COLUMN GENERATION WITH GAMS

ERWIN KALVELAGEN

Abstract. This document describes an implementation of a *Column Generation* algorithm using GAMS. The well-known cutting stock problem is used as an example.

1. Introduction

In this paper we will use the *cutting stock problem* as an example how a problem-specific decomposition algorithm can be build in GAMS. The algorithm consist of two different models, a master and a sub-problem which exchange information. The master problem grows dynamically in size in this *column generation* algorithm. Such a structure can be conveniently implemented using dynamic sets in GAMS.

Although GAMS overhead will be large compared to an algorithm coded in a language like C or Fortran, the straightforward formulation and implementation of the algorithm in GAMS is highly suited for educational and prototypical situations. It is not unusual that algorithms designed and prototyped in a GAMS environment are later rewritten in a more traditional programming language, so it can be commercialized.

2. The cutting stock problem

The cutting stock problem can be described as follows. Assume d_i is the demand for products of length i. Stock is consisting of rolls that can be cut in patterns j. The possible patterns are denoted by $a_{i,j}$ being the number of products of length i that are the result of applying pattern j. The number of times pattern j is used is x_j. The cutting stock problem can now be formulated as the Mixed Integer Programming Model:

Date: 30 may 2003.

width (inches)	demand
12	211
31	395
36	610
45	97

Table 1. Demand data

$$\begin{aligned} \min \quad & \sum_j x_j \\ & \sum_j a_{i,j} x_j \geq d_i \\ & x_j \in \{0, 1, 2, ...\} \end{aligned} \tag{1}$$

Consider the data from [1]. The rolls to be cut are 100 inch wide and the demand data is shown in table 1.

A MIP model for this problem is trivially formulated in GAMS once we have enumerated all possible cutting patterns. Even for this extremely small example with four final widths we have 37 possible patterns.

Model cuttingstockmip.gms. [1]

```
$ontext


   Cutting Stock Example

   Erwin Kalvelagen, december 2002

   Data from Chvatal, Linear Programming, 1983.

$offtext


set
  i 'widths' /width1*width4/
  j 'patterns' /pattern1*pattern37/
;


*-----------------------------------------------------
* Data
*-----------------------------------------------------

scalar r 'raw width' /100/;

table demanddata(i,*)
           width  demand
  width1    45      97
  width2    36     610
  width3    31     395
  width4    14     211
;

table patterndata(j,i)
           width1  width2  width3  width4

pattern1     2
pattern2     1       1               1
pattern3     1       1
pattern4     1               1       1
pattern5     1               1
pattern6     1                       3
pattern7     1                       2
pattern8     1                       1
pattern9     1
pattern10            2               2
pattern11            2               1
pattern12            2
pattern13            1       2
pattern14            1       1       2
```

[1] http://www.gams.com/~erwin/colgen/cuttingstockmip.gms

```
pattern15           1       1       1
pattern16           1       1
pattern17           1               4
pattern18           1               3
pattern19           1               4
pattern20           1               1
pattern21           1
pattern22                   3
pattern23                   2       2
pattern24                   2       1
pattern25                   2
pattern26                   1       4
pattern27                   1       3
pattern28                   1       2
pattern29                   1       1
pattern30                   1
pattern31                           7
pattern32                           6
pattern33                           5
pattern34                           4
pattern35                           3
pattern36                           2
pattern37                           1
;


parameter w(i);
w(i) = demanddata(i,'width');

parameter d(i);
d(i) = demanddata(i,'demand');

parameter a(i,j);
a(i,j) = patterndata(j,i);

abort$(sum(j$(sum(i, a(i,j)*w(i)) > r+0.0001), 1)) "Pattern exceeds raw width";

*-----------------------------------------------------
* MIP formulation
*-----------------------------------------------------

integer variable x(j) 'patterns used';
variable z 'objective';

*
* default integer upperbound of 100 is too tight
*
x.up(j) = sum(i, d(i));

equations
    numpat    'number of patters used (objective)'
    demand(i) 'meet demand'
;

numpat..     z =e= sum(j, x(j));
demand(i)..  sum(j, a(i,j)*x(j)) =g= d(i);

model cut1 /numpat,demand/;
option optcr = 0.0;
solve cut1 using mip minimizing z;

display x.l;
```

3. Gilmore-Gomory column generation

The number of possible patterns is in general obviously very large, and therefore a delayed column generation algorithm is beneficial where only interesting patterns are considered. The Gilmore-Gomory algorithm[2, 3] is a famous column generation method for this problem.

Instead of enumerating all possible patterns j we start with a small initial set. An easy small initial pattern set would be for each final width w_i to use a complete roll of raw material. A slightly more advanced initial set of patterns is to use as many widths w_i that fit. I.e. if the raw width is $r = 100$, then pattern i would be to cut

$$\left\lfloor \frac{r}{w_i} \right\rfloor \tag{2}$$

equal widths. The notation $\lfloor x \rfloor$ is used to indicate the *floor* function which returns the largest integer not exceeding x. This results in an initial matrix:

```
----    190 PARAMETER aip  matrix growing in dimension p

               p1          p2          p3          p4

width1       2.000
width2                   2.000
width3                               3.000
width4                                           7.000
```

In the Gilmore-Gomory algorithm the cutting stock problem is not solved as a MIP but as an LP, i.e. the integer restrictions are relaxed. This model we will denote as the *Master Problem*.

To find a new pattern to add to the set of columns under consideration we look at the reduced cost σ_j of a column x_j. As is known from linear programming, the reduced costs are defined by:

$$\begin{aligned} \sigma_j &= c_j - \pi^T A_j \\ &= 1 - \pi^T A_j \end{aligned} \tag{3}$$

where π is the vector of duals of the constraint

$$\sum_j a_{i,j} x_j \geq d_i \tag{4}$$

For a column $x_j \geq 0$ to be eligible to enter the basis of a minimization problem, we must have $\sigma_j < 0$. The sub-problem of finding the possible pattern with the most negative reduced cost can be formulated as a special MIP problem, called a *knapsack problem*:

$$\begin{aligned} \min\ & 1 - \sum_i \pi_i y_i \\ & \sum_i w_i y_i \leq r \\ & y_i \in \{0, 1, 2, ...\} \end{aligned} \tag{5}$$

There are specialized algorithms for solving the knapsack problem very efficiently, including methods based on dynamic programming. The solution y of this problem forms a new column A_j in the Master Problem.

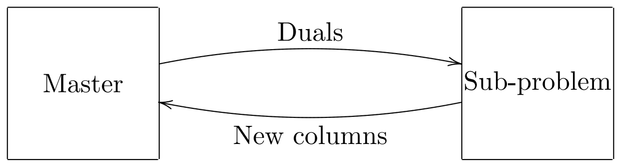

FIGURE 1. Communication between restricted master and sub-problems

The above algorithm finds a good subset of interesting columns which can then be used to find an integer solution by formulating and solving a MIP model:

$$\begin{aligned} &\min \sum_{j\in J} x_j \\ &\sum_{j\in J} a_{i,j}x_j \geq d_i \\ &x_j \in \{0,1,2,...\} \text{for } j \in J \end{aligned} \tag{6}$$

where J is the set of generated columns.

This algorithm will not always find the optimal solution of the original problem: it is possible a suboptimal solution is produced.

A complete algorithm formulated in GAMS can now be formulated as follows:

Model colgen.gms. [2]

```
$ontext

   Cutting Stock Example Using Column Generation

   Erwin Kalvelagen, december 2002

$offtext


set  i 'widths' /width1*width4/;


*-----------------------------------------------------
* Data
*-----------------------------------------------------

scalar r 'raw width' /100/;

table demanddata(i,*)
           width  demand
  width1    45      97
  width2    36     610
  width3    31     395
  width4    14     211
;


parameter w(i);
w(i) = demanddata(i,'width');

parameter d(i);
d(i) = demanddata(i,'demand');


*-----------------------------------------------------
* Gilmore-Gomory column generation algorithm
```

[2] http://www.gams.com/~erwin/colgen/colgen.gms

```
*-----------------------------------------------------

set p 'possible patterns' /p1*p1000/;
set iter 'maximum iterations' /iter1*iter25/;


*-----------------------------------------------------
* Master model
*-----------------------------------------------------

parameter aip(i,p) 'matrix growing in dimension p';
integer variable xp(p) 'patterns used';
variable z 'objective variable';

*
* default integer upperbound of 100 is too tight
*
xp.up(p) = sum(i, d(i));

equations
   master_numpat    'number of patterns used'
   master_demand(i) 'meet demand'
;

set pp(p) 'dynamic subset';

master_numpat..     z =e= sum(pp, xp(pp));
master_demand(i)..  sum(pp, aip(i,pp)*xp(pp)) =g= d(i);
model master /master_numpat,master_demand/;

* reduce amount of information written to the listing file
master.solprint = 2;
master.limrow = 0;
master.limcol = 0;
* faster execution of solve statements: keep gams in memory
master.solvelink = 2;


*-----------------------------------------------------
* Knapsack model
*-----------------------------------------------------

integer variables
   y(i)  'new pattern'
;
y.up(i) = ceil(r/w(i));

equations
   knapsack_obj
   knapsack_constraint
;

knapsack_obj..         z =e= 1 - sum(i, master_demand.m(i)*y(i));
knapsack_constraint..  sum(i, w(i)*y(i)) =l= r;
model knapsack /knapsack_obj,knapsack_constraint/;

knapsack.solprint = 2;
knapsack.solvelink = 2;
knapsack.optcr = 0;
knapsack.limrow = 0;
knapsack.limcol = 0;


*-----------------------------------------------------
* initialization
* get initial set pp and initial matrix aip
*-----------------------------------------------------
set pi(p);
pi('p1') = yes;
loop(i,
    aip(i,pi) = floor(r/w(i));
    pp(pi) = yes;
```

```
    pi(p) = pi(p-1);
);
display "--------------------------------------------------------------",
        "Initial value",
        "--------------------------------------------------------------",
        pp,aip;


scalar done /0/;
scalar iteration;

loop(iter$(not done),

*
* solve master problem
*
   solve master using rmip minimizing z;

*
* solve knapsack problem
*
   solve knapsack using mip minimizing z;

*
* new pattern found?
*
   if(z.l < -0.001,
      aip(i,pi) = y.l(i);
      pp(pi) = yes;
      iteration = ord(iter);
      display "--------------------------------------------------------------",
              iteration,
              "--------------------------------------------------------------",
              pp,aip;
      pi(p) = pi(p-1);
   else
      done = 1;
   );
);

abort$(not done) "Too many iterations.";


*
* solve final mip
*

display "--------------------------------------------------------------",
        "Final MIP",
        "--------------------------------------------------------------";

master.optcr=0;
solve master using mip minimizing z;
display z.l,xp.l;

parameter pat(*,*) "pattern usage";
pat(i,p)$(xp.l(p)>0.1) = aip(i,p);
pat('Count',p) = round(xp.l(p));
pat(i,'Total')= sum(p, aip(i,p)*round(xp.l(p)));
pat('Count','Total') = sum(p,pat('Count',p));
display pat;
```

Note that the master problem is expressed in terms of variables indexed by a dynamic set *pp*. This dynamic set will grow as long as the knapsack problem finds a column or pattern with a negative reduced cost.

GAMS does not allow that the variables are *declared* over a dynamic set. Therefore the declaration

```
integer variable xp(p) 'patterns used';
```

is over a static superset p which contains the largest possible dynamic subset pp.

When a new column is added to the Master Problem, we increase the set pp, using

```
      pp(pi) = yes;
```

after filling the column A_j with appropriate data with the statement:

```
      aip(i,pi) = y.l(i);
```

where `y.l` are the solution values of the sub-problem y. When the knapsack problem does not find any eligible columns anymore we are done.

The duals π_i are the marginals in GAMS terms. They are exchanged, just by using the marginals in the sub-problem objective:

```
knapsack_obj..         z =e= 1 - sum(i, master_demand.m(i)*y(i));
```

The final MIP model now can find the optimal solution of a smaller problem. Indeed, the final MIP has only six patterns in the model instead of 37 for the original mixed integer programming formulation.

The complete trace written to the listing file looks like:

```
----    114 --------------------------------------------------------------
            Initial value
            --------------------------------------------------------------

----    114 SET pp  dynamic subset

p1,    p2,    p3,    p4


----    114 PARAMETER aip  matrix growing in dimension p

                p1          p2          p3          p4

width1       2.000
width2                   2.000
width3                               3.000
width4                                           7.000

----    142 --------------------------------------------------------------
            PARAMETER iteration           =        1.000
            --------------------------------------------------------------

----    142 SET pp  dynamic subset

p1,    p2,    p3,    p4,    p5


----    142 PARAMETER aip  matrix growing in dimension p

                p1          p2          p3          p4          p5

width1       2.000
width2                   2.000                               2.000
width3                               3.000
width4                                           7.000       2.000

----    142 --------------------------------------------------------------
            PARAMETER iteration           =        2.000
            --------------------------------------------------------------

----    142 SET pp  dynamic subset

p1,    p2,    p3,    p4,    p5,    p6


----    142 PARAMETER aip  matrix growing in dimension p
```

```
                p1          p2          p3          p4          p5          p6

width1       2.000
width2                   2.000                                2.000       1.000
width3                               3.000                                2.000
width4                                           7.000        2.000

----    159 -------------------------------------------------------------
            Final MIP
            -------------------------------------------------------------
----    165 VARIABLE z.L                   =      453.000  objective variable

----    165 VARIABLE xp.L  patterns used

p1  49.000,    p2 100.000,    p5 106.000,    p6 198.000


----    169 PARAMETER pat  pattern usage

                p1          p2          p5          p6       Total

width1       2.000                                          98.000
width2                   2.000       2.000       1.000     610.000
width3                                           2.000     396.000
width4                               2.000                 212.000
Count       49.000     100.000     106.000     198.000     453.000
```

It shows how the dynamic set *pp* grows from the initial size of four to the final size of six. The coefficient matrix, represented by the parameter *aip* grows accordingly.

[1] contains a well-written chapter on this algorithm. See also [4] for a detailed description. For a GAMS implementation of delayed column generation in the context of Dantzig-Wolfe decomposition see [5].

References

1. V. Chvátal, *Linear programming*, Freeman, 1983.
2. P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting stock problem, Part I*, Operations Research **9** (1961), 849–859.
3. ______, *A linear programming approach to the cutting stock problem, Part II*, Operations Research **11** (1963), 863–888.
4. Robert W. Haessler, *Selection and design of heuristic procedures for solving roll trim problems*, Management Science **34** (1988), no. 12, 1460–1471.
5. Erwin Kalvelagen, *Dantzig-Wolfe Decomposition with GAMS*, `http://www.gams.com/~erwin/dw/dw.pdf`, May 2003.

GAMS Development Corp., Washington D.C.
E-mail address: `erwin@gams.com`