```
! LINGO model for
  Rectangular 2D cutting stock with guillotine cuts.
 Given:
  Raw material master rectangle of dimensions x1 by y1,
  and finished good (f.g.) rectangles i, of dimensions dx(i) by dy(i),
 Find a sequence of guillotine cuts x(j) and y(j), so as to
 maximize the value of the f.g. rectangles cut from the master.
   Cutting process is viewed as a binary tree, with
 node i having the two children: 2*i and 1+2*i.
 Each cut splits a rectangle into two smaller
 rectangles. Material may be isotropic, (rotate = 1)such as glass,
 so finished good may be rotated 90 degrees, or
 non-isotropic (rotate = 0) such as a heavily grained wood;
! Some of the ideas in this formulation were presented by Dyckhoff;
!Ref:
  Dyckhoff, H. (1981),"A New Linear Programming Approach to the Cutting Stock Problem,"
Operations Research, vol 29, no. 6, pp. 1092-1104.;
! Keywords: Cutting stock, Dyckhoff, Guillotine cut, LINGO, Packing, Rectangle, Two-
dimensional cutting stock;
sets:
 piece: x, y, uflag,
    ubx, uby;
 fg: dx, dy, v, nc, nr;
 pxf( piece, fg): zn, zr;
endsets

data:! A small problem to illustrate ideas;
!Case04; x1= 2800 ;
!Case04; y1= 2450 ;
!Case04;
      fg     dx     dy  v  nc =
      MV    330    996  1  2
Vanities    510    296  1  3
;

! Number of nodes in tree, should be of order
  2^(number of cuts);
!Case04;   piece = 1..127;
! Parameters:
    nterm = lowest indexed terminal node of the tree.
    A f.g. can be assigned only to a terminal node;
!Case04;  nterm = 64;
! Set rotate = 1 if rotation allowed (isotropic),
             = 0 otherwise (non-isotropic);
!Case04;  rotate = 0;

 ! Raw material dimensions. Standard window glass;
!Case01   x1 = 100;
!Case01   y1 = 200; ! The raw material;
! Finished good dimensions, and their values, and max number copies useful;
!Case01
  fg  dx  dy  v  nc =
  w01  45  50  1   1
  w02  55  50  1   1
  w03  15 150  1   1
  w04  35  65  1   1
  w05  50  40  1   1
  w06  50  25  1   1
  w07  40  45  1   1
  w08  45  45  1   1
  w09  35  40  1   1
  w10  20  40  1   1
  w11  30  40  1   1;
! ???Optimal obj =  if rotation is allowed,
               =  if no rotation allowed;
! Number of nodes in tree, should be of order
  2^(number of cuts);
!Case01   piece = 1..127;
```

```
! Parameters:
    nterm = lowest indexed terminal node of the tree.
    A f.g. can be assigned only to a terminal node;
!Case01   nterm = 64;
! Set rotate = 1 if rotation allowed (isotropic),
             = 0 otherwise (non-isotropic);
!Case01     rotate = 0;


! Raw material dimensions. Standard window glass;
!Case02  x1 = 84;
!Case02  y1 = 72; ! The raw material;
! Finished good dimensions, and their values, and max number copies useful;
!Case02
  fg dx dy  v  nc =
  w1 35 41  1   2
  w2 16 43  1   1
  w3 36 18  1   1
  w4 36 19  1   1
  w5 35 14  1   1
  w6 37 13  1   1
  w7 43 19  1   2
  w8 23 35  1   1;

! ???Optimal obj = 7 if rotation is allowed,
              = 7 if no rotation allowed;

! Number of nodes in tree, should be of order
  2^(number of cuts);
!!Case02   piece = 1..127;
! Parameters:
    nterm = lowest indexed terminal node of the tree.
    A f.g. can be assigned only to a terminal node;
!!Case02  nterm = 64;
! Set rotate = 1 if rotation allowed (isotropic),
             = 0 otherwise (non-isotropic);
!Case02  rotate = 0;

!Case03  x1= 2800 ;
!Case03  y1= 2450 ;
!Case03
      fg     dx     dy  v  nc =
      MV    330    996  1  2
Vanities    510    296  1  8
   Staff    750    576  1  1
Scoop1a    2740     45  1  2
Scoop1b    2740     50  1  2
Fill        780    150  1  1
;

! Optimal obj = 16 if rotation is allowed,
              = 7 if no rotation allowed;

! Number of nodes in tree, should be of order
  2^(number of cuts);
!Case03    piece = 1..127;
! Parameters:
    nterm = lowest indexed terminal node of the tree.
    A f.g. can be assigned only to a terminal node;
!Case03   nterm = 64;
! Set rotate = 1 if rotation allowed (isotropic),
             = 0 otherwise (non-isotropic);
!Case03   rotate = 1;
enddata
! Maximize the value of fg's assigned.
  zn(i,j) = 1 if final node i is assigned non-rotated fg j.
  zr(i,j) = 1 if final node i is assigned rotated fg j,
   x(i) = horizontal dimension of the piece at node i,
```

```
   y(i) = vertical dimension of the piece at node i;
! This model is useful for modest size problems, e.g., < 8 fg's.
  Thereafter, solution time increases rapidly with problem size;
! Basic idea: Create a tree of rectangles. Root node is original rectangle.
   Each cut of a rectangle creates two new rectangles.
;

SUBMODEL GENGPAT:
! Kill some symmetry by setting size bounds on rectangles;
  max = @sum( pxf(i,j)| (i #ge# nterm)
       #AND# (dx(j) #LE# UBX(i)) #AND# ( dy( j) #LE# UBY(i))
     : v(j)*(zn(i,j)+ zr(i,j)));
! The z(i,j)'s are binary;
  @for( pxf(i,j):
     @bin( zn(i,j)); @bin(zr(i,j));
     zr(i,j) <= rotate;
     );

! The first node is the original raw material;
   x(1) = x1;  y(1) = y1;
   ubx(1) = x1; uby(1) = y1;

! We make horizontal cuts to get to these nodes;
@for( piece(i)| ( i #ge#   2 #and# i #lt#    4) #or#
                ( i #ge#   8 #and# i #lt#   16) #or#
                ( i #ge#  32 #and# i #lt#   64) #or#
                ( i #ge# 128 #and# i #lt#  256) #or#
                ( i #ge# 512 #and# i #lt# 1024) :
! So the horizontal length is unchanged;
 [H1]    x(i) = x(@floor(i/2));
  @for( piece(i1)| i1 #eq# i #and# i #eq# 2*@floor(i/2):
!  Verticl dimension is split into 2 parts;
   [HT] y(i) + y( i+1) = y(i/2);
! Kill some symmetry, Force the lower, even-numbered
  twin to be the smaller one;
        uby(i) = uby(i/2)/2;
        ubx(i) = ubx(i/2);
        uby(i+1) = uby(i/2);
        ubx(i+1) = ubx(i/2);
 [HS]    y( i) <= y(i/2)/2;
      );
    );

! We make vertical cuts to get to these nodes;
@for( piece(i)| ( i #ge#    4 #and# i #lt#    8) #or#
                ( i #ge#   16 #and# i #lt#   32) #or#
                ( i #ge#   64 #and# i #lt#  128) #or#
                ( i #ge#  256 #and# i #lt#  512) #or#
                ( i #ge# 1024 #and# i #lt# 2048)  :
! So the vertical length is unchanged;
   [V1]  y(i) = y(@floor(i/2));
  @for(piece(i1)|i1 #eq# i #and#  i1 #eq# 2*@floor(i/2):
  [VT]   x(i) + x( i+1) = x(i/2);
! Kill some symmetry, Force the lower, even-numbered
  twin to be the smaller one;
        ubx(i) = ubx(i/2)/2;
        uby(i) = uby(i/2);
        ubx(i+1) = ubx(i/2);
        uby(i+1) = uby(i/2);
  [VS]   x(i) <= x(i/2)/2;
      );
    );

! A finished good can be assigned only to a terminal node
   and to a node in which it fits;
  @sum( pxf( i, j)| ( i #lt# nterm)
     ! #OR# (dx(j) #GT# UBX(i)) #OR# (dy(j) #GT# uby(i));
     : zn(i,j) + zr(i,j)) = 0;
```

```
  @for( piece(i)| i #ge# nterm:
! At most one finished good to a node;
 [FN]   @sum( pxf(i,j): zn( i, j) + zr( i, j)) <= 1;
      );
! Each finished good assigned at most nc(j) times;
  @for( fg(j):
 [F1] @sum( pxf(i,j): zn(i,j)+zr(i,j)) = nr(j);
      @BND( 0, nr(j), nc(j));
      );
 @for( piece(i) | i #ge# nterm:
! If piece i satisfies fg j with no rotation, then dimensions match;
 [N1]   x(i) >= @SUM( fg(j): dx(j)*zn(i,j));
 [N2]   y(i) >= @SUM( fg(j): dy(j)*zn(i,j));
 [N3]   x(i) <= x1 - @SUM( fg(j): (x1-dx(j))*zn(i,j));
 [N4]   y(i) <= y1 - @SUM( fg(j): (y1-dy(j))*zn(i,j));
! If piece i satisfies fg j with rotation, then dimensions match;
 [R1]   y(i) >= @SUM( fg(j): dx(j)*zr(i,j));
 [R2]   x(i) >= @SUM( fg(j): dy(j)*zr(i,j));
 [R3]   y(i) <= y1 - @SUM( fg(j): (y1-dx(j))*zr(i,j));
 [R4]   x(i) <= x1 - @SUM( fg(j): (x1-dy(j))*zr(i,j));
      );

! Some cuts;
! Knapsack, from which further cuts can be derived.
  The area of rectangles produced must be <= area of raw material;
   @sum(pxf(i,j): dx(j)*dy(j)*(zn(i,j) + zr(i,j))) <= x1*y1;

! Kill some symmetry.
   The bigger width must be assigned to the higher twin;
 @for( pxf(i,j)| i #ge# nterm #AND# i #eq# 2*@floor(i/2):
    @sum(fg(j): dx(j)*zn(i,j)   + dy(j)*zr(i,j)) <=
    @sum(fg(j): dx(j)*zn(i+1,j) + dy(j)*zr(i+1,j))
     );
ENDSUBMODEL

CALC:
  @SET( 'OROUTE',1); ! Buffer size for routing output to window;
  @SET( 'TERSEO',2); ! Output level (0:verbose, 1:terse, 2:only errors, 3:none);
  @SET( 'IPTOLR', 0.0001);! Set IP ending relative optimality tolerance(Should be >0);
  @SET( 'TIM2RL', 30);! Time in seconds to apply IPTOLR tolerance;
  @SOLVE( GENGPAT);
  ISTAT = @STATUS();! 0: Optimal to tolerance. 1: infeasible, 2: unbounded,
                     3: undetermined, 4: Feasible,  5: Infeasible/unbounded in
preprocessor,
                     6: Local optimum, 7: locally infeasible, 8: Objective cutoff reached,
                     9: numeric error;
  @WRITE(' Solve status= ', ISTAT, @NEWLINE( 1));

 ! Set uflag(i) > 1 if any fg cut from subrectangle i;
 @FOR( PIECE(i): uflag(i) = 0);
 iset = @SIZE( PIECE);
 @FOR( PIECE(i) | i #GT# 1:
   @IFC( iset #GE# nterm:
     ! Guard against small round-off error;
       uflag(iset) = @FLOOR( 0.5+@SUM( fg(j) : j*(zn(iset,j)+zr(iset,j))));
      );
    parent = @FLOOR( iset/2);
    uflag(parent) = uflag(parent)+uflag(iset);
    iset = iset -1;
     );

 @WRITE(@NEWLINE(1),' Input data:', @NEWLINE(1));
 @WRITE(' Raw Material   Length   Width', @NEWLINE(1));
 @WRITE( '                ',  @FORMAT( x1,"6.1f"),'   ',@FORMAT( y1,"6.1f"), @NEWLINE(2));
 @WRITE(' Finish Good    Length   Width  Copies', @NEWLINE(1));
  @FOR( FG( f):
   @WRITE( @FORMAT( FG( f), '12s'), '    ',@FORMAT( x(f),"6.1f"),'   ',
```

```
@FORMAT(y(f),"6.1f"),'    ', nc( f),@NEWLINE(1));
      );
  @IFC( rotate #EQ# 1:
    @WRITE(' Rotation allowed (No grain)', @NEWLINE( 2));
   @ELSE
    @WRITE(' Rotation not allowed (Has grain)', @NEWLINE( 2));
     );

  @WRITE( ' Total area available: ', @FORMAT(   x1*y1, '12.1f'), @NEWLINE( 1));
  totneed = @SUM( fg( f): dx( f) * dy( f));
  @WRITE( '    Total area needed: ', @FORMAT( totneed, '12.1f'), @NEWLINE( 2));



 @WRITE('       Input/             ', @NEWLINE(1));
 @WRITE('  Cut  Parent   Output     Dimensions', @NEWLINE(1));
 @WRITE(' type  rect.    Rectangle   X      Y       Finals', @NEWLINE(1));
 @WRITE('        -          1      ', @FORMAT( x1,"6.1f"),'   ',@FORMAT( y1,"6.1f"),
@NEWLINE(1));
  areaused = 0;  ! Useful area obtained;
@for( piece(i)| i #GT# 1 #AND# uflag(i) #GT# 0:
   parent = @FLOOR(i/2);
   @IFC(( i #ge#  2 #and# i #lt#  4) #or#
                ( i #ge#  8 #and# i #lt# 16) #or#
                ( i #ge# 32 #and# i #lt# 64):
   ! We make horizontal cuts to get to these nodes;
      @IFC( y(i) #GT# 0 #AND# x(i) #GT# 0:  ! Is it a non-trivial cut...;
        @WRITE( '  H   ', @FORMAT( parent,"3.0f"),'        ',  @FORMAT(i,"3.0f"),
             '      ',@FORMAT(x(i),"6.1f"),'   ',@FORMAT(y(i),"6.1f"),@NEWLINE(1));
          );
       @ELSE
        @IFC( x(i) #GT# 0 #AND# y(i) #GT# 0:  ! Is it a non-trivial cut...;
        @WRITE( '  V   ', @FORMAT(parent,"3.0f"), '        ', @FORMAT(i,"3.0f"),
           '      ',@FORMAT(x(i),"6.1f"),'   ',@FORMAT( y(i),"6.1f"));
        @IFC( i #GE# nterm:
           @WRITE(' ',fg( uflag(i)),@NEWLINE(1));
           areaused = areaused + dx(uflag(i)) * dy( uflag(i));
         @ELSE
           @WRITE(@NEWLINE(1));
            );
        );
      );
    );
 ! @write( ' Total area used= ', areaused,,' out of available area of ', x1*y1,
@newline(1));
ENDCALC
```