Optimization Solutions to Some Typical Problems

D'Souza, Ajay
ajaydsouza@gatech.edu

Some Linear Optimization methods for solving some typical problems

Abstract

Description of the Results and Methods of Optimization methods of Column Generation, Kantorovich Formulation, Gilmore-Gomory Formulation and Dantzig-Wolfe method to some typical problems

Contents

List of Figures

List of Tables

1 Optimum Nutrition Based Diet- Column Generation Method

1.1 1

The following is the implementation of the column generation code for the Diet Problem in file *diet.colgen.partial.mos*

```
model DietGen ! Name the model

uses "mmxprs" ! include the Xpress solver package
!uses "mmodbc" ! include package to read from Excel
uses "mmsheet"  ! using mmsheet as he student license, have does not have mmodbc

NumFoods := 7035 ! declare how many foods we have
NumNutrients := 30 ! declare how many nutrients we're tracking consumption of

declarations ! declare sets and arrays

Foods = 1..NumFoods+1  ! We need an extra "dummy" food as we'll see later
Nutrients = 1..NumNutrients

FoodNames: array(Foods) of string ! names of each food
NutNames: array(Nutrients) of string ! names of each nutrient

Chol: array(Foods) of real ! vector of cholesterol for each food
Calorie: array(Foods) of real ! vector of calorie for each food
Contents: array(Foods,Nutrients) of real ! matrix of food content
Minimums: array(Nutrients) of real ! minimum intake of each nutrient
Maximums: array(Nutrients) of real   ! maximum intake of each nutrient

Eaten: dynamic array(Foods) of mpvar ! variables: amount of each food eaten
! "dynamic" means we'll be generating variables
! instead of defining them all up front


minShadowPrices: array(Nutrients) of real ! array to hold the shadow prices of the minim
maxShadowPrices: array(Nutrients) of real ! array to hold the shadown prices of the maxi
```

```
ColsGen: integer ! number of columns generated in the problem so far
BestReducedCost: real ! what is the best reduced cost
BestFood: integer ! which food has the best reduced cost
TotalCalorie: real ! total calorie

EatMin,EatMax:array(Nutrients) of linctr !  min and max linear constraints
eatenConstraints: dynamic array(Foods) of linctr ! Not null constraints for x variables
TotalChol: linctr            ! The objective constraint
objectiveVal: real           ! the objective value of min cholestrol
dualEatMin: array(Nutrients) of real    ! dual variables for min constraints
dualEatMax: array(Nutrients) of real    ! dual variables for max constraints
minCons,maxCons: real          ! hold the reduction values for Y*A(j) - min and max cost

end-declarations ! end declarations section


!initializations from "mmodbc.odbc:diet.xls" ! read from Excel file
initializations from "mmsheet.xls:noindex;diet.xls" ! read from Excel file using mmsheet

 FoodNames as "[a3:a7037]" ! Pls note that the following comments are relevant to
  ! mmodbc only, mmsheet reads from the line specified
  !
  ! NOTE: In the Excel file, the food names
  ! are in cells a3:a7148, not a2:a7148.
  ! For some reason, Xpress makes you always
! include one row of cells above the data you
! want to read.
! You'll see the same extra row in all of
! these other things read from Excel.

 NutNames as "[b2:ae2]"
 Chol as "[ac3:ac7037]"
 Contents as "[b3:ae7037]"
 Minimums as "[b7039:ae7039]"
 Maximums as "[b7041:ae7041]"
 Calorie as "[d3:d7037]"

end-initializations
```

```
! We need to start with at least one column or else the problem
! will start off infeasible.  We'll use a "dummy" food as our
! initial column.  The dummy food will have a very high cost (1,000,000)
! and will contain all of the minimums necessary - that way we can be sure
! that by itself it will a be feasible solution.

! It is hard to manually pick the combination of foods which will give the minimum nutri
!  So we employ the strategy of creating a dummy food which has the minimum nutrient lev
!  each nutrient
!  BUT the cholestrol level fro this food ( whcih is the objective to be minimized)
!  is very very high
! This gives us a manually selected food which is feasible to begin with
forall(k in Nutrients)
Contents(NumFoods+1,k) := Minimums(k) ! create contents of "dummy" food

FoodNames(NumFoods+1) := "DUMMY FOOD"
Chol(NumFoods+1) := 1000000


! Create the new (dummy food) variable.  Because the variables were
! declared as "dynamic" none of the other variables exist in the problem yet.
! you will need this command to generate new variables later
create(Eaten(NumFoods+1))

! Create minimum and maximum intake constraints.
! If all the variables existed, we could say
!       sum(j in Foods) Contents(j,k)*Eaten(j) <= Maximums(k).
! However, not all the variables Eaten(j) are created yet, so as new ones
! get created we'll have to add them to the constraints.
!   Below, we add the one variable we've created so far (the dummy variable) to
! each constraint.
!   These constraints need to be named ("EatMin(k)" and "EatMax(k)") because
! we'll need to get the dual variables for them later, and for us to be able to
! ask Xpress for the dual variable, we need a way to refer to the constraints.

forall(k in Nutrients) do
EatMin(k) := Contents(NumFoods+1,k)*Eaten(NumFoods+1) >= Minimums(k)
EatMax(k) := Contents(NumFoods+1,k)*Eaten(NumFoods+1) <= Maximums(k)
end-do

! Create objective function.  Just like the intake constraints, we'll have
! to add variables to it as they get generated.
TotalChol := Chol(NumFoods+1)*Eaten(NumFoods+1)


!!!!!!!!!!! Begin column generation !!!!!!!!!!!!!!!!
ColsGen := 0
repeat

! solve the problem using the variables that have been generated so far
!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!

! minimize the objective
minimize(TotalChol)

! get the minimum cholestrol the objective
objectiveVal := getobjval
writeln("Total Cholesterol = ",objectiveVal)

! get the dual variable (cB*Binverse)
!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!

! get the dual for each of the two sets of min/max constraints
! for each nutrient
forall(i in Nutrients) dualEatMin(i):=getdual(EatMin(i))
forall(i in Nutrients) dualEatMax(i):=getdual(EatMax(i))


! Find the food with the best reduced cost.
!   As we search through each food, we'll keep track of the best reduced
! cost found so far in BestReducedCost, and whatever food has that best
! reduced cost will be stored in BestFood.

BestReducedCost := 0
BestFood := 0
forall(j in Foods) do

! if food j has a better (lower) reduced cost than the previous best,
! then store its reduced cost in BestReducedCost and let BestFood be j.
!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!

! get Y^T*A_j for each A_j
```

```
minCons := sum(i in Nutrients) dualEatMin(i)*Contents(j,i)
maxCons := sum(i in Nutrients) dualEatMax(i)*Contents(j,i)

if ( BestReducedCost > (Chol(j) - minCons - maxCons) ) then
BestReducedCost := Chol(j) - minCons - maxCons
BestFood := j
end-if

end-do

! If there's a variable with a good reduced cost, then
!  create the variable
! update the number of columns generated
! update the constraints for each nutrient by adding the contents of the new food: const
! update the objective function by adding the cost of the new food: similar here

if BestReducedCost < -0.000001 then
writeln("Adding food ",BestFood," (",FoodNames(BestFood),") : reduced cost is ",BestRedu
!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!

! Create the variable for the food to be added
create(Eaten(BestFood))

! increment the columns in the basis
ColsGen+=1

! append the variable for the BestFood found to all the min and max
! nutrient constraints
forall(k in Nutrients) do
EatMin(k) += Contents(BestFood,k)*Eaten(BestFood)
EatMax(k) += Contents(BestFood,k)*Eaten(BestFood)
end-do

! add the BestFood variable to the objective function
TotalChol+= Chol(BestFood)*Eaten(BestFood)

end-if


! Here and above ("if BestReducedCost < -0.000001") we use 0.000001 instead of 0.
! The reason is that floating point roundoff errors might make the computer
! calculate a reduced cost to be something like 0.000000000001 instead of 0.  This
! is normal -- most computer software has this sort of problem, and they use the
! same sort of checks to catch it.  Specifically, any value of x for which, say,
! -0.000000001 < x < 0.000000001 will be treated as 0.  For this application, we
! can safely use 0.000001 instead.
!   This is really a CS/computer architecture issue; if you're interested in
! learning more about what causes it, let me know and I'd be happy to explain more.

until BestReducedCost >= -0.000001 ! continue until no variable has a good reduced cost


!!!!!!!!!!!! End column generation !!!!!!!!!!!!!!!!!


! The solution can be long, so print a summary: only say what we eat.
TotalCalorie := 0
writeln("\n===============\n")
forall(j in Foods)
if (getsol(Eaten(j)) > 0) then
writeln(FoodNames(j)," = ",getsol(Eaten(j)))

!!! you need to compute the total calorie for the chosen food !!!
!!!!!!!!! write your code here !!!!!!!!!!
TotalCalorie += Calorie(j)*getsol(Eaten(j))


end-if


writeln("\nTotal Cholesterol = ",getobjval)
writeln("\n Total Calorie = ", TotalCalorie)
writeln("\nTotal: ",ColsGen," columns generated.")

end-model
```

1.2 2

The following is the output from *diet.colgen.partial.mos*. The file shows the food selected during each step of column generation. The final optimum list of food selected for minimizing cholesterol along with their total calorie count are printed at the bottom.

```
Total Cholesterol = 1e+006
Adding food 633 (Fish oil, menhaden, fully hydrogenated) : reduced cost is -4.7795e+006
Total Cholesterol = 1e+006
Adding food 219 (Spices, sage, ground) : reduced cost is -2.14313e+007
Total Cholesterol = 1e+006
Adding food 4432 (Mollusks, clam, mixed species, cooked, moist heat) : reduced cost is -
Total Cholesterol = 1e+006
Adding food 630 (Fish oil, cod liver) : reduced cost is -2.49994e+007
Total Cholesterol = 1e+006
Adding food 6706 (Whale, beluga, eyes, raw (Alaska Native)) : reduced cost is -7.83319e+
Total Cholesterol = 999941
Adding food 5772 (Sugars, granulated) : reduced cost is -528622
Total Cholesterol = 983495
Adding food 4243 (Water, bottled, non-carbonated, CRYSTAL GEYSER) : reduced cost is -101
Total Cholesterol = 970330
Adding food 5291 (Leavening agents, yeast, baker's, active dry) : reduced cost is -1.593
Total Cholesterol = 949779
Adding food 5289 (Leavening agents, baking soda) : reduced cost is -2.12105e+007
Total Cholesterol = 865617
Adding food 4218 (Orange-flavor drink, KRAFT, TANG SUGAR FREE Low Calorie Drink M) : red
Total Cholesterol = 811206
Adding food 4441 (Mollusks, oyster, eastern, wild, cooked, moist heat) : reduced cost is
Total Cholesterol = 718728
Adding food 3333 (Seeds, cottonseed meal, partially defatted (glandless)) : reduced cost
Total Cholesterol = 627582
Adding food 4197 (Tea, instant, unsweetened, powder, decaffeinated) : reduced cost is -6
Total Cholesterol = 507300
Adding food 4594 (Soy protein isolate, PROTEIN TECHNOLOGIES INTERNATIONAL, SUPRO) : redu
Total Cholesterol = 497085
Adding food 3446 (Nuts, mixed nuts, without peanuts, oil roasted, with salt added) : red
Total Cholesterol = 432518
Adding food 5288 (Leavening agents, baking powder, low-sodium) : reduced cost is -5.5393
Total Cholesterol = 77.97
```

```
Adding food 685 (Oil, industrial, coconut (hydrogenated), used for whipped toppi) : reduced co
Total Cholesterol = 29.1
Adding food 1666 (Cereals ready-to-eat, KELLOGG, KELLOGG'S Complete Wheat Bran Fl) : reduced c
Total Cholesterol = 19.0471
Adding food 4315 (Fish, herring, Atlantic, raw) : reduced cost is -32.4521
Total Cholesterol = 14.0516
Adding food 3039 (Carrot, dehydrated) : reduced cost is -34.6291
Total Cholesterol = 12.3829
Adding food 1958 (Cereals, QUAKER, Instant Oatmeal, NUTRITION FOR WOMEN, Vanilla ) : reduced c
Total Cholesterol = 0

===============

Spices, sage, ground = 0.0358466
Oil, industrial, coconut (hydrogenated), used for whipped toppi = 0.112111
Cereals ready-to-eat, KELLOGG, KELLOGG'S Complete Wheat Bran Fl = 0.302161
Cereals, QUAKER, Instant Oatmeal, NUTRITION FOR WOMEN, Vanilla  = 1.04767
Seeds, cottonseed meal, partially defatted (glandless) = 0.00290843
Nuts, mixed nuts, without peanuts, oil roasted, with salt added = 0.882755
Orange-flavor drink, KRAFT, TANG SUGAR FREE Low Calorie Drink M = 0.023294
Water, bottled, non-carbonated, CRYSTAL GEYSER = 9.86139
Soy protein isolate, PROTEIN TECHNOLOGIES INTERNATIONAL, SUPRO = 0.299797
Leavening agents, baking powder, low-sodium = 0.239075
Sugars, granulated = 1.65535
Whale, beluga, eyes, raw (Alaska Native) = 0.00362453

Total Cholesterol = 0

 Total Calorie = 1916.54

Total: 21 columns generated.
```

1.3 3

The following is the output from *diet.colgen.partial.mos* with an additional constraint of total calorie intake between 1800 and 2200 calories. The food selected at each step of the column generation algorithm are shown. The optimum list of foods selected to minimize cholesterol while at the same time meeting the additional calorie intake constraint are listed at the bottom of the output.

```
Total Cholesterol = 1e+006
Adding food 633 (Fish oil, menhaden, fully hydrogenated) : reduced cost is -4.7795e+006
Total Cholesterol = 1e+006
Adding food 219 (Spices, sage, ground) : reduced cost is -2.14313e+007
Total Cholesterol = 1e+006
Adding food 4432 (Mollusks, clam, mixed species, cooked, moist heat) : reduced cost is -
Total Cholesterol = 1e+006
Adding food 630 (Fish oil, cod liver) : reduced cost is -2.49994e+007
Total Cholesterol = 1e+006
Adding food 6706 (Whale, beluga, eyes, raw (Alaska Native)) : reduced cost is -7.83319e+
Total Cholesterol = 999941
Adding food 5772 (Sugars, granulated) : reduced cost is -528622
Total Cholesterol = 983541
Adding food 4243 (Water, bottled, non-carbonated, CRYSTAL GEYSER) : reduced cost is -101
Total Cholesterol = 970330
Adding food 5291 (Leavening agents, yeast, baker's, active dry) : reduced cost is -1.593
Total Cholesterol = 949779
Adding food 5289 (Leavening agents, baking soda) : reduced cost is -2.12105e+007
Total Cholesterol = 865617
Adding food 4218 (Orange-flavor drink, KRAFT, TANG SUGAR FREE Low Calorie Drink M) : red
Total Cholesterol = 811206
Adding food 4441 (Mollusks, oyster, eastern, wild, cooked, moist heat) : reduced cost is
Total Cholesterol = 718728
Adding food 3333 (Seeds, cottonseed meal, partially defatted (glandless)) : reduced cost
Total Cholesterol = 627983
Adding food 4197 (Tea, instant, unsweetened, powder, decaffeinated) : reduced cost is -6
Total Cholesterol = 507598
Adding food 4594 (Soy protein isolate, PROTEIN TECHNOLOGIES INTERNATIONAL, SUPRO) : redu
Total Cholesterol = 502222
Adding food 3446 (Nuts, mixed nuts, without peanuts, oil roasted, with salt added) : red
Total Cholesterol = 432975
Adding food 5288 (Leavening agents, baking powder, low-sodium) : reduced cost is -5.5532
Total Cholesterol = 76.9936
Adding food 685 (Oil, industrial, coconut (hydrogenated), used for whipped toppi) : redu
Total Cholesterol = 28.7742
Adding food 1838 (Cereals ready-to-eat, KELLOGG, KELLOGG'S Complete Oat Bran Flak) : red
Total Cholesterol = 19.044
```

```
Adding food 4315 (Fish, herring, Atlantic, raw) : reduced cost is -32.4518
Total Cholesterol = 15.6761
Adding food 3039 (Carrot, dehydrated) : reduced cost is -35.113
Total Cholesterol = 12.4005
Adding food 1958 (Cereals, QUAKER, Instant Oatmeal, NUTRITION FOR WOMEN, Vanilla ) : reduced c
Total Cholesterol = 0

===============

Spices, sage, ground = 0.0440682
Oil, industrial, coconut (hydrogenated), used for whipped toppi = 0.179782
Cereals ready-to-eat, KELLOGG, KELLOGG'S Complete Oat Bran Flak = 0.136139
Cereals, QUAKER, Instant Oatmeal, NUTRITION FOR WOMEN, Vanilla  = 1.11386
Seeds, cottonseed meal, partially defatted (glandless) = 0.201204
Nuts, mixed nuts, without peanuts, oil roasted, with salt added = 0.0981756
Orange-flavor drink, KRAFT, TANG SUGAR FREE Low Calorie Drink M = 0.162746
Water, bottled, non-carbonated, CRYSTAL GEYSER = 9.85867
Soy protein isolate, PROTEIN TECHNOLOGIES INTERNATIONAL, SUPRO = 0.832635
Leavening agents, baking powder, low-sodium = 0.205749
Sugars, granulated = 1.73092
Whale, beluga, eyes, raw (Alaska Native) = 0.00550088

Total Cholesterol = 0

 Total Calorie = 1808.09

Total: 21 columns generated.
```

2 Cutting Stock Problem - A take on the Knapsack problem

For the Kantorovich formulation we try to determine if there is a better bound than 500 on K as shown below. Here K the maximum number of rolls.

```
! determing if there is a better bound than 500 on K value
K := 0
forall ( i in iRange) do
```

```
K += ceil( b(i) / (floor(W/(w(i)))) )
end-do
if ( K > 500 ) then
K := 500
end-if
```

2.1 Kantorovich Method - Constraint Generation

2.1.1 1

If we have K as the bound on the maximum number of rolls to be considered for cutting and if m is the number different widths to be cut, then Kantorovich will have a total of $K + m$ constraints. This is in addition to the bounds and integrality constraints. So the total number of decision variables for these constraints will be $K \times y_k$ binary variables to indicate if a roll is cut or not and $m \times K \times x_{ik}$ variables which will indicate how many widths i are cut from roll k. This make a total of
$K + m \times K = K(1 + m)$ decision variables

In comparison the Gilmore-Gomory column generation method has only a fixed number of m constraints, which is the number of different widths to be cut. If the number of patterns selected by the column generation method is g, then the number of decision variables will also be just g. In general the number of patterns g generated by column generation method is much smaller than K, which is a bound on the maximum number of rolls considered in Kantorovich. So column generation method has far fewer constraints and far fewer decision variables then the Kantorovich formulation.

2.1.2 2

The Kantorovich formulation is implemented in file $cs.Kantorovich.partial.mos$. The following is the output of the program for data file $kant1.dat$. The optimum value for the minimum number of rolls is 22. The output lists the 22 rolls to be cut along with the widths to be cut in each roll and any unused width in each roll

```
obj=22
y(1)=1
1 copies of width 25
1 copies of width 35
2 copies of width 45
unused width=0


y(2)=1
6 copies of width 25
unused width=0


y(4)=1
5 copies of width 12
2 copies of width 45
unused width=0


y(5)=1
1 copies of width 25
1 copies of width 35
2 copies of width 45
unused width=0


y(7)=1
1 copies of width 25
1 copies of width 35
2 copies of width 45
unused width=0


y(8)=1
6 copies of width 25
unused width=0


y(9)=1
6 copies of width 25
unused width=0
```

```
y(10)=1
1 copies of width 25
1 copies of width 35
2 copies of width 45
unused width=0


y(11)=1
6 copies of width 25
unused width=0


y(12)=1
6 copies of width 25
unused width=0


y(13)=1
5 copies of width 12
2 copies of width 45
unused width=0


y(14)=1
5 copies of width 12
2 copies of width 45
unused width=0


y(15)=1
1 copies of width 25
1 copies of width 35
2 copies of width 45
unused width=0


y(16)=1
1 copies of width 25
1 copies of width 35
```

```
2 copies of width 45
unused width=0


y(17)=1
3 copies of width 35
1 copies of width 45
unused width=0


y(18)=1
1 copies of width 12
3 copies of width 45
unused width=3


y(19)=1
6 copies of width 25
unused width=0


y(21)=1
1 copies of width 12
1 copies of width 35
2 copies of width 45
unused width=13


y(22)=1
6 copies of width 25
unused width=0


y(23)=1
3 copies of width 12
2 copies of width 45
unused width=24


y(24)=1
2 copies of width 25
```

```
2 copies of width 45
unused width=10


y(25)=1
5 copies of width 12
2 copies of width 45
unused width=0
```

2.1.3 3

2.1.3.1 a The following table (1) tabulates the termination results for the different data files for the Kantorovich formulation in $cs.Kantorovich.partial.mos$. The program had to be terminated only for $cs1.dat$ after 5 mins, the other data files completed successfully.

File	Completed	Comments
cs1.dat	No	Forcibly terminated the run after 300 secs.
cs2.dat	Yes	Completed in 2.6 secs.
kant1.dat	Yes	Completed in 0.1 secs.
kant2.dat	Yes	Completed in 1.0 secs.

Table 1: Kantorovich Solver: Termination results for different data files

2.1.3.2 b The following table (2) tabulates the branch-and-bound (BB) nodes searched by the BB algorithm when the solver in $cs.Kantorovich.partial.mos$ terminates for the different data files. For cs1.dat the program is had to be terminated after 5 mins, the other data files completed successfully.

2.1.3.3 c The following table (3) tabulates the objective value of the best lower bound Z_L and the objective value of the best integer solution Z_U when the solver in $cs.Kantorovich.partial.mos$ terminates for the different files. For cs1.dat the program is has to be terminated after 5 mins, the other data files completed successfully.

File	branch-and-bound (BB) nodes searched	Comments
cs1.dat	94503	Forcibly terminated the run after 300 secs.
cs2.dat	77	Completed in 2.6 secs
kant1.dat	1	Completed in 0.1 secs.
kant2.dat	1950	Completed in 1.0 secs.

Table 2: Kantorovich Solver: branch-and-bound (BB) nodes results searched for different data files

File	Z_L	Z_U	Comments
cs1.dat	542	544	Forcibly terminated the run after 300 secs.
cs2.dat	201	201	Completed in 2.6 secs
kant1.dat	22	22	Completed in 0.1 secs.
kant2.dat	30	30	Completed in 1.0 secs.

Table 3: Kantorovich Solver: Z_L and Z_U results for different data files

2.1.3.4 d The following table (4) tabulates the optimality gap, i.e. $\frac{(Z_U Z_L)}{Z_U}$ when the solver in $cs.Kantorovich.partial.mos$ terminates for the different files. For cs1.dat the program is has to be terminated after 5 mins, the other data files completed successfully.

File	Optimality Gap $\frac{(Z_U Z_L)}{Z_U}$	Comments
cs1.dat	0.3677%	Forcibly terminated the run after 300 secs.
cs2.dat	0%	Completed in 2.6 secs
kant1.dat	0%	Completed in 0.1 secs.
kant2.dat	0%	Completed in 1.0 secs.

Table 4: Kantorovich Solver: Optimality Gap $\frac{(Z_U Z_L)}{Z_U}$ for different data files

2.1.3.5 e The following table (5) tabulates how many integer solutions are found when the solver in $cs.Kantorovich.partial.mos$ terminates for the different files. For cs1.dat the program is has to be terminated after 5 mins, the other data files completed successfully.

File	Count of integer solutions $count(y_i * x_{ij})$ where $y_i * x_{ij} = ceil(y_i * x_{ij})$	Total Solutions $= count(y_i * x_{ij})$	Min. No of Rolls integer constraints on both y_i, x_{ij} relaxed	Min. No Rolls integer constraints on only x_{ij} relaxed
cs1.dat	79	545	542	542
cs2.dat	16	209	200.18	201
kant1.dat	8	23	21.667	22
kant2.dat	17	32	30	30

Table 5: Kantorovich Solver: How many integer solutions found for different data files

2.2 Gilmore Gomory Column Generation Formulation

2.2.1 1

The following is the implementation of the Gilmore-Gomory column generation formulation in file *cs.colgen.partial.mos*

```
model CuttingStockColGen ! Name the model

uses "mmxprs" ! include the Xpress solver package

! knapsack function solves the knapsack problem. You will need to write
! the content of the function in the later part of the code
! here is the initial declaration of the function
! forward function knapsack(y: array(range) of real, W: integer,
! w: array(range) of integer) : real

! cs1.dat - 8, cs2.dat-13, kant1.dat - 4 , kant2.dat - 5
m := 5                                               ! the number of different types o

declarations    ! declare sets, arrays, constraints, and variables
```

```
dRange = 1 .. m                                    ! the index range of demand
pRange : range                                     ! the index range of patterns
W : integer    ! width of the large roll
w : array(dRange) of integer                       ! widths of the small rolls
b : array(dRange) of integer                      ! demands of the small rolls

initialPatterns: array(dRange, pRange) of integer  ! set of initial patterns to start the colu
newPattern: array(dRange) of real                  ! the newly generated pattern by the knapsa
numPatterns: integer                               ! number of patterns in the restricted mast
Z : real                                 ! knapsack objective value
dualVar : array(dRange) of real                    ! dual variables for the demand constraints
DemandConstr: array(dRange) of linctr              ! Demand constraints
numRolls : linctr                                 ! objective value of cutting stock restricted
x : dynamic array(pRange) of mpvar                 ! decision variable: number of rolls cut in
numRollsBasis:basis                             ! save the basis for numRolls objective
totalWidth:real                                    ! count the total width of each pattern

end-declarations    ! end declarations section

! initialization from data file
! test on different data files cs1.dat, cs2.dat, cs3.dat, and report results for each instance
!!!!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!!!!!!!!!

initializations from 'kant2.dat'
W as 'W'
w as 'w'
b as 'b'
end-initializations

! Column generation needs to start from a subset of initial columns ir patterns
! As discussed in class, you can easily find a subset of m initial patterns which generates a
! Choose the subset of initial patterns and save them in the array initialPatterns
!!!!!!!!!!!!!!!!! Write your code here !!!!!!!!!!!!!!!!!!!!!!!!!
forall (j in dRange)
initialPatterns(j,j) := floor(W/w(j))

! set the number of initial patterns
! this number will need to be updated everytime a new pattern is generated
numPatterns := m
```

```
! create the corresponding x variables and set the nonnegativity constraints
!!!!!!!!!!!!!!!!! Write your code here !!!!!!!!!!!!!!!!!!!!!!!!!
forall (j in 1..numPatterns) do
create(x(j))
x(j) >= 0
! Since initial patterns are of only one type of cut, the number of rolls with those pat
! cannot be more than total demand/cuts per pattern rounded to the higher integer
x(j) <= integer(ceil(b(j)/initialPatterns(j,j)))
end-do

! Create demand constraints using the initialPatterns and the variables x created above.
! These constraints need to be named DemandConstr(i) because we'll need to get their sha
! and for us to be able to ask Xpress for shadow prices we need a way to refer to the co
!!!!!!!!!!!!!!!!!!! Write your code here !!!!!!!!!!!!!!!!!!!!!!!!!!!!
forall (j in dRange) do
DemandConstr(j) := sum(i in 1..numPatterns) x(i)*initialPatterns(j,i) >= b(j)
end-do

! Create objective function. The objective needs to be named numRolls
! Just like the demand constraints, we'll need to add variables to it as they get genera
!!!!!!!!!!!!!!!!!! Write your code here !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
numRolls := sum(i in 1..numPatterns) x(i)

!!!!!!!!!!!! Begin column generation !!!!!!!!!!!!!!!!!
defcut:=getparam("XPRS_CUTSTRATEGY") ! Save setting of CUTSTRATEGY
setparam("XPRS_CUTSTRATEGY", 0) ! Disable automatic cuts
setparam("XPRS_PRESOLVE", 0) ! Switch presolve off

repeat

! Solve the problem using the variables that have been generated so far
! Notice: we may need to use the XPRS_LIN parameter in the minimize command
! this is to make sure we do not take into accotn any of the global constraints
minimize(XPRS_LIN, numRolls)

! Print the objective value and the solution x
writeln("Number of Large Rolls = ", getobjval)
forall (j in 1..numPatterns)
writeln("x(",j,")=",getsol(x(j)))

! Get the dual varialbes (shadow prices)
```

```
!!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!!
forall (j in dRange) do
dualVar(j) := getdual(DemandConstr(j))
end-do

! Call the knapsack function, set Z to be the knapsack objective value
! for example: Z=knapsack(y,W,z), but you need to figure out what y is.
!!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!!

! hide the demands constraints before calling knapsack, so they are not used
! in knapsack optimization problem

(!forall (j in dRange) do
sethidden(DemandConstr(j),true)
end-do
!)

Z := knapsack(dualVar,W,w)

! unhide the constraints
(!forall (j in dRange) do
sethidden(DemandConstr(j),true)
end-do
!)


! If the reduced cost is less than zero
if 1 - Z <= -0.000001 then

! Print out the new pattern found by knapsack problem
!!!!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!!!!!
write("New Pattern :")
forall (j in dRange) do
write(newPattern(j)," ,")
end-do
writeln("")

! update the number of patterns generated
!  create the new variable
! update the constraints for each demand by adding the new pattern generated by knapsack
! update the objective function by adding the new pattern
```

```
!!!!!!!!!!!!!!! write your code here  !!!!!!!!!!!!!
numPatterns += 1

create(x(numPatterns))

numRolls += x(numPatterns)

xMax := 0
forall (j in dRange) do
if (newPattern(j) > 0) then
DemandConstr(j) +=  x(numPatterns)*newPattern(j)
xMax := maxlist(xMax,ceil(b(j)/newPattern(j)))
end-if
end-do

x(numPatterns) <= xMax

end-if

until 1 - Z >= -0.000001 ! continue until no variable has a good reduced cost
setparam("XPRS_CUTSTRATEGY", defcut) ! Enable automatic cuts
setparam("XPRS_PRESOLVE", 1) ! Switch presolve on

!!!!!!!!!!!! End column generation !!!!!!!!!!!!!!!!!

! optimize again with optimum patterns to get the context to the numRolls objectce
minimize(numRolls)

writeln("\n===============\n")
! Print a summary of the final LP solution: selected patterns, number of rolls cut using
!!!!!!!!!!! wrtie your code here !!!!!!!!!!!!!!!!!

! print the coefficients of constraints, showing the patterns picked
writeln("LP Solution")
writeln("=======================")
writeln("Number of rolls to be used is = ",getobjval)
writeln("")
writeln("The number and types of Patterns to be cut");
writeln("")
forall (i in 1..numPatterns) do
totalWidth := 0
```

```
if ( getsol(x(i)) > 0 ) then
write("x(",i,")=",getsol(x(i)))
write(" of : [")
forall(j in 1..m) do
r := getcoeff(DemandConstr(j) ,x(i) )
write(w(j),":",r,",")
totalWidth+= w(j)*r
end-do
write("]")
writeln(", Total Width = ",totalWidth)
end-if

end-do





! We also want to get integer solutions. There are at least two approaches to do this.
! The first approach is to round up the LP solution.
! The second approach is to re-solve the master problem as an integer program using all the ge


!!!!!!!!!!!!!!!!!!! ROUNDING LP SOLUTION !!!!!!!!!!!!!!!!!!!!!!!!!
! Implement the first approach: rounding the LP solution and print out a summary ofthe solutio
! number of rolls cut using each selected pattern, total number of rolls
!!!!!!!!!!!!!!!!!!! Write your code here !!!!!!!!!!!!!!!!!!!!!!!!!
! print the coefficients of constraints, showing the patterns picked
writeln("\n===============\n")
writeln("Rounded LP Solution")
writeln("======================")
writeln("Number of rolls to be used is = ",ceil(getobjval))
writeln("")
writeln("The number and types of Patterns to be cut");
writeln("")
forall (i in 1..numPatterns) do
totalWidth := 0
if ( getsol(x(i)) > 0 ) then
write("x(",i,")=",ceil(getsol(x(i))))
write(" of : [")
forall(j in 1..m) do
```

```
r := getcoeff(DemandConstr(j) ,x(i) )
write(w(j),":",r,",")
totalWidth+= w(j)*r
end-do
write("]")
writeln(", Total Width = ",totalWidth)
end-if

end-do




!!!!!!!!!!!!!!!!!!! INTEGER SOLUTION !!!!!!!!!!!!!!!!!!!!!!!!!
! Implement the second approach: first, constrain all variables to be integer (example:
! then solve the master problem using all generated patterns
! You also need to print out a summary of the solution: number of rolls cut using each s
! total number of rolls
!!!!!!!!!!!!!!!!!!! Write your code here !!!!!!!!!!!!!!!!!!!!!!!!!
!delcell(x)

! set the constraints as linear
forall (j in 1..numPatterns) do
x(j) is_integer
end-do

! optimize again with optimum patterns to get the context to the numRolls objectce
minimize(numRolls)

writeln("\n===============\n")
! Print a summary of the final LP solution: selected patterns, number of rolls cut using
!!!!!!!!!!! wrtie your code here !!!!!!!!!!!!!!!!!!

! print the coefficients of constraints, showing the patterns picked
writeln("Integer Solution")
writeln("=======================")
writeln("Number of rolls to be used is = ",getobjval)
writeln("")
writeln("The number and types of Patterns to be cut");
writeln("")
forall (i in 1..numPatterns) do
```

```
totalWidth := 0
if ( getsol(x(i)) > 0 ) then
write("x(",i,")=",getsol(x(i)))
write(" of : [")
forall(j in 1..m) do
r := getcoeff(DemandConstr(j) ,x(i) )
write(w(j),":",r,",")
totalWidth+= w(j)*r
end-do
write("]")
writeln(", Total Width = ",totalWidth)
end-if

end-do





!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!! Knapsack Problem !!!!!!!!!!!!!!!!!!!!!!
!! implement a function that solves the knapsack problem   !!
!! Inputs: objective coefficient array y                   !!
!!         width of the large roll W                       !!
!!         widths of the small rolls array w               !!
!! Return: optimal objective function value                !!
!!         optimal solution (i.e. the new pattern)         !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function knapsack(y: array(range) of real, W: integer, w: array(range) of integer) : real

! declare decision variable
! form constraints and objective
! do not forget the integrality constraints on the variables
! solve the problem
!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!
declarations

k:array(1..m) of mpvar   ! the optimum solution for the new pattern
knapcnstr:linctr  ! knapsack constraint that the total width of cuts < width of troll
knapobj:linctr    ! knapsack objective of maximizing c_B B^-1 A_j to get new pattern
```

```
end-declarations

! select pattern that maximizes  c_B B^-1 A_j to get new pattern
knapobj := sum(j in 1..m) y(j)*k(j)

! knapsack constraint that the total width of cuts < width of troll
knapcnstr := sum(j in 1..m) w(j)*k(j) <= W

! since we define number of cuts of each width in the pattern, it needs to be a integer
forall (j in 1..m)
k(j) is_integer

! perform the optimization
maximize(knapobj)

! return the optimal objective value
! the keyword returned is reserved for this purpose by Xpress
returned := getobjval


! return the optimal solution, save it in the array newPattern defined in the beginning
!!!!!!!!!!!! write your code here !!!!!!!!!!!!!!
forall ( j in 1..m) do
newPattern(j) := getsol(k(j))
!newPattern(j) := round(getsol(k(j)))
end-do

! erase the knapsack constraints before returning, so they are not used
! in demand optimization
knapobj := 0
knapcnstr := 0

end-function

end-model
```

2.2.2 2

2.2.2.1 cs1.dat The following is the summary output from the column generation formulation in *cs.colgen.partial.mos* for data file *cs1.dat*. The summary gives the optimal LP, the rounded LP and the Integer solution results for the number of rolls to be cut, the types of patterns to be cut and the number of each of those patterns to be cut. The debug output during each iteration of column generation is not shown here for brevity.

```
===============

LP Solution
========================
Number of rolls to be used is = 542.5

The number and types of Patterns to be cut

x(4)=50 of : [45:0,35:0,30:0,25:4,20:0,12:0,7:0,6:0,], Total Width = 100
x(6)=10 of : [45:0,35:0,30:0,25:0,20:0,12:8,7:0,6:0,], Total Width = 96
x(9)=305 of : [45:0,35:2,30:1,25:0,20:0,12:0,7:0,6:0,], Total Width = 100
x(10)=90 of : [45:1,35:0,30:1,25:0,20:0,12:0,7:1,6:3,], Total Width = 100
x(13)=10 of : [45:1,35:0,30:0,25:0,20:0,12:1,7:0,6:7,], Total Width = 99
x(14)=2.5 of : [45:0,35:0,30:0,25:0,20:0,12:0,7:4,6:12,], Total Width = 100
x(15)=33.75 of : [45:0,35:0,30:0,25:0,20:2,12:4,7:0,6:2,], Total Width = 100
x(16)=41.25 of : [45:0,35:0,30:0,25:0,20:2,12:0,7:0,6:10,], Total Width = 100

===============

Rounded LP Solution
========================
The number and types of Patterns to be cut

x(4)=50 of : [45:0,35:0,30:0,25:4,20:0,12:0,7:0,6:0,], Total Width = 100
x(6)=10 of : [45:0,35:0,30:0,25:0,20:0,12:8,7:0,6:0,], Total Width = 96
x(9)=305 of : [45:0,35:2,30:1,25:0,20:0,12:0,7:0,6:0,], Total Width = 100
x(10)=90 of : [45:1,35:0,30:1,25:0,20:0,12:0,7:1,6:3,], Total Width = 100
x(13)=10 of : [45:1,35:0,30:0,25:0,20:0,12:1,7:0,6:7,], Total Width = 99
x(14)=3 of : [45:0,35:0,30:0,25:0,20:0,12:0,7:4,6:12,], Total Width = 100
x(15)=34 of : [45:0,35:0,30:0,25:0,20:2,12:4,7:0,6:2,], Total Width = 100
x(16)=42 of : [45:0,35:0,30:0,25:0,20:2,12:0,7:0,6:10,], Total Width = 100

Number of rolls to be used is = 544
```

```
===============

Integer Solution
========================
Number of rolls to be used is = 543

The number and types of Patterns to be cut

x(4)=50 of : [45:0,35:0,30:0,25:4,20:0,12:0,7:0,6:0,], Total Width = 100
x(6)=10 of : [45:0,35:0,30:0,25:0,20:0,12:8,7:0,6:0,], Total Width = 96
x(9)=305 of : [45:0,35:2,30:1,25:0,20:0,12:0,7:0,6:0,], Total Width = 100
x(10)=90 of : [45:1,35:0,30:1,25:0,20:0,12:0,7:1,6:3,], Total Width = 100
x(13)=10 of : [45:1,35:0,30:0,25:0,20:0,12:1,7:0,6:7,], Total Width = 99
x(14)=3 of : [45:0,35:0,30:0,25:0,20:0,12:0,7:4,6:12,], Total Width = 100
x(15)=34 of : [45:0,35:0,30:0,25:0,20:2,12:4,7:0,6:2,], Total Width = 100
x(16)=41 of : [45:0,35:0,30:0,25:0,20:2,12:0,7:0,6:10,], Total Width = 100
```

2.2.2.2 cs2.dat The following is the summary output from the column generation formulation in *cs.colgen.partial.mos* for data file *cs2.dat*. The summary gives the optimal LP, the rounded LP and the Integer solution results for the number of rolls to be cut, the types of patterns to be cut and the number of each of those patterns to be cut. The debug output during each iteration of column generation is not shown here for brevity.

```
===============

LP Solution
========================
Number of rolls to be used is = 200.18

The number and types of Patterns to be cut

x(3)=2.11432 of : [3:0,4:0,5:10,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total W
x(8)=8.6 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:5,11:0,12:0,13:0,14:0,15:0,], Total Width
x(14)=15 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:1,12:0,13:3,14:0,15:0,], Total Width
x(15)=50 of : [3:0,4:0,5:0,6:0,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:3,15:0,], Total Width
x(16)=12.5 of : [3:0,4:0,5:0,6:1,7:0,8:0,9:0,10:0,11:4,12:0,13:0,14:0,15:0,], Total Widt
x(17)=11 of : [3:0,4:0,5:1,6:0,7:0,8:0,9:5,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width
x(18)=41.6667 of : [3:0,4:0,5:1,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:3,], Total W
```

```
x(19)=3.28219 of : [3:14,4:0,5:0,6:0,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width
x(20)=6.54545 of : [3:0,4:11,5:0,6:1,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width
x(21)=10.4221 of : [3:0,4:0,5:0,6:7,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =
x(22)=25 of : [3:0,4:0,5:0,6:0,7:2,8:0,9:0,10:0,11:0,12:3,13:0,14:0,15:0,], Total Width = 50
x(23)=6.85915 of : [3:1,4:0,5:0,6:0,7:1,8:5,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =
x(24)=7.19014 of : [3:1,4:0,5:1,6:0,7:6,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =

===============

Rounded LP Solution
========================
The number and types of Patterns to be cut

x(3)=3 of : [3:0,4:0,5:10,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(8)=9 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:5,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(14)=15 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:1,12:0,13:3,14:0,15:0,], Total Width = 50
x(15)=50 of : [3:0,4:0,5:0,6:0,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:3,15:0,], Total Width = 50
x(16)=13 of : [3:0,4:0,5:0,6:1,7:0,8:0,9:0,10:0,11:4,12:0,13:0,14:0,15:0,], Total Width = 50
x(17)=11 of : [3:0,4:0,5:1,6:0,7:0,8:0,9:5,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(18)=42 of : [3:0,4:0,5:1,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:3,], Total Width = 50
x(19)=4 of : [3:14,4:0,5:0,6:0,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(20)=7 of : [3:0,4:11,5:0,6:1,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(21)=11 of : [3:0,4:0,5:0,6:7,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(22)=25 of : [3:0,4:0,5:0,6:0,7:2,8:0,9:0,10:0,11:0,12:3,13:0,14:0,15:0,], Total Width = 50
x(23)=7 of : [3:1,4:0,5:0,6:0,7:1,8:5,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(24)=8 of : [3:1,4:0,5:1,6:0,7:6,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50

Number of rolls to be used is = 205


===============

Integer Solution
========================
Number of rolls to be used is = 203

The number and types of Patterns to be cut

x(1)=3 of : [3:16,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 48
x(3)=2 of : [3:0,4:0,5:10,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 50
x(4)=2 of : [3:0,4:0,5:0,6:8,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width = 48
```

```
x(5)=1 of : [3:0,4:0,5:0,6:0,7:7,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =
x(8)=9 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:5,11:0,12:0,13:0,14:0,15:0,], Total Width =
x(10)=5 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:4,13:0,14:0,15:0,], Total Width =
x(14)=15 of : [3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:1,12:0,13:3,14:0,15:0,], Total Width
x(15)=50 of : [3:0,4:0,5:0,6:0,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:3,15:0,], Total Width
x(16)=13 of : [3:0,4:0,5:0,6:1,7:0,8:0,9:0,10:0,11:4,12:0,13:0,14:0,15:0,], Total Width
x(17)=11 of : [3:0,4:0,5:1,6:0,7:0,8:0,9:5,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width
x(18)=42 of : [3:0,4:0,5:1,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:3,], Total Width
x(20)=7 of : [3:0,4:11,5:0,6:1,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width
x(21)=8 of : [3:0,4:0,5:0,6:7,7:0,8:1,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =
x(22)=19 of : [3:0,4:0,5:0,6:0,7:2,8:0,9:0,10:0,11:0,12:3,13:0,14:0,15:0,], Total Width
x(23)=8 of : [3:1,4:0,5:0,6:0,7:1,8:5,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =
x(24)=8 of : [3:1,4:0,5:1,6:0,7:6,8:0,9:0,10:0,11:0,12:0,13:0,14:0,15:0,], Total Width =
```

2.2.2.3 kant1.dat

The following is the summary output from the column generation formulation in *cs.colgen.partial.mos* for data file *kant1.dat*. The summary gives the optimal LP, the rounded LP and the Integer solution results for the number of rolls to be cut, the types of patterns to be cut and the number of each of those patterns to be cut. The debug output during each iteration of column generation is not shown here for brevity.

```
===============

LP Solution
========================
Number of rolls to be used is = 21.6667

The number and types of Patterns to be cut

x(2)=6.66667 of : [12:0,25:6,35:0,45:0,], Total Width = 150
x(7)=5 of : [12:5,25:0,35:0,45:2,], Total Width = 150
x(8)=10 of : [12:0,25:1,35:1,45:2,], Total Width = 150

===============

Rounded LP Solution
========================
The number and types of Patterns to be cut
```

```
x(2)=7 of : [12:0,25:6,35:0,45:0,], Total Width = 150
x(7)=5 of : [12:5,25:0,35:0,45:2,], Total Width = 150
x(8)=10 of : [12:0,25:1,35:1,45:2,], Total Width = 150

Number of rolls to be used is = 22


===============

Integer Solution
========================
Number of rolls to be used is = 22

The number and types of Patterns to be cut

x(2)=7 of : [12:0,25:6,35:0,45:0,], Total Width = 150
x(7)=5 of : [12:5,25:0,35:0,45:2,], Total Width = 150
x(8)=10 of : [12:0,25:1,35:1,45:2,], Total Width = 150
```

2.2.2.4 kant2.dat The following is the summary output from the column generation formulation in *cs.colgen.partial.mos* for data file *kant2.dat*. The summary gives the optimal LP, the rounded LP and the Integer solution results for the number of rolls to be cut, the types of patterns to be cut and the number of each of those patterns to be cut. The debug output during each iteration of column generation is not shown here for brevity.

```
===============

LP Solution
========================
Number of rolls to be used is = 30

The number and types of Patterns to be cut

x(2)=6.66667 of : [12:0,25:6,35:0,45:0,50:0,], Total Width = 150
x(5)=8.33333 of : [12:0,25:0,35:0,45:0,50:3,], Total Width = 150
x(8)=5 of : [12:5,25:0,35:0,45:2,50:0,], Total Width = 150
x(9)=10 of : [12:0,25:1,35:1,45:2,50:0,], Total Width = 150
```

```
===============

Rounded LP Solution
========================
The number and types of Patterns to be cut

x(2)=7 of : [12:0,25:6,35:0,45:0,50:0,], Total Width = 150
x(5)=9 of : [12:0,25:0,35:0,45:0,50:3,], Total Width = 150
x(8)=5 of : [12:5,25:0,35:0,45:2,50:0,], Total Width = 150
x(9)=10 of : [12:0,25:1,35:1,45:2,50:0,], Total Width = 150

Number of rolls to be used is = 31


===============

Integer Solution
========================
Number of rolls to be used is = 31

The number and types of Patterns to be cut

x(2)=7 of : [12:0,25:6,35:0,45:0,50:0,], Total Width = 150
x(5)=9 of : [12:0,25:0,35:0,45:0,50:3,], Total Width = 150
x(8)=5 of : [12:5,25:0,35:0,45:2,50:0,], Total Width = 150
```

2.2.3 3

Table (6) shows the optimum number of rolls to be cut for each file for each of the three models of LP, LP Rounded and Integer. The results are close. As the number of widths to be cut is higher the Integer model appears to give a better solution than the rounded LP model. The regular LP model while providing a good lower bound to the solutions of LP Rounded and Integer, is not something that can be implemented. This is because the regular LP model is not guaranteed to give the number of cuts for all patterns as whole numbers. This cannot work in practice as the number of cuts has to be a whole number. But rounding the cuts of each individual pattern in the regular LP model to the next higher integer adding them to get the total number of rolls gives us a practical solution which can be implemented. This is the LP Rounded solution. But this will not give us a lower count than minimum number of rolls given by a Integer constraint solution. Because, if the regular LP where to give us a better solution where the individual counts are all integers, then in that case it will be the the Integer solution.

Model	cs1.dat	cs2.dat	kant1.dat	kant2.dat
LP	542.5	200.18	21.667	30
LP Rounded	544	205	22	31
Integer	543	203	22	31

Table 6: Compare No. Rolls to be cut - LP, LP Rounded and Integer approaches

2.2.4 4

Table (7) compares the best Integer Solution from the Column Generation approach to the solution from the Kantorovich Formulation for each of the data files. The Kantorovich formulation gives slightly a better solution for the data files $cs2.dat$ and $kant2.dat$. The Column Generation method gives a marginally better solution for the $cs1.dat$ file, as for this file the Kantorovich Solution was forced to terminate after 300 secs before it could complete. For the data files $kant2.dat$ the results of both the formulations match.

	cs1.dat	cs2.dat	kant1.dat	kant2.dat
Kantorovich Formulation	544(5 mins)	201	22	30
Column Generation	543	203	22	31

Table 7: Best Solution from Column Generation Vs Kantorovich Formulation

3 Dantzig Wolfe Algorithm

3.1 1

The following is the implementation of the Dantzig-Wolfe algorithm in file $DW.partial.mos$ with data from question 2a.

```
model ModelName
uses "mmxprs"; !gain access to the Xpress-Optimizer solver
uses "mmsystem"

! prob1 & prob 2
n := 5  ! number of variables
m := 1  ! number of coupling constraints


forward function pricing(c: array(range) of real, D : array(range, range) of real, y : a

declarations
varRange = 1 .. n ! index of x variables
constrRange = 1 .. m ! index of constraints on x
constrRangeplus1 = 1 .. m+1 ! including the convexity constraint
c : array(varRange) of real ! cost of x
D : array(constrRange, varRange) of real ! D
b0 : array(constrRange) of real ! Dx = b0
    generatedExtrPt : dynamic array(varRange, range) of real  ! generated extreme points
    newExtrPt : array(varRange) of real ! newly generated extreme point in the current i
    dualVar_y : array(constrRange) of real ! dual variables for the coupling constraints
    dualVar_r : real ! dual variable for the convexity constraint
    numExtrPt : integer ! counting the number of generated extreme points
    Z : real ! optimal objective value of the pricing problem
    currentlambda : dynamic array(range) of real ! holding the optimal solution of the c
    lambda : dynamic array(range) of mpvar ! lambda variable

    couplingConstr: array(constrRange) of linctr   ! m coupling constraints
    convexityConstr: linctr                        ! convexity constraint

end-declarations

! prob 1 & prob 2
c :: [-4, -1, -6, -3, -5]
D :: [3, 2, 4, 3, 5]
b0 :: [25]


!prob1
generatedExtrPt(1,1) := 1
generatedExtrPt(2,1) := 2
generatedExtrPt(3,1) := 1
generatedExtrPt(4,1) := 1
generatedExtrPt(5,1) := 1
generatedExtrPt(1,2) := 2
generatedExtrPt(2,2) := 2
generatedExtrPt(3,2) := 2
generatedExtrPt(4,2) := 1
generatedExtrPt(5,2) := 1


!prob2
(!
generatedExtrPt(1,1) := 1
generatedExtrPt(2,1) := 1
generatedExtrPt(3,1) := 1
generatedExtrPt(4,1) := 1
generatedExtrPt(5,1) := 1
generatedExtrPt(1,2) := 3
generatedExtrPt(2,2) := 3
generatedExtrPt(3,2) := 3
generatedExtrPt(4,2) := 3
generatedExtrPt(5,2) := 3
!)
numExtrPt := m+1

! lambda are created for the generated points
forall ( i in 1..numExtrPt) do
create(lambda(i))
lambda(i)>=0
end-do

!!!! Write the objective function of the restricted master problem
totalCost := sum(i in 1..numExtrPt, j in 1..n) lambda(i)*c(j)*generatedExtrPt(j,i)

!!!! Write coupling constraint
forall (i in constrRange) do
couplingConstr(i) :=  sum ( l in 1..numExtrPt, j in 1..n ) lambda(l)*D(i,j)*generatedExtrPt(j,
end-do

!!!! Write the convexity constraint
convexityConstr :=  sum ( i in 1..numExtrPt) lambda(i) = 1
```

```
repeat
!!!! solve the restricted master problem using the variables that have been generated so
minimize(XPRS_LIN, totalCost)
writeln("")
    writeln("0 if min is solved:", getsysstat)
    writeln("current obj of RMP = ", getobjval)

    !!!! get current lambda solution
    forall (i in 1..numExtrPt) do
     currentlambda(i) := getsol(lambda(i))
     writeln("lambda(",i,")=", currentlambda(i))
    end-do

    !!!! get the dual solution !!!!
    forall (i in constrRange) do
     dualVar_y(i) := getdual(couplingConstr(i))
     writeln("dual var of coupling constraint(", i, ")=", dualVar_y(i))
    end-do
    dualVar_r := getdual(convexityConstr)
    writeln("dual var of convexity constraint=", dualVar_r)

    !!!! solve the pricing problem !!!!
    Z := pricing(c, D, dualVar_y)
    writeln("min reduced cost=", Z)

    !!!! If min reduced cost is less than -0.000001, then found a new extreme point !!!!
    !!!! You need to figure out how to compute the min reduced cost !!!!
    if (   ( Z - dualVar_r)    < -0.000001) then
     writeln("found a new extreme point")

     !!!! write you own code here to add the new column to the lambda problem !!!!
     numExtrPt += 1
     forall (i in 1..n) do
     create(generatedExtrPt(i,numExtrPt))
     generatedExtrPt(i,numExtrPt) := newExtrPt(i)
     end-do

      ! lambda are created for the generated point
     create(lambda(numExtrPt))
     lambda(numExtrPt) >= 0
     !!!! update the objective function of the restricted master problem
totalCost += sum(j in 1..n) lambda(numExtrPt)*c(j)*generatedExtrPt(j,numExtrPt)

!!!! update coupling constraint
forall (i in constrRange) do
couplingConstr(i) +=  sum (j in 1..n ) lambda(numExtrPt)*D(i,j)*generatedExtrPt(j,numExtrPt)
end-do

!!!! update the convexity constraint
convexityConstr +=  lambda(numExtrPt)

    end-if

    !!!! repeat until the min reduced cost is great than -0.000001 !!!!
until (    ( Z - dualVar_r)    > -0.000001)

minimize(totalCost)

writeln("\n====================================")
writeln("\nOptimal solution:")


writeln("Optimum obj of MP = ", getobjval)

!!!! print out the optimal lambda solution !!!!
!!!! notice that in the above code, after solving the RMP, you solve the pricing problem, ther
!!!! the optimal lambda solution from the RMP is lost. That's why we save the optimal lambda i
!!!! vector currentlambda right after solving the RMP. Use it here
    ! Write your code here !

! lambda are created for the generated points
writeln("\nlambda:")
forall ( i in 1..numExtrPt) do
writeln("lambda(",i,") = ",currentlambda(i))
end-do


!!!! print out the optimal x solution !!!!
! The generated X
writeln("\n\nGenerated Extreme points x:")
```

```
forall ( i in 1..n) do
if ( i = 1 ) then
forall( j in 1..numExtrPt)
write("x(",j,")   ")
writeln("")
end-if
forall( j in 1..numExtrPt) do
write(generatedExtrPt(i,j),"      ")
end-do
writeln("")
end-do
writeln("")

!!!! print out the optimal x solution !!!!
! write your code here !
writeln("\nOptimal x:")
forall ( i in 1..n) do
x := sum( j in 1..numExtrPt) currentlambda(j)*generatedExtrPt(i,j)
writeln("x(",i,") = ",x)
end-do




    !!!! Pricing function !!!!
    function pricing(c: array(range) of real, D : array(range,range) of real, y : array(

!!!! write your pricing function code here!!!!
!!!! notice that you need to save the optimal solution of the
!!!! pricing problem as a new extreme point
!!!! in the provided vector newExtrPt
! Write your code here !
declarations
yD:dynamic array(range) of real ! array to hold y*D
x:array(1..n) of mpvar    ! the x array
minObj: linctr            ! the objective constraints
end-declarations

! Y^t D
forall ( i in 1..getsize(c)) do
create(yD(i))
```

```
yD(i) := sum(j in 1..getsize(y)) y(j)*D(j,i)
end-do

! ( c- y^T D) x
minObj := sum( i in 1..getsize(c)) (c(i) - yD(i))*x(i)

! polyhedron constraints
! and general constraints
forall ( i in 1..n) do
x(i) >= 1
x(i) <= 2
end-do

! minimize the objective
minimize(minObj)

! get Z
returned := getobjval

! new extreme point
forall( i in 1..n )
     newExtrPt(i) := getsol(x(i))

    end-function



end-model
```

3.2 2

3.2.0.1 a The following is the output of the Dantzig-Wolfe algorithm in file *DW.partial.mos* with data and initial extreme points from question (2a). The output shows the optimal solution of the (RMP) and the associated dual solution and the the minimum reduced cost at each intermediate iteration of the algorithm during the process of generating a new extreme point for the RMP.

The final optimal objective solution, along with the final optimal solution for λ and the final computed optimal x, computed from the final λ's and the generated extreme points are listed at the end of the output.

```
0 if min is solved:0
current obj of RMP = -28.5714
lambda(1)=0.142857
lambda(2)=0.857143
dual var of coupling constraint(1)=-1.42857
dual var of convexity constraint=7.14286
min reduced cost=5
found a new extreme point

0 if min is solved:0
current obj of RMP = -29
lambda(1)=0
lambda(2)=0.8
lambda(3)=0.2
dual var of coupling constraint(1)=-1
dual var of convexity constraint=-4
min reduced cost=-5
found a new extreme point

0 if min is solved:0
current obj of RMP = -29.5
lambda(1)=0
lambda(2)=0.5
lambda(3)=0
lambda(4)=0.5
dual var of coupling constraint(1)=-0.5
dual var of convexity constraint=-17
min reduced cost=-21
found a new extreme point

0 if min is solved:0
current obj of RMP = -30
lambda(1)=0
lambda(2)=0
lambda(3)=0
lambda(4)=0.875
lambda(5)=0.125
dual var of coupling constraint(1)=-1
dual var of convexity constraint=-5
min reduced cost=-5

===================================

Optimal solution:
Optimum obj of MP = -30

lambda:
lambda(1) = 0
lambda(2) = 0
lambda(3) = 0
lambda(4) = 0.875
lambda(5) = 0.125


Generated Extreme points x:
x(1)   x(2)   x(3)   x(4)   x(5)
1      2      1      2      2
2      2      1      1      1
1      2      2      2      2
1      1      1      1      2
1      1      1      1      2


Optimal x:
x(1) = 2
x(2) = 1
x(3) = 2
x(4) = 1.125
x(5) = 1.125
```

3.2.0.2 b The following is the output of the Dantzig-Wolfe algorithm in file *DW.partial.mos* with data and initial extreme points from question (2b). The output shows the optimal solution of the (RMP) and the associated dual solution and the

the minimum reduced cost at each intermediate iteration of the algorithm during the process of generating a new extreme point for the RMP.

The final optimal objective solution, along with the final optimal solution for λ and the final computed optimal x, computed from the final λ's and the generated extreme points are listed at the end of the output.

```
0 if min is solved:0
current obj of RMP = -27.9412
lambda(1)=0.764706
lambda(2)=0.235294
dual var of coupling constraint(1)=-1.11765
dual var of convexity constraint=0
min reduced cost=-4.35294
found a new extreme point

0 if min is solved:0
current obj of RMP = -30.4286
lambda(1)=0.428571
lambda(2)=0
lambda(3)=0.571429
dual var of coupling constraint(1)=-1.42857
dual var of convexity constraint=5.28571
min reduced cost=4.71429
found a new extreme point

0 if min is solved:0
current obj of RMP = -31
lambda(1)=0
lambda(2)=0
lambda(3)=0
lambda(4)=1
dual var of coupling constraint(1)=-1.33333
dual var of convexity constraint=2.33333
min reduced cost=2.33333

===================================

Optimal solution:
Optimum obj of MP = -31
```

```
lambda:
lambda(1) = 0
lambda(2) = 0
lambda(3) = 0
lambda(4) = 1


Generated Extreme points x:
x(1)   x(2)   x(3)   x(4)
1      3      3      1
1      3      1      1
1      3      3      3
1      3      1      1
1      3      1      1


Optimal x:
x(1) = 1
x(2) = 1
x(3) = 3
x(4) = 1
x(5) = 1
```

4 References

[1] Bertsimas Dimitris, Tsitsiklis N. John, *Introduction to Linear Optimization*, Athena Scientific Edition 6, 1997.