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



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 chocolate milk processed

Then we add a component to the model as follows



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. | $\sum\_{m}^{}a\_{im}X\_{m}$ -$\sum\_{k}^{}CAP\_{ik}Y\_{k}$ | ≤ 0 | for all i |
|  |  $X\_{m}$ | ≥ 0 | for all m |
|  | $$Y\_{k} $$ | ε(0,1) | for all k |

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



or if buying one meant we must buy another

Y1 -Y2 =0

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



Handling Indivisibilities

Integer Programming in GAMS

Maximize 7X1 -3X2 -10X3

 X1 -2X2  0

 X1 -20X3  0

 X1  0 X2  0 integer X3 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 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



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





Here

 Rounding not good

 Movement between points hard

Handling Indivisibilities

Integer Programming Solution Difficulty

Mixed Integer Programming Feasible Region X1 integer, X2 continuous



Handling Indivisibilities

Integer Programming Solution – Rounding

Solving the problem (solintr.gms)



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)



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



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

Handling Indivisibilities

Integer Programming Solution -- Branch and Bound

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 insuring 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 Branch and Bound approach in action

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

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

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