Handling Indivisibilities

Bruce A. McCarl

Specialist in Applied Optimization
Regents Professor of Agricultural Economics,
Texas A&M University
Principal, McCarl and Associates

bruceamccarl@cox-internet.com
agecon.tamu.edu/faculty/mccarl

979-693-5694
979-845-1706

Handling Indivisibilities

All or None -- Integer Programming

Many investment and other problems involve cases where one has to choose to have all or none of an item.

We cannot build ½ of a plant or buy 3/4 of a machine. We must buy or build 1 or 2 or 3 or none but not a fractional part

This leads to integer programming

$$
\begin{array}{lllll}
\max & C_1 W & + C_2 X & + C_3 Y & \\
s.t. & A_1 W & + A_2 X & + A_3 Y & \le b \\
 & W & & & \ge 0 \\
 & & X & & \ge 0 \text{ and integer} \\
 & & & Y & \text{equals } 0 \text{ or } 1
\end{array}
$$

Here W is a normal, continuous LP variable,
X is an integer variable,
Y is a zero one variable

When problems have
- only X they are called pure integer
- only Y they are called pure zero one
- W and X they are called mixed integer
- other variants exist

Handling Indivisibilities

Logical Conditions -- Integer Programming

Integer programming also allows many powerful logical conditions to be imposed

Consider the following: Suppose I am running a bottling plant that runs white milk but sometimes chocolate

If any chocolate is run, I need to clean at cost of F. Let X then be the amount of white milk processed

Then we add a component to the model as follows

$$\begin{array}{lrrl} \max & CX & -FY & \\ s.t. & X & -MY & \le 0 \\ & X & & \ge 0 \\ & & Y & \text{equals } 0 \text{ or } 1 \end{array}$$

Note in this component M is a big number (10 billion) Also if X is non zero this implies Y must equal 1
While if X is zero then given F>0 the Y will equal 0.

So if we run any chocolate milk we set y=1 and must clean incurring the cost of cleaning whether it be 1 gallon or one million

Y is an indicator variable.

Handling Indivisibilities
.Integer Programming

Similarly suppose we can buy from k different types of machines and get from them capacity for the ith time period

Max $\sum_m C_m X_m \; -\sum_k F_k Y_k$

S.t. $\sum_m a_{im} X_m \; -\sum_k CAP_{ik} Y_k \leq 0$ for all i

$X_m \geq 0$ for all m

$Y_k \; \varepsilon(0,1)$ for all k

In this case if they were mutually exclusive we could also add

$$\sum_k Y_k = 1$$

or if buying one meant we must buy another

$$Y_1 - Y_2 = 0$$

or if a mutually exclusive machine can only be purchased if we have a minimum volume being used

$$-\sum_m a_{im} X_m + \sum_k LL_{ik} Y_k \leq 0 \; for\; all\; i$$

Handling Indivisibilities
Integer Programming in GAMS

Maximize $7X_1$ $-3X_2$ $-10X_3$

X_1 $-2X_2$ ≤ 0

X_1 $-20X_3$ ≤ 0

X_1 ≥ 0 X_2 ≥ 0 integer X_3 0,1

GAMS Input for Example Integer Program (basint.gms)

```
POSITIVE VARIABLE      X1
INTEGER VARIABLE       X2
BINARY VARIABLE        X3
VARIABLE               OBJ
EQUATIONS              OBJF
                       X1X2
                       X1X3;
OBJF..    7*X1-3*X2-10*X3 =E= OBJ;
X1X2..    X1-2*X2 =L=0;
X1X3..    X1-20*X3 =L=0;
option optcr=0.01;
MODEL IPTEST /ALL/;
SOLVE IPTEST USING MIP MAXIMIZING OBJ;
```

Differences

1. Must tell the type of integer variable
2. Should set optcr or optca (problems occur if this is not done because the default values are very large)
3. Use MIP solve – need OSL, CPLEX or XPress not ZOOM

Handling Indivisibilities

Integer Programming Solution Difficulty

All sounds good but problems are hard. Let's explore why

Calculus is basis of all continuous optimization but not here because there is no neighborhood around a point in which a derivative can be defined

$$
\begin{aligned}
2X_1 + 3X_2 &\le 16 \\
3X_1 + 2X_2 &\le 16 \\
X_1 \;,\; X_2 &\ge 0 \;\&\; \text{integer}
\end{aligned}
$$

Feasible Region for X,Y nonnegative integers

Handling Indivisibilities
Integer Programming Solution Difficulty

Note

1. Solutions are finite
2. A line between 2 feasible points does not contain all feasible points
3. Moving between points is not always easy
4. Points are on boundary, interior, and not in general at corners
5. Rounding of LP point may not be bad

Handling Indivisibilities

Integer Programming Solution Difficulty

Consider

$$\begin{aligned} -X_1 &+ 7X_2 \geq 23 \\ X_1 &+ 10X_2 \leq 54 \\ X_1 &, X_2 \geq 0 \ \& \ \text{int}\, eger \end{aligned}$$

Here

- Rounding not good
- Movement between points hard

Handling Indivisibilities

Integer Programming Solution Difficulty

Mixed Integer Programming Feasible Region X1 integer, X2 continuous

$$
\begin{aligned}
3X_1 \quad &+2X_2 \quad \le \ 16 \\
2X_1 \quad &+3X_2 \quad \le \ 16 \\
X_1 \quad & \qquad \ge 0 \ \& \ \text{int}\, eger \\
, \quad & X_2 \quad \ge 0
\end{aligned}
$$

Handling Indivisibilities
Integer Programming Solution – Rounding

Solving the problem (solintr.gms)

$$
\begin{array}{lrcl}
Max & 1.4\,X_1 + X_2 & & \\
s.t. & 2\,X_1 + 3\,X_2 & \le & 16 \\
 & 3\,X_1 + 2\,X_2 & \le & 16 \\
 & X_1 \quad , \quad X_2 & \ge & 0 \;\&\; \text{int}\,eger
\end{array}
$$

as an LP yields X1=X2=3.2 which can be rounded to X1=X2=3 , Obj=7.2

But this may not always be feasible or optimal

In this case an objective of 7.6 arises at X1=4,X2=2 (solint.gms)

Rounding only works well if variable values are large

Handling Indivisibilities

Integer Programming Solution -- Branch and Bound

Solving (solintr.gms)

$$\begin{array}{lrcl} Max & 1.4X_1 + X_2 & & \\ s.t. & 2X_1 + 3X_2 & \le & 16 \\ & 3X_1 + 2X_2 & \le & 16 \\ & X_1 \quad , \quad X_2 & \ge & 0 \;\&\; \text{int}\, eger \end{array}$$

as an LP yields X1=X2=3.2 which can be rounded to X1=X2=3 , Obj=7.2

We can generate 2 related problems that collectively do not exclude integer variables as follows

$$\begin{bmatrix} \max & 1.4X_1 & +X_2 & \\ & 2X_1 & +3X_2 & \le 16 \\ & 3X_1 & +2X_2 & \le 16 \\ & X_1 & & \le 3 \\ & X_1 & X_2 & \ge 0 \;\&\; \text{int}\, eger \end{bmatrix} or \begin{bmatrix} \max & 1.4X_1 & +X_2 & \\ & 2X_1 & +3X_2 & \le 16 \\ & 3X_1 & +2X_2 & \le 16 \\ & X_1 & & \ge 4 \\ & X_1 & X_2 & \ge 0 \;\&\; \text{int}\, eger \end{bmatrix}$$

Suppose we solve the first we get (solx13.gms) x1=3.x2=3.33 and again generate 2 more problems the first with x2 ■ 3 and the other with x2 ▲ 4. Solving these solx1x23.gms, solx1x24.gms yields an integer solution at x1=x2=3 obj=7.2 and another at x1=2,x2=4 obj=6.8

Now since we are maximizing we choose the 7.2 as the best solution and call it the incumbent. But it is not necessarily optimal (in fact it is not at all). To verify its optimality we need to go back and investigate the problems we have not yet solved which still have the potential of having an objective function above our current best (7.2). We would then go back to the right hand problem from the first setup and eventually find X1=4,X2=2, Obj=7.6.

The above reveals the basic nature of branch and bound. It begins by solving an LP then finds a variable that is not integer and generates 2 problems (creating a branch). It then solves one of these and continues until it finds an integer solution which establishes a bound. We then backtrack and try to eliminate all other possible branches either by finding they cannot have a better objective than the incumbent (the bounding step) or are infeasible.

Suppose we solve a real example and see how this performs

Handling Indivisibilities

Integer Programming Solution -- Branch and Bound

Here is a problem in construction project setting where the agency paying for construction wished to invest funds subject to constraints ensuring payments could be met and composition restraints and that certain investments had to be of minimum size and in even amounts.

The model is as follows (secur.gms)

```
variables           obj      objective function
integer variables   invest(investment,month)         investment income
                    investmin(investment,month)      minimum bonds to buy
positive variables  endwrth               ending net worth
                    reinvest(month)       reinvestment income
                    cashflow(month)       cash withdrawn by authority
                    initcash              initial cash    ;
 equations          objt
                    money(month)                  money balance in a month
                    mintreas                      minimum in us treasuries
                    maxgovt(investment)           max in govt agencies
                    maxinprime                    maximum in prime commercial paper
                    maxindprim (investment)       maximum in one prime paper
                    mincash(month)                minimum cash
                    investmina(investment,month)  helps impose minimum bonds to buy
                    investminb(investment,month)  helps impose minimum bonds to buy
                    initalcash                      initial cash;
 investmina(investment,month)$(returndata(investment,month,"minreq") gt 0)..
     invest(investment,month)=l=invest.up(investment,month)*investmin(investment,month);
 investminb(investment,month)$(returndata(investment,month,"minreq") gt 0)..
     invest(investment,month)=g= returndata(investment,month,"minreq")
        *investmin(investment,month);
 initalcash$(docash eq 0)..    initcash=e=available;
 money(month)..
*     all investments in first month only
      sum((investment,months)$returndata(investment,months,"matureval")
      ,returndata(investment,months,"price")*invest(investment,months))$(ord(month) eq 1)
      +reinvest(month)$(ord(month) lt card(month))
     +endwrth$(ord(month) eq card(months))   +cashflow(month)$needcash(month)
      =e= sum(investment$returndata(investment,month,"matureval"),
                 invest(investment,month)*(returndata(investment,month,"matureval")
                 +returndata(investment,month,"reinvest")))
     +sum(investment$returndata(investment,month+6,"prior6"),
                 invest(investment,month+6)* returndata(investment,month+6,"prior6"))$
                 (ord(month) le card(month)-6)
     +sum(investment$returndata(investment,month+12,"prior12"),
                 invest(investment,month+12)* returndata(investment,month+12,"prior12"))$
                 (ord(month) le card(month)-12)
     + reinvest(month-1) *(1+reinvrate)**(1/12) $(ord(month) gt 1)
      +(initcash)$(ord(month) eq 1);
mincash(month)$needcash(month)..    cashflow(month) =g=needcash(month);
     objt.. obj=e=endwrth$(docash eq 0) -initcash$docash;
```

Handling Indivisibilities
Integer Programming Solution -- Branch and Bound
(secur.gms)

When solved with cplex

```
        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Node    ItCnt     Gap

      0     0   2.2303e+007    24                2.2303e+007        0
    100    93   2.2303e+007     1                2.2303e+007       53
    200   193   2.2303e+007     1                2.2303e+007       53
*   280+  266   2.2303e+007     0   2.2303e+007   2.2303e+007       53    0.00%
    300   270   2.2303e+007     1   2.2303e+007   2.2303e+007       53    0.00%
*   390    65   2.2303e+007     0   2.2303e+007   2.2303e+007      104    0.00%
Fixing integer variables, and solving final LP..
MIP Solution  :     22303062.100023    (104 iterations, 391 nodes)
Final LP      :     22303062.100023    (0 iterations)
Best integer solution possible :    22303113.765793
Absolute gap          :            51.6658
Relative gap          :       2.31653e-006
```

This report shows the Branch and Bound approach in action

No solution is found for a while (indicated by a blank entry in Best Integer until iteration 280), then one is found and another. The best node is lower bound, Best integer is incumbent. Note the last solution is not necessarily the global best. IInf tells the number of integer variables with noninteger solution level.

Node is the number of branch problems examined. Nodes left is the number of problems created during the branching process yet to be examined. Gap gives max percentage difference from the theoretical optimum.

MIP often ends with a gap between the solution found and the best possible. This is controlled by time and optcr/optca.