LINEAR PROGRAMMING

Homework 9

Fall 2014 Csci 628 Megan Rose Bryant

1. *Consider a cutting-stock problem with raw width 100 inch and order summary calling for*

   - 600 finals of width 52 inches
   - 600 finals of width 29 inches
   - 600 finals of width 27 inches
   - 1200 finals of width 21 inches

   (a) *List all patterns that are maximal, i.e., the part of the raw that is discarded is too small to cut any final.*

Pattern #	1	2	3	4	5	6	7	8	9	10	11	12
a_{1j}	1	1	0	0	0	0	0	0	0	0	0	0
a_{2j}	1	0	0	3	2	2	1	1	0	0	0	0
a_{3j}	0	1	0	0	1	0	2	0	3	2	1	0
a_{4j}	0	1	2	0	0	2	0	3	0	2	3	4
Total	81	100	94	87	85	100	83	92	81	96	90	84

   (b) *Show that the optimal solution uses 900 raws.* We see from the AMPL output below that the optimal solution uses 900 raws and 2 cutting patterns. The AMPL files are available for review in **Appendix A**.
   AMPL Output:

```
ampl: reset; include cut.run;
Gurobi 5.6.3: optimal solution; objective 0.5833333333
Gurobi 5.6.3: optimal solution; objective 950
Gurobi 5.6.3: optimal solution; objective 0.1666666667
1 simplex iterations
Gurobi 5.6.3: optimal solution; objective 950
1 simplex iterations
Gurobi 5.6.3: optimal solution; objective 0.1666666667
1 simplex iterations
Gurobi 5.6.3: optimal solution; objective 900
1 simplex iterations
Gurobi 5.6.3: optimal solution; objective 0
```

```
nbr [*,*] (tr)
:   1   2   3   4    :=
1   1   0   0   0
2   0   3   0   0
3   0   0   3   0
4   0   0   0   4
5   1   0   1   1
6   0   0   2   2
7   0   2   0   2
;

Cut [*] :=
1    0
2    0
3    0
4    0
5  600
6    0
7  300
;
```

(c) *First-fit-decreasing means that we always cut the largest final for which we still have orders left, and which fits into the remaining material of the raw. Show that the solution found by first-fit-decreasing uses 1100 raws.*

This methodology would lead us to satisfy the largest length finals first. Therefore, we would begin by making cuts according to pattern 1 600 times. This would result in 600 finals of length 52 and 600 finals of length 29 thereby satisfing the two largest lengths. The next largest length final is length 27. In order to satisfy demand for this final, we must make $\frac{600}{3} = 200$ cuts following pattern 9. Finally, we must satisfy the demand for the remaining finals of length 21. These are best satisfied using pattern 12. Therefore, we will make $1200/4 = 300$ cuts following pattern 12. This results in 1200 cuts of length 21.

Therefore, we see that using the first-fit-decreasing method we require $600 + 200 + 300 = 1100$ total cuts. This is a large increase over the optimal number of cuts, which was found to be 900 in part (b).

2. *Suppose the raw width is 181 inch, and the order summary is calling for*

- 90 finals of width $21\frac{5}{8}$ inches
- 51 finals of width $20\frac{1}{2}$ inches
- 45 finals of width 20 inches
- 11 finals of width $17\frac{1}{4}$ inches

(a) *Find an initial set of patterns (as we did in class when we started the revised simplex method), where the i-th pattern cuts only finals of type i.*

Here we have an initial set of patterns (starting basis) where the ith pattern cuts only final type i.

$$B = \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

(b) *(Optional: you do not need to hand this in) Execute one full iteration of the revised simplex method, starting with the columns you generated in (a).*

(c) *Create a data file for this instance, and use the AMPL files provided on blackboard to solve the LP relaxation. Report the optimal value of the LP relaxation, and report the optimal solution (i.e., the patterns used, and the (fractional) number of raws cut according to each pattern.)*

The optimal objective value is 23 raws cut according to the following patterns:

- 10×**Pattern 1:** 8 cuts of final 1.
- 6.0×**Pattern 2:** 7 cuts of final 2, 1 cut of final 3, and 1 cut of final 4.
- 4.5×**Pattern 3:** 2 cuts of final 2, 7 cuts of final 3.
- 2.5×**Pattern 4:** 4 cuts of final 1, 3 cuts of final 3, and 2 cuts of final 4. The AMPL output as well as the data, model, and run files used to obtain the above solution is available in **Appendix B**.

(d) *Use part (c) to give the optimal (integer) solution to this cutting stock problem. Explain how you know that this is indeed the optimal solution.*

In order to derive the optimal integer solution to this cutting stock problem we must consider first the implications of the fractional solution. Currently, we have an optimal solution of $x^{*T} = \begin{bmatrix} 10 & 0 & 0 & 0 & 6 & 0 & 0 & 4.5 & 2.5 \end{bmatrix}$.

In order to achieve an integer solution while ensuring that we satisfy production, we must round up our non-integer solution. This will give us the following integer solution: $x^{*T} = \begin{bmatrix} 10 & 0 & 0 & 0 & 6 & 0 & 0 & 5 & 3 \end{bmatrix}$.

This integer solution will result in a surplus of the finals used in the effected patterns (patterns 8 and 9) and an increase in the total number of raws needed from 23 to 24.

We know that this is the optimal integer solution since the number of raws needed for the integer solution is the next integer (24) after the number of raws needed for the non-integer solution. Therefore, there are no integer solutions that can decrease the number of raws needed while meeting production demand and we know that this integer solution is an optimal integer solution.

(e) *Suppose now that the machine that cuts the raws has only eight knives, and, consequently, a cutting pattern specified by $a_1, \ldots, a_m$ is admissible, only if $\sum_{i=1}^{m} a_i$ does not exceed eight. Explain how does this changes your answer to part (a). You do not have to redo part (b), but explain how Step 2 (finding k such that $c_k > 0$) should be changed to make sure we generate a column (pattern) that does not cut more than eight finals.*

This would have a significant impact on our starting basis found in part (a). This is because patterns 3 and 4 of that basis have more than 8 cuts, which is prohibited with

the new, 8-knife machine. This new restriction would therefore invalidate the given starting basis and exclude those two patterns from the feasible region.

"Step 2" is where we choose an entering index $k \in \mathbb{N}$ such that $c_k > 0$. Given that we found $\mathbf{y}$ in step 1 such that $A_B^T \mathbf{y} = c_b$, we must select k such that $c_k > y^T A_k$ in addition to our new requirement that the sum of the coeffients is less than 8 (the maximum number of knife cuts). We will do this be ensuring that the patterns do not include more than 8 cuts. This is further achieved by capping all patterns at 8 cuts. The ratio test will therefore be effected and we will have to select the k by choosing the minumum ratio with this new constraint.

(f) *Modify the AMPL file and solve this new problem. Feel free to come for help there is a lot of new syntax in this AMPL file, and there is no need to spend a lot of time figuring it out.*

The modified AMPL program is available for review in **Appendix C**.

(g) *Report the optimal value of the LP relaxation for the modified problem.* The optimal solution for the relaxed LP is $x^{*T} = \begin{bmatrix} 11.25 & 6.375 & 5.625 & 0 & 1.1 \end{bmatrix}$. This gives us an optimal Raw usage of 24.35. The complete AMPL output is available in **Appendix C**.

(h) *Use the previous part to give an integer solution to the problem. Explain what you know about the quality of the solution (is it guaranteed to be optimal? If not, how far off can it be?)*

In order to derive an integer solution from the noninteger optimal solution, we must round all noninteger values up.

This makes the derived integer solution:$x^{*T} = \begin{bmatrix} 12 & 7 & 6 & 0 & 2 \end{bmatrix}$. This integer solution gives us a objective value of 27. We cannot guarantee that this integer objective value is optimal since the value is more than one integer greater than the noninteger objective value. We can say that it is at most off by two since the lowest integer value above the noninteger optimal objective is 25.

Appendix A

AMPL Data File, Problem 1:

```
param roll_width := 100 ;
param m := 4;
param : width orders :=
1 52 600
2 29 600
3 27 600
4 21 1200;
```

AMPL Model File, Problem 1:

```
# ----------------------------------------
# CUTTING STOCK USING PATTERNS
# ----------------------------------------
param roll_width > 0;          # width of raw rolls
param m; # number of final types
set FINALTYPES = 1..m;
param width {FINALTYPES};       # widths of each final type
param orders {FINALTYPES};     # number of orders for each final type
param nPAT integer >= 0;       # number of patterns
set PATTERNS := 1..nPAT;       # set of patterns
param nbr {FINALTYPES,PATTERNS} integer >= 0;  #nbr(i,j)= number of times final type i appears
   check {j in PATTERNS}:
      sum {i in FINALTYPES} width[i] * nbr[i,j] <= roll_width;
                            # Note that nbr is a parameter, not a variable,
                            # and Check_Fit is not a constraint, but a check before solving
var Cut {PATTERNS} integer >= 0;    # rolls cut using each pattern
minimize Number_Cut : # minimize total raw rolls cut
   sum {j in PATTERNS} Cut[j];
subj to Fill_Orders {i in FINALTYPES}: #fill the orders for each final type
   sum {j in PATTERNS} nbr[i,j] * Cut[j] = orders[i];
# ----------------------------------------
# KNAPSACK SUBPROBLEM FOR CUTTING STOCK
# ----------------------------------------
param price {FINALTYPES} default 0.0;
var Use {FINALTYPES} integer >= 0;
maximize Reduced_Cost:
   sum {i in FINALTYPES} price[i] * Use[i] - 1;
subj to Width_Limit:
   sum {i in FINALTYPES} width[i] * Use[i] <= roll_width;
```

AMPL Run File, Problem 1:

```
# ----------------------------------------
# GILMORE-GOMORY METHOD FOR
```

```
# CUTTING STOCK PROBLEM
# ----------------------------------------
option solver gurobi;
model cut.mod;
data cut.dat;
problem Cutting_Opt: Cut, Number_Cut, Fill_Orders;
   option relax_integrality 1;
problem Pattern_Gen: Use, Reduced_Cost, Width_Limit;
   option relax_integrality 0;
let nPAT := 0;
for {i in FINALTYPES} {
   let nPAT := nPAT + 1;
   let nbr[i,nPAT] := floor (roll_width/width[i]);
   let {i2 in FINALTYPES: i2 <> i} nbr[i2,nPAT] := 0;
   };
repeat {
   solve Cutting_Opt;
   let {i in FINALTYPES} price[i] := Fill_Orders[i].dual;
   solve Pattern_Gen;
   if Reduced_Cost > 0.00001 then {
      let nPAT := nPAT + 1;
      let {i in FINALTYPES} nbr[i,nPAT] := Use[i];
      }
   else break;
   };
display nbr;
display Cut;
```

Appendix B

AMPL Output, Problem 2, Part C:

```
Gurobi 5.6.3: optimal solution; objective 0.08611111111
4 simplex iterations
Gurobi 5.6.3: optimal solution; objective 23.09761905
Gurobi 5.6.3: optimal solution; objective 0.0376984127
29 simplex iterations
48 branch-and-cut nodes
Gurobi 5.6.3: optimal solution; objective 23.07979798
2 simplex iterations
Gurobi 5.6.3: optimal solution; objective 0.02398989899
11 simplex iterations
10 branch-and-cut nodes
Gurobi 5.6.3: optimal solution; objective 23.04861111
2 simplex iterations
Gurobi 5.6.3: optimal solution; objective 0.006944444444
25 simplex iterations
```

```
25 branch-and-cut nodes
Gurobi 5.6.3: optimal solution; objective 23.00932836
3 simplex iterations
Gurobi 5.6.3: optimal solution; objective 0.003731343284
32 simplex iterations
36 branch-and-cut nodes
Gurobi 5.6.3: optimal solution; objective 23
4 simplex iterations
Gurobi 5.6.3: optimal solution; objective 0
1 simplex iterations
nbr [*,*] (tr)
:   1   2   3   4     :=
1   8   0   0    0
2   0   8   0    0
3   0   0   9    0
4   0   0   0   10
5   0   7   1    1
6   1   1   0    8
7   5   1   0    3
8   0   2   7    0
9   4   0   3    2
;

Cut [*] :=
1  10
2   0
3   0
4   0
5   6
6   0
7   0
8   4.5
9   2.5
;
```

AMPL Data File, Problem 2, Part c:

```
param roll_width := 181 ;
param m := 4;
param : width orders :=
1 21.625 90
2 20.5 51
3 20 45
4 17.25 11;
```

AMPL Model File, Problem 2, Part C:

```
# ---------------------------------------
# CUTTING STOCK USING PATTERNS
# ---------------------------------------
param roll_width > 0;          # width of raw rolls
param m; # number of final types
set FINALTYPES = 1..m;
param width {FINALTYPES};       # widths of each final type
param orders {FINALTYPES};     # number of orders for each final type
param nPAT integer >= 0;       # number of patterns
set PATTERNS := 1..nPAT;       # set of patterns
param nbr {FINALTYPES,PATTERNS} integer >= 0;  #nbr(i,j)= number of times final type i appears
   check {j in PATTERNS}:
      sum {i in FINALTYPES} width[i] * nbr[i,j] <= roll_width;
                            # Note that nbr is a parameter, not a variable,
                            # and Check_Fit is not a constraint, but a check before solving
var Cut {PATTERNS} integer >= 0;    # rolls cut using each pattern
minimize Number_Cut : # minimize total raw rolls cut
   sum {j in PATTERNS} Cut[j];
subj to Fill_Orders {i in FINALTYPES}: #fill the orders for each final type
   sum {j in PATTERNS} nbr[i,j] * Cut[j] = orders[i];
# ---------------------------------------
# KNAPSACK SUBPROBLEM FOR CUTTING STOCK
# ---------------------------------------
param price {FINALTYPES} default 0.0;
var Use {FINALTYPES} integer >= 0;
maximize Reduced_Cost:
   sum {i in FINALTYPES} price[i] * Use[i] - 1;
subj to Width_Limit:
   sum {i in FINALTYPES} width[i] * Use[i] <= roll_width;
```

AMPL Run File, Problem 2, Part C:

```
# ---------------------------------------
# GILMORE-GOMORY METHOD FOR
# CUTTING STOCK PROBLEM
# ---------------------------------------
option solver gurobi;
model cut.mod;
data cut.dat;
problem Cutting_Opt: Cut, Number_Cut, Fill_Orders;
   option relax_integrality 1;
problem Pattern_Gen: Use, Reduced_Cost, Width_Limit;
   option relax_integrality 0;
let nPAT := 0;
for {i in FINALTYPES} {
   let nPAT := nPAT + 1;
   let nbr[i,nPAT] := floor (roll_width/width[i]);
```

```
   let {i2 in FINALTYPES: i2 <> i} nbr[i2,nPAT] := 0;
   };
repeat {
   solve Cutting_Opt;
   let {i in FINALTYPES} price[i] := Fill_Orders[i].dual;
   solve Pattern_Gen;
   if Reduced_Cost > 0.00001 then {
      let nPAT := nPAT + 1;
      let {i in FINALTYPES} nbr[i,nPAT] := Use[i];
      }
   else break;
   };
display nbr;
display Cut;
```

Appendix C

AMPL Data File:

```
param roll_width := 181 ;
param m := 4;
param : width orders :=
1 21.625 90
2 20.5 51
3 20 45
4 17.25 11;
```

AMPL Model File:

```
# ----------------------------------------
# CUTTING STOCK USING PATTERNS
# ----------------------------------------
param roll_width > 0;          # width of raw rolls
param m; # number of final types
set FINALTYPES = 1..m;
 param width {FINALTYPES};       # widths of each final type
param orders {FINALTYPES};     # number of orders for each final type
param nPAT integer >= 0;       # number of patterns
set PATTERNS := 1..nPAT;       # set of patterns
param nbr {FINALTYPES,PATTERNS} integer >= 0;  #nbr(i,j)= number of times final type i appears
   check {j in PATTERNS}:
      sum {i in FINALTYPES} width[i] * nbr[i,j] <= roll_width;
                            # Note that nbr is a parameter, not a variable,
                            # and Check_Fit is not a constraint, but a check before solving
var Cut {PATTERNS} integer >= 0;    # rolls cut using each pattern
minimize Number_Cut : # minimize total raw rolls cut
   sum {j in PATTERNS} Cut[j];
subj to Fill_Orders {i in FINALTYPES}: #fill the orders for each final type
```

```
   sum {j in PATTERNS} nbr[i,j] * Cut[j] = orders[i] ;
# ----------------------------------------
# KNAPSACK SUBPROBLEM FOR CUTTING STOCK
# ----------------------------------------
param price {FINALTYPES} default 0.0;
var Use {FINALTYPES} integer >= 0;
maximize Reduced_Cost:
   sum {i in FINALTYPES} price[i] * Use[i] - 1;
subj to Width_Limit:
   sum {i in FINALTYPES} width[i] * Use[i] <= roll_width;
subj to Max_Cuts {j in PATTERNS}:
sum{i in FINALTYPES} nbr[i,j] <= 8;
```

AMPL Run File:

```
# ----------------------------------------
# GILMORE-GOMORY METHOD FOR
# CUTTING STOCK PROBLEM
# ----------------------------------------
option solver gurobi;
model cut.mod;
data cut.dat;
problem Cutting_Opt: Cut, Number_Cut, Fill_Orders;
   option relax_integrality 1;
problem Pattern_Gen: Use, Reduced_Cost, Width_Limit,Max_Cuts;
   option relax_integrality 0;
let nPAT := 0;
for {i in FINALTYPES} {
   let nPAT := nPAT + 1;
   let nbr[i,nPAT] := floor (roll_width/width[i]);
   if nbr[i,nPAT]>= 8 then {
    let nbr[i,nPAT] := 8;};
   let {i2 in FINALTYPES: i2 <> i} nbr[i2,nPAT] := 0;
   };
repeat {
   solve Cutting_Opt;
   let {i in FINALTYPES} price[i] := Fill_Orders[i].dual;
      solve Pattern_Gen;
   if Reduced_Cost > 0.00001 then {
      let nPAT := nPAT + 1;
      let {i in FINALTYPES} nbr[i,nPAT] := Use[i];
      }
   else break;
   };
display nbr;
display Cut;
```

AMPL Output:

```
Gurobi 5.6.3: optimal solution; objective 0.25
Gurobi 5.6.3: optimal solution; objective 24.35
presolve, constraint Max_Cuts[5]:
no variables, but lower bound = -Infinity, upper = -2
nbr :=
1 1    8
1 2    0
1 3    0
1 4    0
1 5    0
2 1    0
2 2    8
2 3    0
2 4    0
2 5    0
3 1    0
3 2    0
3 3    8
3 4    0
3 5    0
4 1    0
4 2    0
4 3    0
4 4    8
4 5   10
;

Cut [*] :=
1  11.25
2   6.375
3   5.625
4   0
5   1.1
;
```