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 and a personnel planning problem are 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.

Finally, a more complicated personnel planning problem is solved used the same column generation framework as developed for the cutting stock problem.

For a GAMS implementation of delayed column generation in the context of Dantzig-Wolfe decomposition see [5].

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, 10 march 2011.

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

Table 1. Demand data

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

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://amsterdamoptimization.com/models/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.

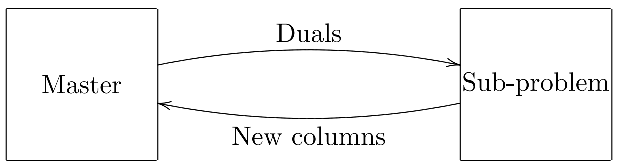


FIGURE 1. Communication between restricted master and sub-problems

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:

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

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://amsterdamoptimization.com/models/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.

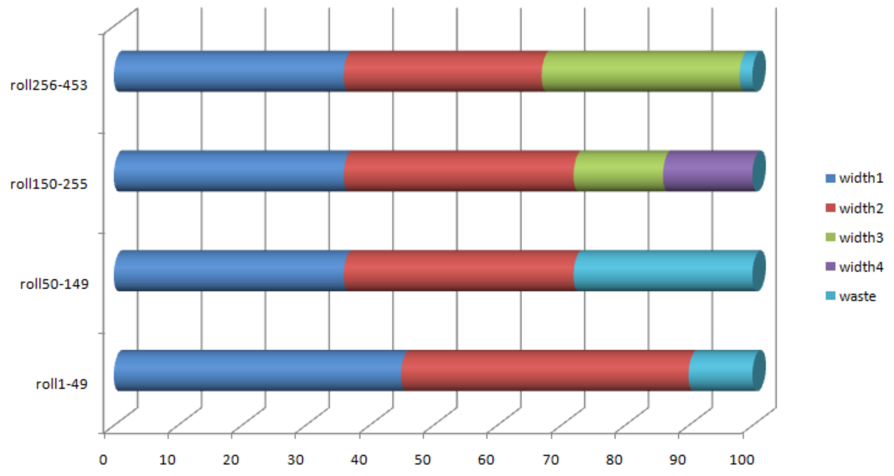


FIGURE 2. Final solution of cutting stock problem

A larger example data set for the above algorithm is:

```
*-----------------------------------------------------
* Data
*-----------------------------------------------------

scalar r 'raw width' /5600/;

table demanddata(i,*)
           width  demand
  width1    1380    22
  width2    1520    25
  width3    1560    12
  width4    1710    14
  width5    1820    18
  width6    1880    18
  width7    1930    20
  width8    2000    10
  width9    2050    12
  width10   2100    14
  width11   2140    16
  width12   2150    18
  width13   2200    20
;
```

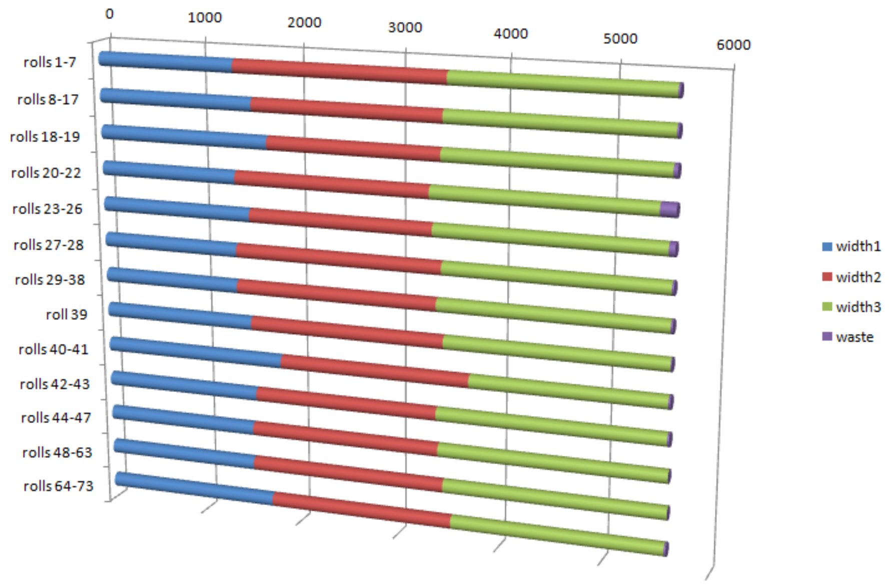


FIGURE 3. Final solution of cutting stock problem with larger data set

[1] contains a well-written chapter on this algorithm. See also [4] for a detailed description.

4. A WORKFORCE SCHEDULING PROBLEM

We follow here a problem described in [6]. A day schedule for call-center extends from 8:00AM through 20:00PM. We divide this up in 48 15-minute periods. The number of required operators for each quarter hour is estimated using an Erlang-C queueing model. The following rules apply when creating a shift for an operator:

- A day shift cannot exceed 9 hours, but is not allowed to be shorter than 4 hours.
- If a shift exceeds 8 hours, 0.75 break time is required of which 0.5 hour contiguous. If a shift exceeds 5.5 hours, a single break of 0.5 hours or two breaks of 0.25 hours is required. For shorter shifts a single break of 0.25 hours applies.

- Maximum contiguous time an operator can work without a break is 3.25 hours.

In this case the master problem looks like:

$$\begin{aligned} &\min \sum_j c_j x_j \\ &\sum_j a_{t,j} x_j \geq d_t \\ &x_j \in \{0,1,2,...\} \end{aligned} \tag{7}$$

where c_j is the number of quarter hours worked in shift x_j, and $a_{t,j}$ indicates whether in shift x_j the operator is working in period t.

The reduced cost $\sigma_j = c_j - \pi^T A_j$ for a new column A_j for the relaxed master problem will look like:

$$\sigma_j = c_j - \sum_t \pi_t a_{t,j} \tag{8}$$

This translates into an objective for the sub-problem as follows:

$$\min \sum_t w_t - \sum_t \pi_t w_t = \sum_t (1-\pi_t) w_t \tag{9}$$

where the binary decision variable w_t indicates whether the operator is working in period t.

The complete model can look like:

Model workforcescheduling.gms. [3]

```
$ontext

   Column generation for work force scheduling

   Erwin Kalvelagen
   erwin@amsterdamoptimization.com

   References:
       Annemieke van Dongen, "Personeelsplanning en kolomgeneratie",
          Vrije Universiteit Amsterdam, 2005

$offtext

set t 'periods' /t1*t48/;

parameter req(t) 'required number of employees' /
  t1  2,  t2  5,  t3  6,  t4  5,  t5  7,  t6  7,  t7  4,  t8  6,  t9  7,  t10 7
  t11 7,  t12 7,  t13 6,  t14 5,  t15 6,  t16 6,  t17 5,  t18 4,  t19 7,  t20 6
  t21 6,  t22 4,  t23 5,  t24 5,  t25 6,  t26 6,  t27 5,  t28 7,  t29 7,  t30 6
  t31 9,  t32 6,  t33 7,  t34 6,  t35 6,  t36 7,  t37 7,  t38 4,  t39 5,  t40 5
  t41 4,  t42 5,  t43 3,  t44 4,  t45 5,  t46 3,  t47 5,  t48 4 /;


set
  slen 'shift lengths' /short, medium, long/
;

table shiftlen(slen,*)  'lengths of shifts in 0.25 hours'
           minwork maxwork  sumbreak  maxnumbreak
  short     16      22        1          1
  medium    23      32        2          2
  long      33      36        3          2
;
```

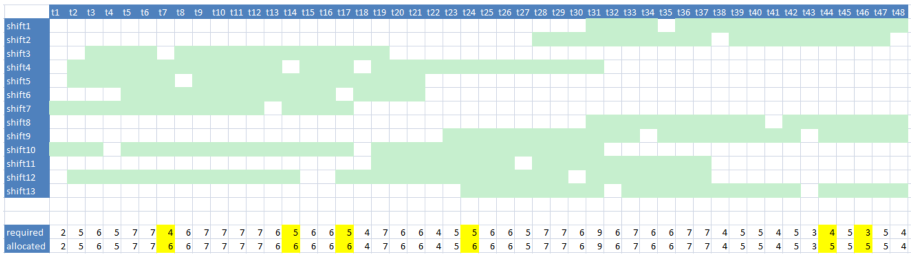
[3]http://amsterdamoptimization.com/models/colgen/workforcescheduling.gms

```
scalar maxwlen 'max periods of contiguous work' /13/;

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

sets
   s    'possible shifts' /shift1*shift1000/
   iter 'maximum iterations' /iter1*iter1000/
;

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

parameters
   a(t,s)   'matrix growing in dimension s'
   c(s)     'cost vector growing in dimension s'
;
variables
   x(s)    'shifts used'
   z       'objective variable'
;
integer variable x;

x.up(s) = sum(t, req(t));

set ds(s) 'dynamic subset';

equations
  masterobj  'cost: total number of periods worked'
  edemand(t) 'meet required manpower'
;

masterobj..  z =e= sum(ds,c(ds)*x(ds));
edemand(t).. sum(ds, a(t,ds)*x(ds)) =g= req(t);

model master /masterobj,edemand/;

* reduce amount of information written to the listing file
master.solprint = 2;
master.limrow = 0;
master.limcol = 0;
master.solvelink = 2;

*-----------------------------------------------------
* Sub model
*-----------------------------------------------------

variables
  sigma       'reduced cost'
  work(t)     'new shift: quarters worked'
  shift(t)    'operator is present'
  break(t)    'operator is on break'
  totwork     'number of periods worked'
  totbreak    'break time'
  startshift(t) 'shift starts'
  startbreak(t) 'break starts'
  tshift(slen) 'type of shift: short, medium, long'
;
binary variables work, present, break, startshift, startbreak, tshift;
positive variables totwork, totbreak, totpresent;

* relax: these variables are automatically integer
break.prior(t) = INF;
startbreak.prior(t) = INF;
startshift.prior(t) = INF;

equations
  subobj        'objective function'
  eshift(t)     'shift = work or break'
  etotwork      'calculate total quarter hours worked'
  etotbreak     'calculate total quarter hours used for breaks'
  estshift(t)   'start of shift'
  estbreak(t)   'start of break'
  estartsh1     'start shift once'
  etshift1      'limit work for short and long shifts'
  etshift2      'limit work for short and long shifts'
  etshift3      'limit breaks for short and long shifts'
  ebreak1       'required break time'
  ebreak2       'limits on number of breaks (forces longer breaks)'
  emaxcont(t)   'max contiguous working time'
;

subobj..       sigma =e= sum(t, (1-edemand.m(t))*work(t));

eshift(t)..    shift(t) =e= work(t) + break(t);

etotwork..     totwork =e= sum(t, work(t));

etotbreak..    totbreak =e= sum(t, break(t));

estshift(t)..  startshift(t) =g= shift(t) - shift(t-1);

estbreak(t)..  startbreak(t) =g= break(t) - break(t-1);

estartsh1..    sum(t, startshift(t)) =e= 1;

etshift1..     totwork =g= sum(slen, shiftlen(slen,'minwork')*tshift(slen));

etshift2..     totwork =l= sum(slen, shiftlen(slen,'maxwork')*tshift(slen));

etshift3..     sum(slen, tshift(slen)) =e= 1;

ebreak1..      totbreak =e= sum(slen, shiftlen(slen,'sumbreak')*tshift(slen));

ebreak2..      sum(t, startbreak(t)) =l= sum(slen, shiftlen(slen,'maxnumbreak')*tshift(slen));

alias (t,tt);
set tmaxw(t,tt);
tmaxw(t,tt)$(ord(tt)>=ord(t) and ord(tt)<=ord(t)+maxwlen and ord(t)<=card(t)-maxwlen) = yes;
display tmaxw;

emaxcont(t)$sum(tmaxw(t,tt),1).. sum(tmaxw(t,tt), work(tt)) =l= maxwlen;


model sub /
  subobj, eshift, etotwork, etotbreak, estshift, estbreak,
  estartsh1, etshift1, etshift2, etshift3, ebreak1, ebreak2,
  emaxcont
/;

equation edummy;
variable dummy;

edummy.. dummy =e= 0;

model subcheck /
  edummy, eshift, etotwork, etotbreak, estshift, estbreak,
  estartsh1, etshift1, etshift2, etshift3, ebreak1, ebreak2,
  emaxcont
/;

* reduce amount of information written to the listing file
sub.solprint = 2;
sub.limrow = 0;
sub.limcol = 0;
sub.solvelink = 2;
sub.optcr=0;

subcheck.solprint = 2;
subcheck.limrow = 0;
subcheck.limcol = 0;
subcheck.solvelink = 2;


*-----------------------------------------------------
* initial feasible solution
* invent some candidate shifts
*-----------------------------------------------------

table init(t,s) '1=work,2=break'
        shift1
  t1      1
  t2      1
  t3      1
  t4      1
  t5      1
  t6      1
  t7      1
  t8      1
  t9      1
  t10     1
  t11     1
  t12     2
  t13     1
  t14     1
  t15     1
  t16     1
  t17     1
  t18     1
  t19     1
  t20     1
  t21     1
  t22     1
  t23     1
;


set s2(s) /shift2*shift26/;

loop(s2(s),
   init(t,s) = init(t-1,s-1);
);
display init;

*
* check initial solution.
*
loop(s$sum(t$init(t,s),1),
   shift.fx(t)$init(t,s) = 1;
   work.fx(t)$(init(t,s)=1) = 1;
   break.fx(t)$(init(t,s)=2) = 1;
   solve subcheck minimizing dummy using mip;
   abort$(subcheck.modelstat<>1 and subcheck.modelstat<>8) "Initial solution not correct";
   shift.lo(t) = 0;   shift.up(t) = 1;
   work.lo(t) = 0;    work.up(t) = 1;
   break.lo(t) = 0;   break.up(t) = 1;
);

ds(s)$sum(t$init(t,s),1) = 1;
a(t,ds)$(init(t,ds)=1) = 1;
c(ds) = sum(t$(init(t,ds)=1),1);

set nexts(s);
nexts(s)$(ord(s)=card(ds)+1) = yes;


option a:0;
option c:0;
display
  "------------------------------------------------"
  "initial solution",
  "------------------------------------------------",
  a,c,ds,nexts;
```

```
*-----------------------------------------------------
* column generation algorithm starts here
*-----------------------------------------------------


scalar done /0/;
scalar iteration;

loop(iter$(not done),

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

*
* solve knapsack problem
*
   solve sub using mip minimizing sigma;

*
* new column found?
*
   if(sigma.l < -0.001,
      a(t,nexts) = work.l(t);
      c(nexts) = sum(t, work.l(t));
      ds(nexts) = yes;
      iteration = ord(iter);
      display "-------------------------------------------------------------",
              iteration,
              "-------------------------------------------------------------",
              ds,a;
      nexts(s) = nexts(s-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,x.l;

*--------------------------------------------------------------
* reporting
*--------------------------------------------------------------


scalars
   totreq 'total required periods'
   totusd 'total allocated periods'
;

totreq = sum(t, req(t));
totusd = z.l;

display totreq, totusd;

alias(s,s3);
set curs(s3) /shift1/;
```

```
scalar k;
parameter schedule(s,t);
loop(s$(x.l(s)>0.5),
   for(k=1 to round(x.l(s)),
       schedule(curs,t) = a(t,s);
       curs(s3) = curs(s3-1);
   );
);
option schedule:0;
display schedule;
```

With this formulation and this set of initial shifts we get a solution with 275 periods worked. The total required amount of work is 267 periods, so the overhead due to regulations is 8 periods. [4]



	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15	t16	t17	t18	t19	t20	t21	t22	t23	t24
shift1																								
shift2																								
shift3			🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩					
shift4		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩
shift5		🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
shift6					🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩			
shift7	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩							
shift8																								
shift9																							🟩	🟩
shift10	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩
shift11																			🟩	🟩	🟩	🟩	🟩	🟩
shift12		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
shift13																								🟩
required	2	5	6	5	7	7	4	6	7	7	7	7	6	5	6	6	5	4	7	6	6	4	5	5
allocated	2	5	6	5	7	7	6	6	7	7	7	7	6	6	6	6	6	4	7	6	6	4	5	6

	t25	t26	t27	t28	t29	t30	t31	t32	t33	t34	t35	t36	t37	t38	t39	t40	t41	t42	t43	t44	t45	t46	t47	t48
shift1							🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
shift2				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
shift3																								
shift4	🟩	🟩	🟩	🟩	🟩	🟩	🟩																	
shift5																								
shift6																								
shift7																								
shift8							🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩
shift9	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩
shift10	🟩	🟩	🟩	🟩	🟩	🟩	🟩																	
shift11	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩											
shift12	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩											
shift13	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩	🟩
required	6	6	5	7	7	6	9	6	7	6	6	7	7	4	5	5	4	5	3	4	5	3	5	4
allocated	6	6	5	7	7	6	9	6	7	6	6	7	7	4	5	5	4	5	3	5	5	5	5	4

FIGURE 4. Column generation solution of call center workforce scheduling problem

For these type of rostering problems it is often not too difficult to enumerate all possible shifts. In our case we ended up with a data file with 20579 shifts. The resulting master model solves very quickly, and proves the optimal solution for this problem is indeed 275.

Model workforceallshifts.gms. [5]

```
$ontext

   Column generation for work force scheduling

   Erwin Kalvelagen
   erwin@amsterdamoptimization.com

   References:
       Annemieke van Dongen, "Personeelsplanning en kolomgeneratie",
          Vrije Universiteit Amsterdam, 2005

$offtext

set
   t 'periods' /t1*t48/
   s 'possible shifts' /k1*k20579/
;

parameter req(t) 'required number of employees' /
  t1  2,  t2  5,  t3  6,  t4  5,  t5  7,  t6  7,  t7  4,  t8  6,  t9  7,  t10 7
  t11 7,  t12 7,  t13 6,  t14 5,  t15 6,  t16 6,  t17 5,  t18 4,  t19 7,  t20 6
  t21 6,  t22 4,  t23 5,  t24 5,  t25 6,  t26 6,  t27 5,  t28 7,  t29 7,  t30 6
  t31 9,  t32 6,  t33 7,  t34 6,  t35 6,  t36 7,  t37 7,  t38 4,  t39 5,  t40 5
```

[4]In [6] a solution of 274 is reported. We have interpreted the restrictions in a more strict way, and as a result we reject some of the proposed shifts.

[5]Model: `http://amsterdamoptimization.com/models/colgen/workforceallshifts.gms`, Data file: `http://amsterdamoptimization.com/models/colgen/AllShifts.inc`

```
  t41 4,  t42 5,  t43 3,  t44 4,  t45 5,  t46 3,  t47 5,  t48 4 /
;

table data(s,t)
$include AllShifts.inc
;


parameters
   a(t,s)   'matrix'
   c(s)     'cost vector'
;

a(t,s)$(data(s,t)=1) = 1;
c(s) = sum(t,a(t,s));

variables
   x(s)    'shifts used'
   z       'objective variable'
;
integer variable x;

x.up(s) = sum(t, req(t));

equations
  masterobj  'cost: total number of periods worked'
  edemand(t) 'meet required manpower'
;

masterobj..  z =e= sum(s,c(s)*x(s));
edemand(t).. sum(s, a(t,s)*x(s)) =g= req(t);

model master /masterobj,edemand/;

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

*---------------------------------------------------------------
* reporting
*---------------------------------------------------------------


scalars
   totreq 'total required periods'
   totusd 'total allocated periods'
;

totreq = sum(t, req(t));
totusd = z.l;

display totreq, totusd;

alias(s,s3);
set curs(s3) /k1/;
scalar k;
parameter schedule(s,t);
loop(s$(x.l(s)>0.5),
   for(k=1 to round(x.l(s)),
       schedule(curs,t) = a(t,s);
       curs(s3) = curs(s3-1);
   );
);
option schedule:0;
display schedule;
```

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://amsterdamoptimization.com/pdf/dw.pdf`, May 2003.
6. Annemieke van Dongen, *Personeelsplanning en Kolomgeneratie*, Vrije Universiteit Amsterdam, November 2005.

Amsterdam Optimization Modeling Group LLC, Washington, D.C.
E-mail address: `erwin@amsterdamoptimization.com`