatically switch over to the best bound strategy. If you are

running ZOOM interactively under IMP, then IMP will ask you to

choose between diving and best bound after you have seen the

outcome of the heuristic. IMP will also give you a chance to

change your choice each time it stops because of LIMIT.

KEYWORD: DUAL VALUES: 'YES', 'NO' DEFAULT: 'NO'

Use 'YES' if you want to use the dual simplex method,

otherwise the primal simplex method will be used.

Some integer programming problems have extremely degenerate

linear programming relaxations (e.q. set covering problems).

In these cases the dual simplex method is more effective.

KEYWORD: ELEMENTS VALUES: positive integer

DEFAULT: 5\*COLUMNS+ROWS

An upper limit on the number of non-zero matrix coefficients

in the model to be solved. Do not count non-zeros in the

objective function or right-hand-side. Do count one extra

non-zero for each row.

KEYWORD: EXPAND VALUES: 1,...,6 DEFAULT: 3

During the branch-and-bound search, each node that is selected

is expanded EXPAND levels down in the search tree, resulting

in 2\*\*EXPAND new nodes.

KEYWORD: FACTOR VALUES: positive integer DEFAULT: 50

SYNONYM: INVERT

The number of iterations to be allowed between re-factorizations

of the basis matrix. A value of 50 is suitable for most problems.

Should be reduced in the presence of serious numerical

instability. Should not be greater than 100. The accuracy of

the current solution is checked every FACTOR/2 iterations and an

early re-factorization done if necessary.

KEYWORD: GAP VALUES: non-negative real DEFAULT: +infinity

An estimate of the relative gap between the value of the integer

program, v(MIP), and the value of the linear program, v(LP).

I.e. GAP is an apriori estimate of

(v(LP)-v(MIP)) / (1+abs(v(LP))).

After computing v(LP), the code will compute an estimate of

v(MIP) using GAP. It will use this as the incumbent value

(see KEYWORD: INCUMBENT) at the beginning of the branch-and-

bound search (unless a better incumbent value has been found

by the heuristic). If the true gap is larger than GAP, and the

corresponding estimate of v(MIP) is not improved by the heuristic,

then the branch-and-bound search will not find any solutions.

Guessing a GAP can save a lot of work in the early part of

the search, until a true incumbent solution is found. It is a

bluff, however, and fails if it is too optimistic (too small).

A value specified with the INCUMBENT keyword will always override

the value of GAP. The default of +infinity has the effect of

not using the GAP feature at all. Suggested value: 0.25.

KEYWORD: HEURISTIC VALUES: 'YES', 'NO' DEFAULT: 'YES'

Specifies whether or not the Pivot and Complement heuristic

is to be used. Automatically re-set to 'NO' for pure linear

programs.

KEYWORD: INCUMBENT VALUES: real

DEFAULT: -infinity for maximization; +infinity for minimization.

The objective function value of the best integer feasible solution

that is known so far. This is used to eliminate nodes in the

search tree whose LP values are not better than INCUMBENT.

You can also bluff and hope that the search turns up a solution

with a better value than INCUMBENT. If you bluff, however,

and the value you give for INCUMBENT is better than the true

optimal value, then the branch-and-bound search will not find

any solutions. A value specified with the INCUMBENT keyword

will always override the GAP value. (see KEYWORD: GAP)

KEYWORD: INTEGER VALUES: non-negative integer DEFAULT: 0

An upper limit on the number of zero/one variables in the model

to be solved. Use the exact number if available, since it is

used to allocate working storage.

KEYWORD: INVERT see: FACTOR

KEYWORD: LIMIT VALUES: positive integer DEFAULT: 5000

An upper limit on the number of LP iterations to be performed

during the branch-and-bound search. Intended as an escape hatch.

If you are running ZOOM interactively under IMP, then IMP will ask

you if you really want to stop or else extend LIMIT.

KEYWORD: MAX MODIFIER: COLUMN VALUES: positive integer

DEFAULT: 20

An upper limit on the number of non-zeros in any single matrix

column. Okay if greater than true value.

KEYWORD: MAX MODIFIER: DEPTH VALUES: positive integer

DEFAULT: number of zero/one variables

An upper limit on the maximum depth of search for branch-and-

bound. The search depth is equal to the number of zero/one

variables only if we are at the very bottom of the B&B tree,

and this should rarely happen. This parameter is provided to

allow you to save memory space on large problems.

KEYWORD: MAX MODIFIER: NODES VALUES: positive integer

DEFAULT: 500

An upper limit on the maximum number of nodes allowed in the

branch-and-bound tree. Default is re-set to one for a pure LP.

KEYWORD: MAX MODIFIER: SAVE VALUES: positive integer

DEFAULT: 5

An upper limit on the number of hot LP bases saved during the

branch-and-bound search. Saving more bases should speed up the

search, but each one saved takes up (COLUMNS+2\*ROWS) memory

locations.

KEYWORD: MINIMIZE

Use this to minimize. The default is to maximize.

KEYWORD: MULTIPLE VALUES: positive integer DEFAULT: 5

Multiple pricing parameter for the simplex method. This

is the number of attractive non-basic variables saved during

a major iteration for basis entry during the subsequent series

of minor iterations.

KEYWORD: OBJECTIVE VALUES: up to 8 characters

DEFAULT: 'PROFIT '

The name of the objective function used in the MPS file.

KEYWORD: ORDER VALUES: 1,2,3 DEFAULT: 1

Specifies the branching order for branch-and-bound.

If BS, UB, and LB denote the sets of zero/one variables that

are basic, at upper bound, and at lower bound respectively

in the optimal solution of the initial linear program, and

if the variables in these sets are ordered as described in

section (2 D.), then:

1 means BS, UB, LB

2 means UB, BS, LB

3 means user specified.

KEYWORD: PARTIAL VALUES: positive integer DEFAULT: 500

Partial pricing parameter for the simplex method. The number of

columns priced out during a major iteration. A major iteration

consists of pricing out PARTIAL columns and saving the MULTIPLE

best ones.

KEYWORD: POSTOP VALUES: 'YES', 'NO' DEFAULT: 'NO'

For use with pure linear programs only. 'YES' means perform a

sensitivity analysis on the objective coefficients and the

right-hand-side coefficients.

KEYWORD: PRINT MODIFIER: CONTINUOUS VALUES: 0,1

DEFAULT: 1

The default is to print the continuous solution, i.e.

the solution of the initial linear program. Use zero (0)

to suppress printing.

KEYWORD: PRINT MODIFIER: LP1 VALUES: 0,1,2,3

DEFAULT: 2

Output level for XMP while solving the initial linear program.

Higher values give more output.

KEYWORD: PRINT MODIFIER: LP2 VALUES: 0,1,2,3

DEFAULT: 0

Output level for XMP during the Pivot & Complement heuristic.

Higher values give more output.

KEYWORD: PRINT MODIFIER: LP3 VALUES: 0,1,2,3

DEFAULT: 0

Output level for XMP during the branch-and-bound search.

Higher values give more output.

KEYWORD: PRINT MODIFIER: HEURISTIC VALUES: 0,1,2,3

DEFAULT: 0

Output level for the Pivot & Complement heuristic.

Higher values give more output.

KEYWORD: PRINT MODIFIER: BRANCH VALUES: 0,1,2,3

DEFAULT: 1

Output level for the branch-and-bound search.

Higher values give more output.

KEYWORD: PRINT MODIFIER: TOUR VALUES: 0,1,...,6

DEFAULT: 0

Output level for the Resource Space Tour during the branch-

and-bound search. Higher values give more output. Values

2,...,6 are intended for serious debugging.

KEYWORD: QUIT VALUES: 'YES', 'NO' DEFAULT: 'NO'

Specifies whether or not the branch-and-bound search should

be halted when an improved incumbent solution is found.

If QUIT='YES', and you are running ZOOM interactively under IMP,

then IMP will ask you if you really want to quit or continue.

KEYWORD: RESTORE

Use this to restore a previously saved basis for the initial

linear program. (see KEYWORD: SAVE)

KEYWORD: ROWS VALUES: positive integer DEFAULT: 200

SYNONYM: CONSTRAINTS

An upper limit on the number of rows in the model to be solved.

The exact number should be used if available (it is not obvious

in the MPS file), since it is used to allocate working storage.

KEYWORD: SAVE

Use this to save the final basis for the initial linear program.

You can then restore this basis at the beginning of subsequent

runs and not have to resolve the initial linear program.

This is very valuable for large problems. (see KEYWORD: RESTORE)

KEYWORD: SELECT VALUES: positive integer DEFAULT: 2

The number of nodes to be selected for each expansion during

the branch-and-bound search. One expansion consists of choosing

a level in the search tree (see KEYWORD: DIVE), selecting

the SELECT best nodes at that level, and expanding them down

EXPAND levels, resulting in SELECT\*(2\*\*EXPAND) nodes.

(see KEYWORD: EXPAND)

KEYWORD: SHRINK VALUES: positive real>=1.0 DEFAULT: 5.0

The control parameter for the cheating strategy (see KEYWORD:

CHEAT). The smaller the value of CHEAT, the more you are

cheating. The more you cheat, the faster the branch-and-

bound search goes, but the less likely you are to find the

optimal solution.

KEYWORD: TOLERANCE VALUES: non-negative real DEFAULT: 0.1

The stopping tolerance for the branch-and-bound search.

During the search, we maintain an upper bound and a lower bound

on v(MIP), the optimal value of the integer program. If

(upper bound - lower bound) / (1+abs(upper bound)) <= TOLERANCE,

then the search is halted, since the current incumbent is

close enough to optimality.

Warning: setting TOLER = 0.0 can result in extremely long run

times for even small problems that have a great many optimal and

near optimal solutions. It is much safer to use a very small

value like 0.0001 than an exact 0.0.

KEYWORD: ZERO/ONE see: INTEGER

3 B. The MPS file

------------

The matrix data for the problem to be solved must be in

MPS standard form. The only restriction on the MPS standard

is that multiple BOUND's, RHS's and RANGE's are not allowed.

We have extended the MPS conventions for linear programs to

permit the specification of mixed-integer programs. There

are two different ways to identify the integer variables.

If all of the integer variables in your model are contiguous,

then you should use markers in the COLUMNS section of the MPS file:

a line with 'INT' in columns 1-3 immediately before the first

line for the first integer variable, and another line with

'INT' in columns 1-3 immediately after the last line for the

last integer variable. All variables between these two 'INT'

markers are assumed to be integer.

If the integer variables in your model are scattered, then

you should identify them in the BOUNDS section of the MPS file.

Use the special code 'BV' in columns 2-3. You may also use both

methods if you have a block of contiguous integer variables and

some scattered integer variables. You may not, however,

identify more than one block of integer variables with the

'INT' markers.

The bounds of zero and one for the integer variables are

established automatically and do not have to be declared

in the BOUNDS section of the MPS file. You can lock a variable

to either zero or one by using an 'FX' code in the BOUNDS section.

Actually typing MPS-standard files directly is a very unpleasant

and error prone task. It is unlikely that there is anyone still

willing to do such onerous tasks. It is suggested that you:

1) express your model in the XML language [10] and use

the XML translator to convert it into an MPS file; or

2) use the GAMS modeling system [9], which includes a bridge

to ZOOM; or

3) write your own matrix generator program to read your

raw data and generate the LP matrix, which can be

written out as an MPS file, or sent directly to

XMP [11].

GAMS is the best alternative, since it allows you to build large

models very quickly by declaring indexed variables and equations.

XML is intended for small models. The XML syntax looks like a

simplified, non-indexed version of GAMS. This should simplify the

transition from XML to GAMS as you move from simplified models to

more realistic ones.

CHAPTER 4. SAMPLE MODEL

------------

The sample model given here is from the Defense Supply Agency.

Their implementation of the model used the XML modeling

language and translator [10] to create the MPS file.

4 A. The SPECS file

--------------

BEGIN

PRINT HEUR 1

PRINT BRANCH 1

INTEGER 30

MINIMIZE

OBJECTIVE COST

BRANCH YES

DIVE NO

QUIT NO

TOLERANCE 0.001

END

4 B. The XML file

------------

$DEFENSE LOGISTICS AGENCY MODEL

$PARAMETERS

! THE COST FUNCTION FOR THE FIRST DRMO

! FOR USABLE AT EACH WORKLOAD LEVEL.

@RATE1=.8

@RATE2=.5

@RATE3=.8

! THE COST FUNCTION FOR BOTH DRMOS FOR SCRAP AT EACH WORKLOAD LEVEL.

@RATE4=.65

@RATE5=.35

! THE COST FUNCTION FOR THE SECOND DRMO

! FOR USABLE AT EACH WORKLOAD LEVEL.

@RATE6= .7

@RATE7= .4

@RATE8= .75

! THE UPPER BOUND OF EACH LEVEL OF USABLE WORKLOAD FOR BOTH DRMOS.

@CAP1 = 10

@CAP2 = 15

@CAP3 = 41

! THE UPPER BOUND OF EACH LEVEL OF SCRAP WORKLOAD FOR BOTH DRMOS.

@CAP4 = 15

@CAP5 = 30

! THE AMOUNT OF USABLE WORK LOAD GENERATED BY EACH CUSTOMER.

! WL1 = 1ST CUSTOMERS WORK LOAD, ETC.

@WL1 = 16

@WL2 = 17

@WL3 = 14

! THE AMOUNT OF SCRAP WORK LOAD GENERATED BY EACH CUSTOMER.

@WL1S = 5

@WL2S = 10

@WL3S = 5

! THE REUTILIZATION DEMAND BY CUSTOMER.

@DEM1 = 3

@DEM2 = 2

@DEM3 = 4

$VARIABLES

! THESE VARIABLES REPRESENT THE AMOUNT OF WORKLOAD PROCESSED

! BY THE JTH DRMO AT THE KTH LEVEL OF WORKLOAD. THE DESIGNATION

! U REPRESENT USABLE LINE ITEMS AND S REPRESENTS SCRAP.

Y11U Y12U Y13U Y21U Y22U Y23U

Y11S Y12S Y21S Y22S

! THE R VARIABLES REPRESENT THE AMOUNT OF MATERIAL REUTILIZED BY

! THE ITH CUSTOMER FROM THE JTH DRMO.

R11 R12 R21 R22 R31 R32

$VARIABLES BINARY

! THE X VARIABLES ARE THE ASSIGNMENT OF THE ITH CUSTOMER TO THE

! JTH DRMO. THE SUBSCRIPT U REPRESENTS USABLE AND S REPRESENTS

! SCRAP.

X11U X12U X21U X22U X31U X32U

X11S X12S X21S X22S X31S X32S

! THE Z VARIABLES ARE THE ASSIGNMENT OF THE KTH WORKLOAD LEVEL

! TO THE JTH DRMO

Z11U Z12U Z13U Z21U Z22U Z23U

Z11S Z12S Z21S Z22S

$EQUATIONS

COST .. 15X11U + 6X12U + 12X21U + 4X22U + 3X31U + 14X32U

+4X11S + 3X12S + 5X21S + 5X22S + 4X31S + 3X32S

+@RATE1 Y11U + @RATE2 Y12U + @RATE3 Y13U + @RATE6 Y21U

+@RATE7 Y22U + @RATE8 Y23U

+@RATE4 Y11S + @RATE5 Y12S

+@RATE4 Y21S + @RATE5 Y22S

+10 Z11U + 13 Z12U + 10.5 Z13U + 8 Z11S + 12 Z12S

+12 Z21U + 15 Z22U + 19.65 Z23U + 8 Z21S + 12 Z22S

+2 R11 + 3 R12 + 3 R21 + 4 R22 + 2 R31 + 3 R32 =N=

! THESE CONSTRAINTS FORCE THE WORKLOAD AT EACH FACILITY TO

! LIE ON A DISCRETE LINE SEGMENT WHICH IS ASSOCIATED WITH

! A WORK LOAD LEVEL Zjk. EACH LINE SEGMENT HAS ITS OWN COST

! RATE ASSOCIATED WITH IT. THE UPPER BOUND OF EACH LINE

! SEGMENT FOR EACH DRMO IS REPRESENTED BY @CAP\*.

SIZE1 .. Y11U - @CAP1 Z11U =L= 0

SIZE2 .. Y12U - @CAP2 Z12U =L= 0

SIZE3 .. Y13U - @CAP3 Z13U =L= 0

SIZE1S .. Y11S - @CAP4 Z11S =L= 0

SIZE2S .. Y12S - @CAP5 Z12S =L= 0

SIZE4 .. Y21U - @CAP1 Z21U =L= 0

SIZE5 .. Y22U - @CAP2 Z22U =L= 0

SIZE6 .. Y23U - @CAP3 Z23U =L= 0

SIZE4S .. Y21S - @CAP4 Z21S =L= 0

SIZE5S .. Y22S - @CAP5 Z22S =L= 0

! THESE CONSTRAINTS REQUIRE THE FACILITY WORKLOAD TO BE

! EQUAL TO THE CUSTOMER INPUT

FACWL1 .. Y11U + Y12U + Y13U - @WL1 X11U

-@WL2 X21U - @WL3 X31U =E= 0

FACWL2 .. Y21U + Y22U + Y23U

- @WL1 X12U - @WL2 X22U - @WL3 X32U =E=0

FACWL3 .. Y11S + Y12S - @WL1S X11S - @WL2S X21S - @WL3S X31S =E= 0

FACWL4 .. Y21S + Y22S - @WL1S X12S - @WL2S X22S - @WL3S X32S =E= 0

! THESE FORCE THE CUSTOMER TO SHIP HIS MATERIAL TO ONE AND

! ONLY ONE DRMO

FORCE1 .. X11U + X12U =E= 1

FORCE2 .. X21U + X22U =E= 1

FORCE3 .. X31U + X32U =E= 1

FORCE4 .. X11S + X12S =E= 1

FORCE5 .. X21S + X22S =E= 1

FORCE6 .. X31S + X32S =E= 1

! THESE REQUIRE THAT ONLY ONE WORKLOAD LEVEL CAN BE ASSIGNED

! TO EACH DRMO

ZSEL1U .. Z11U + Z12U + Z13U =L= 1

ZSEL1S .. Z11S + Z12S =L= 1

ZSEL2U .. Z21U + Z22U + Z23U =L= 1

ZSEL2S .. Z12S + Z22S =L= 1

! THESE REQUIRE THAT ALL DEMAND FOR REUTILIZATION

! BY EACH CUSTOMER TO BE MET BY THE DRMOS

FILL1 .. R11 + R12 =E= @DEM1

FILL2 .. R21 + R22 =E= @DEM2

FILL3 .. R31 + R32 =E= @DEM3

! THESE REQUIRE THAT THE DRMO THAT SATISFIES REUTILIZATION

! DEMAND MUST HAVE AT LEAST THE WORKLOAD INPUT TO SATISFY

! THAT DEMAND.

REUT1 .. R11 - @WL1 X11U - @WL2 X21U - @WL3 X31U =L= 0

REUT2 .. R12 - @WL1 X12U - @WL2 X22U - @WL3 X32U =L= 0

REUT3 .. R21 - @WL1 X11U - @WL2 X21U - @WL3 X31U =L= 0

REUT4 .. R22 - @WL1 X12U - @WL2 X22U - @WL3 X32U =L= 0

REUT5 .. R31 - @WL1 X11U - @WL2 X21U - @WL3 X31U =L= 0

REUT6 .. R32 - @WL1 X12U - @WL2 X22U - @WL3 X32U =L= 0

$BOUNDS

$END

4 C. The MPS file

------------

The field definitions for MPS files are as follows:

Field 1. spaces 2 to 3 indicator

Field 2. spaces 5 to 12 name

Field 3. spaces 15 to 22 name

Field 4. spaces 25 to 36 value

Field 5. spaces 40 to 47 name

Field 6. spaces 50 to 61 value

The Defense Supply Agency model was converted by the XML

translator into the following MPS file.

NAME DEFENSE LOGISTICS AGENCY MODEL

ROWS

N COST

L SIZE1

L SIZE2

L SIZE3

L SIZE1S

L SIZE2S

L SIZE4

L SIZE5

L SIZE6

L SIZE4S

L SIZE5S

E FACWL1

E FACWL2

E FACWL3

E FACWL4

E FORCE1

E FORCE2

E FORCE3

E FORCE4

E FORCE5

E FORCE6

L ZSEL1U

L ZSEL1S

L ZSEL2U

L ZSEL2S

E FILL1

E FILL2

E FILL3

L REUT1

L REUT2

L REUT3

L REUT4

L REUT5

L REUT6

COLUMNS

Y11U FACWL1 1.

Y11U SIZE1 1.

Y11U COST .8

Y12U FACWL1 1.

Y12U SIZE2 1.

Y12U COST .5

Y13U FACWL1 1.

Y13U SIZE3 1.

Y13U COST .8

Y21U FACWL2 1.

Y21U SIZE4 1.

Y21U COST .7

Y22U FACWL2 1.

Y22U SIZE5 1.

Y22U COST .4

Y23U FACWL2 1.

Y23U SIZE6 1.

Y23U COST .75

Y11S FACWL3 1.

Y11S SIZE1S 1.

Y11S COST .65

Y12S FACWL3 1.

Y12S SIZE2S 1.

Y12S COST .35

Y21S FACWL4 1.

Y21S SIZE4S 1.

Y21S COST .65

Y22S FACWL4 1.

Y22S SIZE5S 1.

Y22S COST .35

R11 REUT1 1.

R11 FILL1 1.

R11 COST 2.

R12 REUT2 1.

R12 FILL1 1.

R12 COST 3.

R21 REUT3 1.

R21 FILL2 1.

R21 COST 3.

R22 REUT4 1.

R22 FILL2 1.

R22 COST 4.

R31 REUT5 1.

R31 FILL3 1.

R31 COST 2.

R32 REUT6 1.

R32 FILL3 1.

R32 COST 3.

X11U REUT5 -16.

X11U REUT3 -16.

X11U REUT1 -16.

X11U FORCE1 1.

X11U FACWL1 -16.

X11U COST 15.

X12U REUT6 -16.

X12U REUT4 -16.

X12U REUT2 -16.

X12U FORCE1 1.

X12U FACWL2 -16.

X12U COST 6.

X21U REUT5 -17.

X21U REUT3 -17.

X21U REUT1 -17.

X21U FORCE2 1.

X21U FACWL1 -17.

X21U COST 12.

X22U REUT6 -17.

X22U REUT4 -17.

X22U REUT2 -17.

X22U FORCE2 1.

X22U FACWL2 -17.

X22U COST 4.

X31U REUT5 -14.

X31U REUT3 -14.

X31U REUT1 -14.

X31U FORCE3 1.

X31U FACWL1 -14.

X31U COST 3.

X32U REUT6 -14.

X32U REUT4 -14.

X32U REUT2 -14.

X32U FORCE3 1.

X32U FACWL2 -14.

X32U COST 14.

X11S FORCE4 1.

X11S FACWL3 -5.

X11S COST 4.

X12S FORCE4 1.

X12S FACWL4 -5.

X12S COST 3.

X21S FORCE5 1.

X21S FACWL3 -10.

X21S COST 5.

X22S FORCE5 1.

X22S FACWL4 -10.

X22S COST 5.

X31S FORCE6 1.

X31S FACWL3 -5.

X31S COST 4.

X32S FORCE6 1.

X32S FACWL4 -5.

X32S COST 3.

Z11U ZSEL1U 1.

Z11U SIZE1 -10.

Z11U COST 10.

Z12U ZSEL1U 1.

Z12U SIZE2 -15.

Z12U COST 13.

Z13U ZSEL1U 1.

Z13U SIZE3 -41.

Z13U COST 10.5

Z21U ZSEL2U 1.

Z21U SIZE4 -10.

Z21U COST 12.

Z22U ZSEL2U 1.

Z22U SIZE5 -15.

Z22U COST 15.

Z23U ZSEL2U 1.

Z23U SIZE6 -41.

Z23U COST 19.65

Z11S ZSEL1S 1.

Z11S SIZE1S -15.

Z11S COST 8.

Z12S ZSEL2S 1.

Z12S ZSEL1S 1.

Z12S SIZE2S -30.

Z12S COST 12.

Z21S SIZE4S -15.

Z21S COST 8.

Z22S ZSEL2S 1.

Z22S SIZE5S -30.

Z22S COST 12.

RHS

RHS FORCE1 1.

RHS FORCE2 1.

RHS FORCE3 1.

RHS FORCE4 1.

RHS FORCE5 1.

RHS FORCE6 1.

RHS ZSEL1U 1.

RHS ZSEL1S 1.

RHS ZSEL2U 1.

RHS ZSEL2S 1.

RHS FILL1 3.

RHS FILL2 2.

RHS FILL3 4.

BOUNDS

BV BOUNDS X11U

BV BOUNDS X12U

BV BOUNDS X21U

BV BOUNDS X22U

BV BOUNDS X31U

BV BOUNDS X32U

BV BOUNDS X11S

BV BOUNDS X12S

BV BOUNDS X21S

BV BOUNDS X22S

BV BOUNDS X31S

BV BOUNDS X32S

BV BOUNDS Z11U

BV BOUNDS Z12U

BV BOUNDS Z13U

BV BOUNDS Z21U

BV BOUNDS Z22U

BV BOUNDS Z23U

BV BOUNDS Z11S

BV BOUNDS Z12S

BV BOUNDS Z21S

BV BOUNDS Z22S

ENDATA

4 D. The Output

----------

Here is the ZOOM output for the sample model. All of the lower

case text is annotation that has been edited into the actual

output file.

Z O O M / X M P --- VERSION 4.0 APR 1987

= = = = = = = = = =

PROBLEM SPECIFICATIONS

----------------------

BEGIN

PRINT HEUR 1

PRINT BRANCH 1

INTEGER 30 echo print of your specs file

MINIMIZE

OBJECTIVE COST

BRANCH YES

DIVE NO

QUIT NO

TOLERANCE 0.001

END

listing of the most important parameter settings:

NO MORE THAN 200 CONSTRAINTS

NO MORE THAN 400 VARIABLES

NO MORE THAN 2200 NON-ZERO MATRIX ELEMENTS

THIS IS A ZERO/ONE INTEGER PROGRAM

NO MORE THAN 30 ZERO/ONE VARIABLES

STOPPING TOLERANCE 0.10000000D-02

GAP ESTIMATE (OR +INF) 0.10000000D+31

INCUMBENT VALUE (OR -INF) -0.10000000D+31

WE WILL USE THE HEURISTIC

WE WILL DO BRANCH-AND-BOUND

LIMIT ON LP ITERATIONS DURING B&B SEARCH: 5000

DURING B&B WE WILL SELECT 2 NODES AND

EXPAND THEM DOWN 3 LEVELS

WE WILL SAVE UP TO 5 WARM BASES

WE WILL ALLOW UP TO 500 NODES IN THE B&B TREE

B&B WITH BEST BOUND STRATEGY

B&B SEARCH WILL CONTINUE UNTIL TREE IS EMPTY,

STOPPING TOLERANCE IS SATISFIED,

OR LIMIT ON LP ITERATIONS IS EXCEEDED

B&B BRANCHING ORDER: BS,UB,LB

information about available memory space:

YOU HAVE ROOM FOR 13809 NON-ZEROS IN THE BASIS FACTORS

YOU COULD REDUCE REAL MEMORY FROM 31200 TO 25236

DEFENSE LOGISTICS AGENCY MODEL

messages from the routine that reads the MPS file:

CONSTRAINTS IN THE MPS FILE: 33

VARIABLES IN THE MPS FILE: 38

MATRIX NON-ZEROS IN THE MPS FILE: 94

ACTUAL NUMBER OF CONSTRAINTS= 33

ACTUAL NUMBER OF VARIABLES= 38

ACTUAL NUMBER OF ZERO/ONE VARIABLES= 22

these messages from xfeas and xphas2 appear whenever an lp

is being solved, unless the XMP print level is set to zero;

they refer to the current linear program, not the overall

mixed-integer program:

XFEAS...FEASIBLE SOLUTION FOUND.

XPHAS2... OPTIMAL SOLUTION FOUND.

INITIAL LP VALUE= 0.11435122D+03 PIVOTS= 55

TERMINATION CODE= 1

1 means finite optimal solution

2 means unbounded optimal value

3 means infeasible

>3 means trouble

here is the solution to the lp relaxation of (mip),

notice the fractional values for some of the zero/one variables:

\*\*\* INITIAL LP SOLUTION \*\*\*

DEFENSE LOGISTICS AGENCY MODEL

OPTIMAL SOLUTION TO LINEAR PROGRAM

COST = 0.11435122D+03

ROW EQUATION STATUS SLACK/SURPLUS SHADOW PRICE

1 SIZE1 TIGHT -0.25609756D+00

2 SIZE2 TIGHT -0.86666667D+00

3 SIZE3 TIGHT -0.25609756D+00

4 SIZE1S TIGHT -0.10000000D+00

5 SIZE2S TIGHT -0.40000000D+00

6 SIZE4 TIGHT -0.52926829D+00

7 SIZE5 TIGHT -0.82926829D+00

8 SIZE6 TIGHT -0.47926829D+00

9 SIZE4S TIGHT -0.10000000D+00

10 SIZE5S TIGHT -0.40000000D+00

11 FACWL1 TIGHT 0.10560976D+01

12 FACWL2 TIGHT 0.12292683D+01

13 FACWL3 TIGHT 0.75000000D+00

14 FACWL4 TIGHT 0.75000000D+00

15 FORCE1 TIGHT 0.31897561D+02

16 FORCE2 TIGHT 0.24897561D+02

17 FORCE3 TIGHT 0.17785366D+02

18 FORCE4 TIGHT 0.67500000D+01

19 FORCE5 TIGHT 0.12500000D+02

20 FORCE6 TIGHT 0.77500000D+01

21 ZSEL1U LOOSE 0.65853659D+00

22 ZSEL1S LOOSE 0.66666667D+00

23 ZSEL2U LOOSE 0.19512195D+00

24 ZSEL2S LOOSE 0.33333333D+00

25 FILL1 TIGHT 0.20000000D+01

26 FILL2 TIGHT 0.30000000D+01

27 FILL3 TIGHT 0.20000000D+01

28 REUT1 LOOSE 0.11000000D+02

29 REUT2 LOOSE 0.33000000D+02

30 REUT3 LOOSE 0.12000000D+02

31 REUT4 LOOSE 0.33000000D+02

32 REUT5 LOOSE 0.10000000D+02

33 REUT6 LOOSE 0.33000000D+02

COL VARIABLE STATUS VALUE REDUCED COST

1 Y11U BASIC 0.00000000D+00

2 Y12U LOWER 0.31056911D+00

3 Y13U BASIC 0.14000000D+02

4 Y21U BASIC 0.00000000D+00

5 Y22U BASIC 0.00000000D+00

6 Y23U BASIC 0.33000000D+02

7 Y11S BASIC 0.00000000D+00

8 Y12S BASIC 0.10000000D+02

9 Y21S BASIC 0.00000000D+00

10 Y22S BASIC 0.10000000D+02

11 R11 BASIC 0.30000000D+01

12 R12 LOWER 0.10000000D+01

13 R21 BASIC 0.20000000D+01

14 R22 LOWER 0.10000000D+01

15 R31 BASIC 0.40000000D+01

16 R32 LOWER 0.10000000D+01

the x and z variables are the zero/one variables,

note the fractional values:

17 X11U BASIC 0.00000000D+00

18 X12U UPPER 0.10000000D+01 -0.62292683D+01

19 X21U LOWER 0.50560976D+01

20 X22U BASIC 0.10000000D+01

21 X31U BASIC 0.10000000D+01

22 X32U LOWER 0.13424390D+02

23 X11S LOWER 0.10000000D+01

24 X12S BASIC 0.10000000D+01

25 X21S BASIC 0.10000000D+01

26 X22S LOWER 0.00000000D+00

27 X31S BASIC 0.00000000D+00

28 X32S UPPER 0.10000000D+01 -0.10000000D+01

29 Z11U LOWER 0.74390244D+01

30 Z12U BASIC 0.00000000D+00

31 Z13U BASIC 0.34146341D+00

32 Z21U LOWER 0.67073171D+01

33 Z22U LOWER 0.25609756D+01

34 Z23U BASIC 0.80487805D+00

35 Z11S LOWER 0.65000000D+01

36 Z12S BASIC 0.33333333D+00

37 Z21S LOWER 0.65000000D+01

38 Z22S BASIC 0.33333333D+00

the pivot & complement routine has found a

feasible integer solution:

HEURISTIC SOLUTION WITH VALUE 0.14260000D+03

XFEAS...FEASIBLE SOLUTION FOUND.

XPHAS2... OPTIMAL SOLUTION FOUND.

the first feasible integer solution found is improved

by doing additional single and double complements:

(the negative of the cost gets printed by the deeper routines,

since internally we are always maximizing)

ZIMPRO...STARTING VALUE= -0.14260000D+03

IMPROVED ALL INTEGER SOLUTION FOUND:

double complement... variable 23 goes from 1 to 0,

variable 24 goes from 0 to 1

OBJECTIVE= -0.14010000D+03 VARIABLE= 23 NEW STATUS= 0

VARIABLE= 24 NEW STATUS=-1

IMPROVED ALL INTEGER SOLUTION FOUND:

OBJECTIVE= -0.13710000D+03 VARIABLE= 25 NEW STATUS= 0

VARIABLE= 26 NEW STATUS=-1

IMPROVED ALL INTEGER SOLUTION FOUND:

single complement...variable 35 goes from 1 to 0

OBJECTIVE= -0.12910000D+03 VARIABLE= 35 NEW STATUS= 0

HEURISTIC SOLUTION IMPROVED TO 0.12910000D+03

here is the best soution found by the pivot & complement

heuristic:

\*\*\* BEST SOLUTION FOUND BY HEURISTIC \*\*\*

DEFENSE LOGISTICS AGENCY MODEL

OPTIMAL SOLUTION TO LINEAR PROGRAM

COST = 0.12910000D+03

ROW EQUATION STATUS SLACK/SURPLUS SHADOW PRICE

1 SIZE1 TIGHT 0.00000000D+00

2 SIZE2 TIGHT -0.30000000D+00

3 SIZE3 LOOSE 0.27000000D+02

4 SIZE1S LOOSE 0.00000000D+00

5 SIZE2S TIGHT -0.30000000D+00

6 SIZE4 TIGHT -0.50000000D-01

7 SIZE5 TIGHT -0.35000000D+00

8 SIZE6 LOOSE 0.80000000D+01

9 SIZE4S LOOSE 0.00000000D+00

10 SIZE5S LOOSE 0.10000000D+02

11 FACWL1 TIGHT 0.80000000D+00

12 FACWL2 TIGHT 0.75000000D+00

13 FACWL3 TIGHT 0.65000000D+00

14 FACWL4 TIGHT 0.35000000D+00

15 FORCE1 TIGHT 0.00000000D+00

16 FORCE2 TIGHT 0.00000000D+00

17 FORCE3 TIGHT 0.00000000D+00

18 FORCE4 TIGHT 0.00000000D+00

19 FORCE5 TIGHT 0.00000000D+00

20 FORCE6 TIGHT 0.00000000D+00

21 ZSEL1U LOOSE 0.00000000D+00

22 ZSEL1S LOOSE 0.10000000D+01

23 ZSEL2U LOOSE 0.00000000D+00

24 ZSEL2S LOOSE 0.00000000D+00

25 FILL1 TIGHT 0.20000000D+01

26 FILL2 TIGHT 0.30000000D+01

27 FILL3 TIGHT 0.20000000D+01

28 REUT1 LOOSE 0.11000000D+02

29 REUT2 LOOSE 0.33000000D+02

30 REUT3 LOOSE 0.12000000D+02

31 REUT4 LOOSE 0.33000000D+02

32 REUT5 LOOSE 0.10000000D+02

33 REUT6 LOOSE 0.33000000D+02

COL VARIABLE STATUS VALUE REDUCED COST

1 Y11U BASIC 0.00000000D+00

2 Y12U BASIC 0.00000000D+00

3 Y13U BASIC 0.14000000D+02

4 Y21U BASIC 0.00000000D+00

5 Y22U BASIC 0.00000000D+00

6 Y23U BASIC 0.33000000D+02

7 Y11S BASIC 0.00000000D+00

8 Y12S BASIC 0.00000000D+00

9 Y21S LOWER 0.30000000D+00

10 Y22S BASIC 0.20000000D+02

11 R11 BASIC 0.30000000D+01

12 R12 LOWER 0.10000000D+01

13 R21 BASIC 0.20000000D+01

14 R22 LOWER 0.10000000D+01

15 R31 BASIC 0.40000000D+01

16 R32 LOWER 0.10000000D+01

note all zero/one values for the x's and z's this time:

17 X11U LOWER 0.27800000D+02

18 X12U UPPER 0.10000000D+01 0.18000000D+02

19 X21U LOWER 0.25600000D+02

20 X22U UPPER 0.10000000D+01 0.16750000D+02

21 X31U UPPER 0.10000000D+01 0.14200000D+02

22 X32U LOWER 0.24500000D+02

23 X11S LOWER 0.72500000D+01

24 X12S UPPER 0.10000000D+01 0.47500000D+01

25 X21S LOWER 0.11500000D+02

26 X22S UPPER 0.10000000D+01 0.85000000D+01

27 X31S LOWER 0.72500000D+01

28 X32S UPPER 0.10000000D+01 0.47500000D+01

29 Z11U LOWER 0.10000000D+02

30 Z12U LOWER 0.85000000D+01

31 Z13U UPPER 0.10000000D+01 0.10500000D+02

32 Z21U LOWER 0.11500000D+02

33 Z22U LOWER 0.97500000D+01

34 Z23U UPPER 0.10000000D+01 0.19650000D+02

35 Z11S LOWER 0.80000000D+01

36 Z12S LOWER 0.30000000D+01

37 Z21S LOWER 0.80000000D+01

38 Z22S UPPER 0.10000000D+01 0.12000000D+02

now we are ready to start branch-and-bound

we know that 114.35 <= v(mip) <= 129.10

(note that the negative of the cost gets printed -- that is

because internally zoom is always maximizing)

UPPER BOUND= -0.11435122D+03

LOWER BOUND= -0.12910000D+03 GAP= 0.12897790D+00

about a 13% gap

with print level 1 during branch-and-bound, you will get

a few lines printed for each select, expand, and resource

space tour cycle. the information printed includes the

depth expanded from, the depth expanded to,

the number of nodes going in to the resource space tour,

and the number of nodes surviving the tour.

here, for example, the single root node is expanded into

8 nodes at depth 3, and 4 of these nodes survive the

resource space tour:

EXPAND FROM DEPTH 0 TO DEPTH 3 NODES= 8

4 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

we also print the current status of the tree, i.e. how

many waiting leaf nodes there are at each depth:

4 NODES AT DEPTH 3 BEST BOUND= 0.11435122D+03

UPPER BOUND= -0.11435122D+03

LOWER BOUND= -0.12910000D+03 GAP= 0.128978D+00

EXPAND FROM DEPTH 3 TO DEPTH 6 NODES= 16

there are 16 nodes because

we are using a 2,3 strategy

6 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2 NODES AT DEPTH 3 BEST BOUND= 0.11535122D+03

6 NODES AT DEPTH 6 BEST BOUND= 0.12253333D+03

note that the upper bound decreases as the search continues,

since it is always the maximum lp value over all of the leaf

nodes in the current tree:

UPPER BOUND= -0.11535122D+03

LOWER BOUND= -0.12910000D+03 GAP= 0.119191D+00

EXPAND FROM DEPTH 3 TO DEPTH 6 NODES= 16

6 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12 NODES AT DEPTH 6 BEST BOUND= 0.12253333D+03

UPPER BOUND= -0.12253333D+03

LOWER BOUND= -0.12910000D+03 GAP= 0.535909D-01

EXPAND FROM DEPTH 6 TO DEPTH 9 NODES= 16

here we get lucky and the simplex method turns up a

natural integer lp solution:

INTEGER FEASIBLE LP SOLUTION...VALUE= 0.12740000D+03

NEW INCUMBENT: OBJECTIVE FUNCTION VALUE= 0.12740000D+03

this new incumbent allows us to permanently lock one of

the zero/one variables:

ZLOCK...VARIABLE 22 LOCKED AT ZERO

note that the algorithmic routines can only identify variables

by number; their names are only available in the higher

level routines

0 SURVIVORS a massacre!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10 NODES AT DEPTH 6 BEST BOUND= 0.12353333D+03

UPPER BOUND= -0.12353333D+03

LOWER BOUND= -0.12740000D+03 GAP= 0.313006D-01

EXPAND FROM DEPTH 6 TO DEPTH 9 NODES= 16

0 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8 NODES AT DEPTH 6 BEST BOUND= 0.12432500D+03

UPPER BOUND= -0.12432500D+03

LOWER BOUND= -0.12740000D+03 GAP= 0.247336D-01

EXPAND FROM DEPTH 6 TO DEPTH 9 NODES= 16

0 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6 NODES AT DEPTH 6 BEST BOUND= 0.12510000D+03

UPPER BOUND= -0.12510000D+03

LOWER BOUND= -0.12740000D+03 GAP= 0.183853D-01

EXPAND FROM DEPTH 6 TO DEPTH 9 NODES= 16

0 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4 NODES AT DEPTH 6 BEST BOUND= 0.12532500D+03

UPPER BOUND= -0.12532500D+03

LOWER BOUND= -0.12740000D+03 GAP= 0.165570D-01

EXPAND FROM DEPTH 6 TO DEPTH 9 NODES= 16

0 SURVIVORS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2 NODES AT DEPTH 6 BEST BOUND= 0.12610000D+03

UPPER BOUND= -0.12610000D+03

LOWER BOUND= -0.12740000D+03 GAP= 0.103093D-01

EXPAND FROM DEPTH 6 TO DEPTH 9 NODES= 16

0 SURVIVORS

END OF BRANCH-AND-BOUND SEARCH BECAUSE

THE SEARCH TREE IS EMPTY

LP PIVOTS= 217 total during b&b

DIRECT HITS= 53 total over all tours

INDIRECT HITS= 66 total over all tours

THE TREE NEVER HAD MORE THAN 68 NODES

BEST BOUND = 0.12740000D+03

INCUMBENT VALUE= 0.12740000D+03

BEST DISCARD = 0.12740000D+03

!!! INCUMBENT SOLUTION PROVEN OPTIMAL !!!

we are solving an lp now with all of the zero/one variables

fixed at their optimal values, to get the associated basis:

XFEAS...FEASIBLE SOLUTION FOUND.

XPHAS2... OPTIMAL SOLUTION FOUND.

here is the proven optimal integer solution for this model:

\*\*\* BEST INTEGER SOLUTION FOUND \*\*\*

DEFENSE LOGISTICS AGENCY MODEL

OPTIMAL SOLUTION TO LINEAR PROGRAM

COST = 0.12740000D+03

ROW EQUATION STATUS SLACK/SURPLUS SHADOW PRICE

1 SIZE1 LOOSE 0.00000000D+00

2 SIZE2 LOOSE 0.10000000D+01

3 SIZE3 LOOSE 0.00000000D+00

4 SIZE1S LOOSE 0.00000000D+00

5 SIZE2S LOOSE 0.00000000D+00

6 SIZE4 TIGHT -0.50000000D-01

7 SIZE5 TIGHT -0.35000000D+00

8 SIZE6 LOOSE 0.80000000D+01

9 SIZE4S LOOSE 0.00000000D+00

10 SIZE5S LOOSE 0.10000000D+02

11 FACWL1 TIGHT 0.50000000D+00

12 FACWL2 TIGHT 0.75000000D+00

13 FACWL3 TIGHT 0.00000000D+00

14 FACWL4 TIGHT 0.35000000D+00

15 FORCE1 TIGHT 0.00000000D+00

16 FORCE2 TIGHT 0.00000000D+00

17 FORCE3 TIGHT 0.00000000D+00

18 FORCE4 TIGHT 0.00000000D+00

19 FORCE5 TIGHT 0.00000000D+00

20 FORCE6 TIGHT 0.00000000D+00

21 ZSEL1U LOOSE 0.00000000D+00

22 ZSEL1S LOOSE 0.10000000D+01

23 ZSEL2U LOOSE 0.00000000D+00

24 ZSEL2S LOOSE 0.00000000D+00

25 FILL1 TIGHT 0.20000000D+01

26 FILL2 TIGHT 0.30000000D+01

27 FILL3 TIGHT 0.20000000D+01

28 REUT1 LOOSE 0.11000000D+02

29 REUT2 LOOSE 0.33000000D+02

30 REUT3 LOOSE 0.12000000D+02

31 REUT4 LOOSE 0.33000000D+02

32 REUT5 LOOSE 0.10000000D+02

33 REUT6 LOOSE 0.33000000D+02

COL VARIABLE STATUS VALUE REDUCED COST

1 Y11U LOWER 0.30000000D+00

2 Y12U BASIC 0.14000000D+02

3 Y13U LOWER 0.30000000D+00

4 Y21U BASIC 0.00000000D+00

5 Y22U BASIC 0.00000000D+00

6 Y23U BASIC 0.33000000D+02

7 Y11S LOWER 0.65000000D+00

8 Y12S LOWER 0.35000000D+00

9 Y21S LOWER 0.30000000D+00

10 Y22S BASIC 0.20000000D+02

11 R11 BASIC 0.30000000D+01

12 R12 LOWER 0.10000000D+01

13 R21 BASIC 0.20000000D+01

14 R22 LOWER 0.10000000D+01

15 R31 BASIC 0.40000000D+01

16 R32 LOWER 0.10000000D+01

17 X11U LOWER 0.23000000D+02

18 X12U UPPER 0.10000000D+01 0.18000000D+02

19 X21U LOWER 0.20500000D+02

20 X22U UPPER 0.10000000D+01 0.16750000D+02

21 X31U UPPER 0.10000000D+01 0.10000000D+02

22 X32U FIXED 0.24500000D+02

23 X11S LOWER 0.40000000D+01

24 X12S UPPER 0.10000000D+01 0.47500000D+01

25 X21S LOWER 0.50000000D+01

26 X22S UPPER 0.10000000D+01 0.85000000D+01

27 X31S LOWER 0.40000000D+01

28 X32S UPPER 0.10000000D+01 0.47500000D+01

29 Z11U LOWER 0.10000000D+02

30 Z12U UPPER 0.10000000D+01 0.13000000D+02

31 Z13U LOWER 0.10500000D+02

32 Z21U LOWER 0.11500000D+02

33 Z22U LOWER 0.97500000D+01

34 Z23U UPPER 0.10000000D+01 0.19650000D+02

35 Z11S LOWER 0.80000000D+01

36 Z12S LOWER 0.12000000D+02

37 Z21S LOWER 0.80000000D+01

38 Z22S UPPER 0.10000000D+01 0.12000000D+02

CHAPTER 5. "EXPERT" ADVICE

---------------

The time and computational effort required to solve a mixed-integer

program by Branch-and-Bound is notoriously unpredictable. ZOOM has

many options and parameters that you can control in your search for

an optimal, or near optimal, integer solution. Here, in the

currently fashionable form of IF-THEN rules, are some suggestions

about using these parameters.

1. IF the model you want to solve is new, unfamiliar, or large,

THEN plan on making more than one run.

2. IF you are going to make more than one run AND this is your

first run, THEN use HEURISTIC='YES', BRANCH='NO', and SAVE.

3. IF (the heuristic has found a feasible integer solution OR

an earlier B&B search has found a feasible integer solution OR

you know a feasible integer solution),

THEN set INCUMBENT to the value of that solution and

compute CRITERION = (v(LP)-INCUMBENT)/(1+abs(v(LP))).

4. IF you do not have any feasible integer solutions,

THEN set CRITERION = 100.

5. IF CRITERION < .10, THEN make a B&B search with BRANCH='YES',

DIVE='NO', QUIT='NO', and RESTORE.

6. IF .10 < CRITERION < .25, THEN make a B&B run with

BRANCH='YES', DIVE='NO', QUIT='YES', and RESTORE.

7. IF CRITERION > .25 AND you have not tried B&B yet,

THEN make a B&B run with BRANCH='YES', DIVE='YES',

QUIT='YES', SELECT=1, EXPAND=4, and RESTORE.

8. IF CRITERION > .25 AND you have already tried B&B,

THEN bluff, i.e. use the INCUMBENT or GAP parameter to

pretend that you have a feasible integer solution that

is within 25% of v(LP).

9. IF CRITERION > .25 AND you have already tried bluffing,

THEN cheat, i.e. use the CHEAT and SHRINK parameters to

try to find a better feasible integer solution.

10. IF you found an improved feasible integer solution during

a B&B run with cheating, THEN make another B&B run with

cheating, but cheat less, i.e. increase SHRINK.

11. IF you have made several B&B runs AND you have a feasible

integer solution AND CRITERION > .50 AND the current upper

bound on v(MIP) is almost the same as v(LP),

THEN you should give up on trying to satisfy your stopping

TOLERANCE. In this event, B&B is just not going to bring

the upper bound down in any reasonable amount of time.

Any further B&B runs should be pure diving raids (DIVE='YES',

QUIT='YES') , perhaps with bluffing or cheating, looking

for improved integer solutions (i.e. increase INCUMBENT).

REFERENCES

----------

1. Marsten, R. E., "The design of the XMP linear programming

library", ACM TRANSACTIONS ON MATHEMATICAL SOFTWARE,

Vol. 7, No. 4, December, 1981, pages 481-497.

2. Reid, J. K., "A sparsity-exploiting variant of the Bartels-

Golub decomposition for linear programming bases",

MATHEMATICAL PROGRAMMING, Vol. 24, No. 1, September, 1982,

pages 55-69.

3. Balas, E. and Martin, C. H., "Pivot and Complement--

a heuristic for 0-1 programming", MANAGEMENT SCIENCE,

Vol. 26, No. 1, January, 1980, pages 86-96.

4. Marsten, R. E. and Morin, T. L., "A hybrid approach to

discrete mathematical programming", MATHEMATICAL PROGRAMMING,

Vol. 14, No. 1, January, 1978, pages 21-40.

5. Marsten, R. E. and Singhal, J., "New results on fixed order

branch-and-bound methods", presentation at the ORSA-TIMS

National Meeting, Houston, Texas, October 11-14, 1981.

6. Singhal J., "Fixed Order Branch-and-Bound Methods for

Mixed-Integer Programming", Ph.D. dissertation, Dept. of

Management Information Systems, University of Arizona,

Tucson, AZ 85721, January, 1982.

7. Morin, T. L. and Marsten R. E., "Branch-and-bound methods

with stronger-than-optimal bounds", presentation at the

ORSA-TIMS Joint National Meeting, Washington, D.C.,

May 4-7, 1980.

8. Marsten, R. E. and Morin, T. L., "Breadth first branch-and-

bound methods for mixed-integer programming: computational

results", presentation at the ORSA-TIMS Joint National

Meeting, Colorado Springs, Colorado, November 10-12, 1980.

9. Kendrick, D. and Meeraus, A., "GAMS: An Introduction",

Development Research Department, The World Bank, February,

1985.

10. Marsten, R., "XML Modeling Language", Dept. of Management

Information Systems, University of Arizona, Tucson, AZ,

85721, April, 1987.

11. Marsten, R., "XMP Technical Reference Manual", Dept. of

Management Information Systems, University of Arizona,

Tucson, AZ, 85721, April, 1987.

end of manual