# Layout Optimization Model for Multi-Recipe and Multi-Route Problems with Application to the Design of a Steel Factory

MILP model for optimizing the left / right split of scrap types' boxes

In [1]:
```
import pandas as pd
```

In [2]:
```
from datetime import datetime
```

In [3]:
```
from IPython.core.magic import register_cell_magic

@register_cell_magic
def skip(line, cell):
    return
```

## DATA

#### Scrapyard data

Width of the 40m deep box for each scrap type

In [4]:
```
box_width_dict = {0: 11.4, 1: 10.0, 2: 11.7, 3: 10.0, 4: 10.0, 5: 10.0, 6: 23.4,
                  7: 10.0, 8: 34.4, 9: 12.4, 10: 10.0, 11: 10.0, 12: 38.3,
                  13: 26.5, 14: 56.9, 15: 43.6, 16: 27.5, 17: 14.3, 18: 69.3,
                  19: 27.2, 20: 56.1, 21: 38.4, 22: 26.0, 23: 10.0, 24: 33.6,
                  25: 31.3, 26: 14.4, 27: 30.9, 28: 16.0, 29: 10.0, 30: 14.3,
                  31: 15.5}
```

#### Grades data

Each grade contains a given quantity of certain scrap types

In [5]:
```
grade_ingredients_dict = {'G01': [6, 8, 4, 5, 0, 1, 2, 7, 9, 16, 20],
 'G02': [6, 8, 4, 5, 0, 1, 3, 7, 16, 20],
 'G03': [6, 8, 4, 0, 3, 7, 9, 16, 20],
 'G04': [6, 8, 18, 4, 5, 7, 9, 11, 17, 30, 16, 19, 20, 27],
```

```
'G05': [8, 5, 0, 1, 7, 17, 16, 20],
'G06': [6, 8, 0, 1, 3, 9, 16, 20],
'G07': [6, 8, 18, 4, 5, 0, 1, 3, 16, 20],
'G08': [8, 28, 22, 15, 17, 14, 16, 21, 19, 24, 27],
'G09': [12, 13, 15, 29, 14, 19, 24],
'G10': [6, 8, 7, 15, 14, 19, 24],
'G11': [22, 23, 15, 21, 19, 20, 24],
'G12': [12, 13, 18, 28, 0, 1, 2, 3, 29, 30, 16, 31, 20, 24],
'G13': [12, 13, 15, 29, 19, 24, 25],
'G14': [12, 15, 29, 14, 19, 24],
'G15': [6, 8, 28, 5, 16, 19, 20],
'G16': [6, 8, 5, 7, 11, 15, 29, 14, 20, 24],
'G17': [12, 13, 15, 29, 14, 19, 24, 25],
'G18': [6, 8, 12, 13, 0, 9, 15, 29, 14, 19, 24, 25],
'G19': [12, 15, 20, 24],
'G20': [8, 12, 13, 15, 29, 19, 24, 25],
'G21': [6, 8, 10, 4, 5, 9, 11, 15, 20, 24],
'G22': [8, 10, 5, 7, 15, 29, 14, 20, 24],
'G23': [8, 12, 15, 20],
'G24': [6, 8, 12, 5, 7, 9, 15, 14, 20, 24],
'G25': [6, 8, 4, 5, 7, 11, 15, 14, 20, 24],
'G26': [12, 13, 15, 20, 24, 25],
'G27': [6, 8, 10, 12, 4, 5, 7, 9, 15, 20, 24],
'G28': [6, 8, 5, 0, 3, 9, 17, 16, 20, 27],
'G29': [12, 23, 15, 21, 19, 24],
'G30': [12, 15, 29, 14, 21, 19, 24],
'G31': [12, 13, 23, 15, 29, 21, 19, 24, 25],
'G32': [12, 22, 15, 21, 19, 24, 25],
'G33': [12, 13, 22, 23, 15, 29, 14, 21, 19, 24, 25],
'G34': [6, 8, 5, 0, 1, 2, 3, 17, 16, 20],
'G35': [8, 10, 28, 5, 16, 31, 20, 27],
'G36': [8, 10, 28, 4, 11, 15, 17, 16, 20, 27],
'G37': [6, 8, 18, 4, 5, 15, 30, 16, 20, 27],
'G38': [12, 15, 29, 14, 19, 24, 25],
'G39': [12, 22, 23, 15, 29, 21, 19, 24, 25],
'G40': [6, 8, 18, 28, 4, 7, 9, 15, 17, 16, 20, 27],
'G41': [8, 12, 13, 22, 15, 29, 14, 21, 19, 24, 25],
'G42': [12, 23, 15, 29, 14, 21, 19, 24],
'G43': [12, 13, 15, 21, 19, 24, 25],
'G44': [12, 15, 24, 25],
'G45': [23, 15, 21, 19, 20, 24],
'G46': [12, 13, 28, 15, 17, 29, 14, 16, 19, 24, 25],
'G47': [12, 15, 24],
'G48': [12, 0, 15, 29, 14, 24],
```

```
 'G49': [12, 15, 29, 14, 24],
 'G50': [8, 18, 28, 0, 2, 23, 17, 30, 16, 21, 20, 27],
 'G51': [18, 26, 28, 16, 19],
 'G52': [6, 8, 10, 12, 18, 28, 0, 2, 9, 16, 20, 25],
 'G53': [12, 15, 14, 24, 25],
 'G54': [12, 13, 15, 29, 14, 24],
 'G55': [12, 15, 24, 25],
 'G56': [6, 8, 18, 28, 0, 3, 9, 16, 20],
 'G57': [12, 26, 19, 25],
 'G58': [6, 8, 10, 28, 11, 16, 19, 24],
 'G59': [6, 8, 4, 9, 11, 30, 16, 19, 24],
 'G60': [8, 12, 13, 28, 16, 19, 20],
 'G61': [8, 10, 18, 4, 5, 11, 17, 16, 19, 24],
 'G62': [12, 15, 29, 14, 24],
 'G63': [15, 14, 20, 24],
 'G64': [12, 13, 15, 14, 19, 24],
 'G65': [12, 13, 15, 29, 14, 19, 24, 25],
 'G66': [12, 13, 15, 14, 19, 24, 25],
 'G67': [15, 29, 21, 24],
 'G68': [6, 8, 12, 5, 15, 19, 24],
 'G69': [12, 13, 21, 19, 24, 25],
 'G70': [12, 22, 21, 20, 24],
 'G71': [12, 15, 16, 24, 25],
 'G72': [0, 3, 15, 14, 16, 24],
 'G73': [12, 18, 0, 1, 15, 14, 16, 24, 27],
 'G74': [12, 13, 15, 16, 24],
 'G75': [8, 4, 5, 0, 1, 2, 7, 9, 16, 20],
 'G76': [6, 8, 0, 3, 16, 20],
 'G77': [8, 4, 0, 1, 3, 9, 16, 20],
 'G78': [6, 8, 4, 0, 3, 7, 9, 16, 20],
 'G79': [6, 8, 18, 28, 0, 1, 16, 20],
 'G80': [18, 0, 2, 17, 30, 16, 31, 20, 27],
 'G81': [18, 28, 0, 17, 30, 16, 31, 20, 27],
 'G82': [0, 3, 30, 16, 20, 27],
 'G83': [18, 0, 17, 30, 16, 20]}
```

The quantity of the scrap type required determines how many grabs the crane has to do

In [6]:
```
total_grabs_dict = {'G01': 25, 'G02': 25, 'G03': 28, 'G04': 36, 'G05': 32, 'G06': 26, 'G07': 32, 'G08': 46, 'G09': 33, 'G10':
                    'G11': 27, 'G12': 47, 'G13': 30, 'G14': 30, 'G15': 31, 'G16': 40, 'G17': 32, 'G18': 44, 'G19': 26, 'G20':
                    'G21': 38, 'G22': 37, 'G23': 28, 'G24': 37, 'G25': 37, 'G26': 30, 'G27': 36, 'G28': 34, 'G29': 41, 'G30':
                    'G31': 42, 'G32': 39, 'G33': 38, 'G34': 40, 'G35': 43, 'G36': 46, 'G37': 45, 'G38': 31, 'G39': 40, 'G40':
                    'G41': 37, 'G42': 32, 'G43': 34, 'G44': 22, 'G45': 24, 'G46': 34, 'G47': 22, 'G48': 23, 'G49': 21, 'G50':
```

```
                     'G51': 41, 'G52': 43, 'G53': 23, 'G54': 25, 'G55': 22, 'G56': 31, 'G57': 22, 'G58': 42, 'G59': 39, 'G60':
                     'G61': 38, 'G62': 21, 'G63': 16, 'G64': 30, 'G65': 32, 'G66': 34, 'G67': 27, 'G68': 24, 'G69': 38, 'G70':
                     'G71': 22, 'G72': 26, 'G73': 31, 'G74': 23, 'G75': 26, 'G76': 25, 'G77': 27, 'G78': 28, 'G79': 28, 'G80':
                     'G81': 32, 'G82': 33, 'G83': 32}
```

Based on the production plan, there is a maximum time available for the loading of each grade

In [7]:
```
grades_max_time_dict = {'G07': 2821000.0, 'G08': 4600000.0, 'G09': 4464000.0, 'G11': 2581000.0,
                        'G13': 3541000.0, 'G28': 5076000.0, 'G40': 4260000.0, 'G44': 3541000.0,
                        'G46': 4146331.0, 'G47': 3421000.0, 'G64': 3990000.0, 'G65': 3601000.0,
                        'G67': 3298718.0, 'G78': 2581000.0}
```

**Scrap types data**

The layer of each scrap type

In [8]:
```
scrap_layer_dict = {6: 1.0, 8: 1.0, 10: 1.0, 12: 1.0, 13: 1.0, 18: 1.0, 26: 1.0, 28: 2.0, 4: 3.0, 5: 3.0,
                    0: 4.0, 1: 4.0, 2: 4.0, 3: 4.0, 7: 4.0, 9: 4.0, 11: 4.0, 22: 4.0, 23: 4.0, 15: 5.0,
                    17: 5.0, 29: 5.0, 30: 5.0, 14: 6.0, 16: 6.0, 21: 6.0, 31: 6.0, 19: 7.0, 20: 7.0, 24: 7.0,
                    25: 7.0, 27: 7.0}
```

**Production plan data**

In [9]:
```
jobs_eaf1_list = ['G11', 'G11', 'G11', 'G40', 'G40', 'G40', 'G11', 'G11', 'G11', 'G28', 'G28', 'G28', 'G09', 'G09', 'G09',
                  'G11', 'G11', 'G11', 'G07', 'G07', 'G07', 'G28', 'G28', 'G28', 'G46', 'G46', 'G46', 'G46', 'G46', 'G07',
                  'G07', 'G07', 'G47', 'G47', 'G47', 'G47', 'G47', 'G47', 'G67', 'G67', 'G67', 'G67', 'G67', 'G78', 'G78',
                  'G78', 'G13', 'G13', 'G13', 'G13', 'G13', 'G13', 'G64', 'G64', 'G64', 'G64', 'G64', 'G65', 'G65', 'G65',
                  'G65', 'G65', 'G65', 'G44', 'G44', 'G44', 'G44', 'G44', 'G67', 'G67', 'G67', 'G67', 'G67', 'G47', 'G47',
                  'G47', 'G47', 'G47', 'G11', 'G11', 'G11', 'G11', 'G11', 'G65', 'G65', 'G65']
```

In [10]:
```
jobs_eaf2_list = ['G11', 'G11', 'G11', 'G40', 'G40', 'G40', 'G11', 'G11', 'G11', 'G28', 'G28', 'G28', 'G09', 'G09', 'G09',
                  'G11', 'G11', 'G11', 'G07', 'G07', 'G07', 'G28', 'G28', 'G28', 'G07', 'G07', 'G78', 'G78', 'G78', 'G08',
                  'G08', 'G08', 'G47', 'G47', 'G78', 'G78', 'G78', 'G11', 'G11', 'G65', 'G65', 'G65']
```

In [11]:
```
# Grades that appear in the production plan
grades_list = ['G07', 'G08', 'G09', 'G11', 'G13', 'G28', 'G40', 'G44', 'G46', 'G47', 'G64', 'G65', 'G67', 'G78']
```

In [11]:

## FUNCTIONS TO GET INFO FROM OUTPUT

### Get left / right split

In [12]:
```
def get_split(scraps):
    left_side = []
    right_side = []

    for i in scraps:
      if side[i].X == 0:
        left_side.append(i)
      else:
        right_side.append(i)

    return left_side, right_side
```

In [12]:

### Get length of yard side

In [13]:
```
def compute_side_length(side):
    length = 20*2 + 1.7
    for scrap in side:
      length += box_width_dict[scrap] + 1.7

    return length
```

In [14]:
```
def compute_yard_length(L, R):

    return max(compute_side_length(L), compute_side_length(R))
```

In [14]:

## MODEL FORMULATION

### Sets

$S$ is the set of scrap types

$A$ is the set of layers to be balanced

### Parameters

- For each scrap type $s \in S$ we are given its layer, $P_s$.
- For each scrap type $s \in S$ we are given the width of the box containing the scrap, $w_s$.

**Variables**

For each scrap type $s \in S$:

- $y_s$ is a binary variable that equals $0$ if $s$ is in the left side of the yard and $1$ is it is on the right side of the yard

For each layer $a \in A$:

- $L_a$ is a continuous variable that represents the sum of the widths of the boxes of layer $a$ on the left side of the yard (including walls)
- $R_a$ is a continuous variable that represents the sum of the widths of the boxes of layer $a$ on the right side of the yard (including walls)
- $D_a$ is a continuous (and possibly negative) variable that represents the difference between $L_a$ and $R_a$

And also:

- $L$ is a continuous variable that represents the total length of the left side of the yard (walls and technical areas included)
- $R$ is a continuous variable that represents the total length of the right side of the yard (walls and technical areas included)
- $D$ is a continuous (and possibly negative) variable that represents the difference between $L$ and $R$
- $C$ is a continuous (and possibly negative) variable that represents the sum of the absolute values of $D_a$

**Objective function**

(1) The objective is to minimize the difference between the total length of the left and right sides of the yard. This can be stated as

$$\min |D|$$

(2a) The objective is, first, to minimize the difference between the total length of the left and right sides of the yard and, second, to minimize the difference between the total length on the left and on the right for the layers in A. This can be stated as

1. $\min |D|$
2. $\min \big(|D_1|+|D_4|+|D_5|+|D_6|+|D_7|\big)$

(2b) The objective is, first, to minimize the difference between the total length of the left and right sides of the yard for the layers in A and, second, to minimize the difference between the total length on the left and on the right. This can be stated as

1. $\min \big(|D_1|+|D_4|+|D_5|+|D_6|+|D_7|\big)$
2. $\min |D|$

(3) The objective is to minimize the difference between the total length of the left and right sides of the yard with weight $\alpha$ and the difference between the left and right side within boxes of layer 1, 4, 5, 6 and 7 with weight $1-\alpha$. This can be stated as

$$\min \big(\alpha |D| + (1-\alpha)(|D_1|+|D_4|+|D_5|+|D_6|+|D_7|)\big)$$

**Constraints**

- For each $a \in A$, the sum of the widths of the boxes of layer $a$ on the left side of the yard: $$L_a = 1.7 + \sum_{s \in S, \, P_s=a} (w_s + 1.7) (1-y_s)$$
- For each $a \in A$, the sum of the widths of the boxes of layer $a$ on the right side of the yard: $$R_a = 1.7 + \sum_{s \in S, \, P_s=a} (w_s + 1.7) y_s$$
- For each $a \in A$, the difference between the sum of the widths of the boxes of layer $a$ on the left and on the right side of the yard: $$D_a = L_a - R_a$$
- The sum of the absolute values:$$C = |D_1|+|D_4|+|D_5|+|D_6|+|D_7|$$
- The total length of the left side of the yard: $$L = 2\times 20 + 1.7 + \sum_{s \in S} (w_s + 1.7) (1-y_s)$$
- The total length of the right side of the yard: $$R = 2\times 20 + 1.7 + \sum_{s \in S} (w_s + 1.7) y_s$$
- The difference between the sum of the widths of the boxes on the left and on the right side of the yard: $$D = L - R$$

## MODEL IMPLEMENTATION

In [15]:
```
!pip install gurobipy
import gurobipy as gp
```

```
Requirement already satisfied: gurobipy in /usr/local/lib/python3.11/dist-packages (12.0.2)
```

In [16]:
```
from gurobipy import *
from gurobipy import GRB
```

In [17]:
```
# Create an environment with your WLS license
params = {
"WLSACCESSID": '********-****-****-****-************',
"WLSSECRET": '********-****-****-****-************',
"LICENSEID": *******,
}
```

```
env = gp.Env(params=params)

# Create the model within the Gurobi environment
model = gp.Model(env=env)
```

```
Set parameter WLSAccessID
Set parameter WLSSecret
Set parameter LicenseID to value 2619152
Academic license 2619152 - for non-commercial use only - registered to fi___@isel.pt
```

In [18]:
```
start_time = datetime.now()
```

**Sets**

In [19]:
```
S = range(32)
A = [1, 4, 5, 6, 7]
```

**Parameters**

Values for parameter $P_s$ (layer of scrap type $s \in S$):

In [20]:
```
scrap_layer = {s:scrap_layer_dict[s] for s in S}
#scrap_layer
```

Values for parameter $w_s$ (width of box containing scrap type $s \in S$):

In [21]:
```
box_width = {s:box_width_dict[s] for s in S}
#box_width
```

**Variables**

For each scrap type, create a variable `side` ($y_s$):

In [22]:
```
side = model.addVars(S, vtype=GRB.BINARY, name="side")
```

For each layer in $A$, create a variable `left` ($L_a$):

In [23]:
```
left = model.addVars(A, name="left")
```

For each layer in $A$, create a variable `right` ($R_a$):

In [24]:
```
right = model.addVars(A, name="right")
```

For each layer in $A$, create a variable `LRdiff` ($D_a$):

In [25]:
```
LRdiff = model.addVars(A, lb=-gp.GRB.INFINITY, name="LRdiff")
```

For each layer in $A$, create variable `absLRdiff` for the absolute value of difference of $D_a$:

In [26]:
```
absLRdiff = model.addVars(A, name="absLRdiff")
```

For the sum of the absolute value of the $D_a$ with $a \in A$, create variable `C` ($C$):

In [27]:
```
C = model.addVar(name="C")
```

For the total length of the left side of the yard, create variable `L` ($L$):

In [28]:
```
L = model.addVar(name="L")
```

For the total length of the right side of the yard, create variable `R` ($R$):

In [29]:
```
R = model.addVar(name="R")
```

For the difference between $L$ and $R$, create variable `D` ($D$):

In [30]:
```
D = model.addVar(lb=-gp.GRB.INFINITY, name="D")
```

For the absolute value of difference between $L$ and $R$, create variable `absD` :

In [31]:
```
absD = model.addVar(name="absD")
```

**Constraints**

In [32]:
```
for a in A:
  model.addConstr((1.7 + sum([(box_width[s] + 1.7)*(1-side[s]) for s in S if scrap_layer[s] == a]) == left[a]), name="L")
  model.addConstr((1.7 + sum([(box_width[s] + 1.7)*side[s] for s in S if scrap_layer[s] == a]) == right[a]), name="R")
  model.addConstr(LRdiff[a] == left[a] - right[a], name="LRdiff")
  model.addConstr(absLRdiff[a] == abs_(LRdiff[a]), name="absLRdiff");
```

In [33]:
```
model.addConstr(( sum(absLRdiff[a] for a in A) ) == C, name="C");
```

In [34]:
```
model.addConstr((20*2 + 1.7 + sum([(box_width[s] + 1.7)*(1-side[s]) for s in S]) == L), name="totallengthL")

model.addConstr((20*2 + 1.7 + sum([(box_width[s] + 1.7)*side[s] for s in S]) == R), name="totallengthR");
```

In [35]:
```
model.addConstr(D == L - R, name="D")
model.addConstr(absD == abs_(D), name="adsC");
```

**Objective function**

In [36]:
```
%%skip

# version (1)

model.setObjective(absD, GRB.MINIMIZE)
```

In [37]:
```
%%skip

# version (2a)

# Primary objective: |D|
model.setObjectiveN(absD, index=0, priority=1)
# Secondary objective: C
model.setObjectiveN(C, index=1, priority=0)
```

In [38]:
```
%%skip

# version (2b)

# Primary objective: C
model.setObjectiveN(C, index=0, priority=1)
# Secondary objective: |D|
model.setObjectiveN(absD, index=1, priority=0)
```

In [39]:
```
#%%skip

# version (3)

alpha = 0.5

model.setObjective(alpha*absD + (1-alpha)*C, GRB.MINIMIZE)
```

In [39]:
```

```

**Solve model**

In [40]:
```
# Optimize
model.optimize()

# do IIS if the model is infeasible
if model.Status == GRB.INFEASIBLE:
    model.computeIIS()
```

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (linux64 - "Ubuntu 22.04.4 LTS")

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license 2619152 - for non-commercial use only - registered to fi___@isel.pt
Optimize a model with 19 rows, 57 columns and 158 nonzeros
Model fingerprint: 0x0dd0e5c2
Model has 6 simple general constraints
  6 ABS
Variable types: 25 continuous, 32 integer (32 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+01]
  Objective range  [5e-01, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 9e+02]
Presolve removed 1 rows and 20 columns
Presolve time: 0.00s
Presolved: 18 rows, 37 columns, 74 nonzeros
Variable types: 0 continuous, 37 integer (21 binary)
Found heuristic solution: objective 797.2500000
Found heuristic solution: objective 779.5500000
Found heuristic solution: objective 773.8500000
Found heuristic solution: objective 765.0500000
Found heuristic solution: objective 747.3500000

Root relaxation: objective 8.000000e-01, 17 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.80000    0    6  747.35000    0.80000   100%     -    0s
H    0     0                      33.4500000    0.80000  97.6%     -    0s
     0     0    1.32204    0   12   33.45000    1.32204  96.0%     -    0s
     0     0    1.33385    0    8   33.45000    1.33385  96.0%     -    0s
H    0     0                      30.4500000    1.33385  95.6%     -    0s
     0     0    1.33385    0    8   30.45000    1.33385  95.6%     -    0s
     0     0    1.75514    0   14   30.45000    1.75514  94.2%     -    0s
H    0     0                      30.0500000    1.75542  94.2%     -    0s
     0     0    2.47028    0   11   30.05000    2.47028  91.8%     -    0s
     0     0    2.47028    0   13   30.05000    2.47028  91.8%     -    0s
     0     0    2.71688    0   16   30.05000    2.71688  91.0%     -    0s
     0     0    2.79289    0   12   30.05000    2.79289  90.7%     -    0s
     0     0    2.79289    0   14   30.05000    2.79289  90.7%     -    0s
```

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 2.79289 | 0 | 16 | 30.05000 | 2.79289 | 90.7% | - | 0s |
| | 0 | 0 | 2.79289 | 0 | 18 | 30.05000 | 2.79289 | 90.7% | - | 0s |
| | 0 | 0 | 2.79853 | 0 | 18 | 30.05000 | 2.79853 | 90.7% | - | 0s |
| | 0 | 0 | 3.26633 | 0 | 20 | 30.05000 | 3.26633 | 89.1% | - | 0s |
| H | 0 | 0 | | | | 29.1500000 | 3.38646 | 88.4% | - | 0s |
| H | 0 | 0 | | | | 24.3500000 | 3.38646 | 86.1% | - | 0s |
| | 0 | 0 | 3.38646 | 0 | 23 | 24.35000 | 3.38646 | 86.1% | - | 0s |
| | 0 | 0 | 3.38646 | 0 | 20 | 24.35000 | 3.38646 | 86.1% | - | 0s |
| | 0 | 0 | 3.38646 | 0 | 21 | 24.35000 | 3.38646 | 86.1% | - | 0s |
| | 0 | 0 | 3.45484 | 0 | 18 | 24.35000 | 3.45484 | 85.8% | - | 0s |
| | 0 | 0 | 3.46052 | 0 | 15 | 24.35000 | 3.46052 | 85.8% | - | 0s |
| H | 0 | 0 | | | | 11.1500000 | 3.46052 | 69.0% | - | 0s |
| | 0 | 0 | 3.54561 | 0 | 19 | 11.15000 | 3.54561 | 68.2% | - | 0s |
| | 0 | 0 | 3.59052 | 0 | 19 | 11.15000 | 3.59052 | 67.8% | - | 0s |
| | 0 | 0 | 3.59052 | 0 | 20 | 11.15000 | 3.59052 | 67.8% | - | 0s |
| | 0 | 0 | 3.76817 | 0 | 20 | 11.15000 | 3.76817 | 66.2% | - | 0s |
| H | 0 | 0 | | | | 8.6500000 | 3.76817 | 56.4% | - | 0s |
| | 0 | 0 | 3.82106 | 0 | 22 | 8.65000 | 3.82106 | 55.8% | - | 0s |
| | 0 | 0 | 3.82350 | 0 | 22 | 8.65000 | 3.82350 | 55.8% | - | 0s |
| | 0 | 0 | 3.82570 | 0 | 24 | 8.65000 | 3.82570 | 55.8% | - | 0s |
| | 0 | 0 | 3.82570 | 0 | 25 | 8.65000 | 3.82570 | 55.8% | - | 0s |
| | 0 | 0 | 3.82932 | 0 | 24 | 8.65000 | 3.82932 | 55.7% | - | 0s |
| | 0 | 0 | 3.83729 | 0 | 25 | 8.65000 | 3.83729 | 55.6% | - | 0s |
| | 0 | 0 | 3.86090 | 0 | 24 | 8.65000 | 3.86090 | 55.4% | - | 0s |
| | 0 | 0 | 3.86147 | 0 | 25 | 8.65000 | 3.86147 | 55.4% | - | 0s |
| | 0 | 0 | 3.87567 | 0 | 23 | 8.65000 | 3.87567 | 55.2% | - | 0s |
| | 0 | 0 | 3.90636 | 0 | 23 | 8.65000 | 3.90636 | 54.8% | - | 0s |
| | 0 | 0 | 3.90689 | 0 | 23 | 8.65000 | 3.90689 | 54.8% | - | 0s |
| | 0 | 0 | 3.97723 | 0 | 24 | 8.65000 | 3.97723 | 54.0% | - | 0s |
| | 0 | 0 | 3.98740 | 0 | 23 | 8.65000 | 3.98740 | 53.9% | - | 0s |
| | 0 | 0 | 3.98860 | 0 | 24 | 8.65000 | 3.98860 | 53.9% | - | 0s |
| | 0 | 0 | 3.98860 | 0 | 24 | 8.65000 | 3.98860 | 53.9% | - | 0s |
| | 0 | 0 | 4.04313 | 0 | 21 | 8.65000 | 4.04313 | 53.3% | - | 0s |
| H | 0 | 0 | | | | 8.2500000 | 4.04313 | 51.0% | - | 0s |
| | 0 | 0 | 4.04542 | 0 | 24 | 8.25000 | 4.04542 | 51.0% | - | 0s |
| | 0 | 0 | 4.04546 | 0 | 22 | 8.25000 | 4.04546 | 51.0% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 23 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 23 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 20 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 23 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 22 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 24 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 19 | 8.25000 | 4.05581 | 50.8% | - | 0s |
| | 0 | 0 | 4.05581 | 0 | 19 | 8.25000 | 4.05581 | 50.8% | - | 0s |

```
     0     1    4.11142    0   19    8.25000    4.11142  50.2%     -    0s
H   11     7                        6.6500000    4.95000  25.6%   1.3    0s

Cutting planes:
  Gomory: 2
  Cover: 17
  MIR: 47
  StrongCG: 11
  Inf proof: 2
  Zero half: 1
  Mod-K: 2
  Relax-and-lift: 2

Explored 230 nodes (1372 simplex iterations) in 0.41 seconds (0.02 work units)
Thread count was 2 (of 2 available processors)

Solution count 10: 6.65 8.25 8.65 ... 747.35

Optimal solution found (tolerance 1.00e-04)
Best objective 6.650000000000e+00, best bound 6.650000000000e+00, gap 0.0000%
```

In [41]:
```
end_time = datetime.now()
runtime = (end_time - start_time).total_seconds()
print('Runtime (sec): {}'.format(runtime))
```

```
Runtime (sec): 0.673448
```

In [42]:
```
print('\nModel output for Split 3 0.5')
```

```
Model output for Split 3 0.5
```

In [43]:
```
# Print the values of all variables
for v in model.getVars():
    print(f"{v.VarName} = {v.X}")
```

```
side[0] = -0.0
side[1] = 1.0
side[2] = -0.0
side[3] = 1.0
side[4] = 1.0
side[5] = 1.0
side[6] = -0.0
side[7] = 1.0
side[8] = -0.0
side[9] = 0.0
side[10] = 0.0
side[11] = 0.0
side[12] = -0.0
side[13] = 1.0
side[14] = -0.0
side[15] = 0.0
side[16] = 1.0
side[17] = 1.0
side[18] = 1.0
side[19] = 1.0
side[20] = 0.0
side[21] = 1.0
side[22] = 1.0
side[23] = 0.0
side[24] = -0.0
side[25] = 1.0
side[26] = 1.0
side[27] = 1.0
side[28] = -0.0
side[29] = 1.0
side[30] = 1.0
side[31] = 0.0
left[1] = 114.6
left[4] = 65.7
left[5] = 47.0
left[6] = 77.5
left[7] = 94.8
right[1] = 117.0
right[4] = 64.5
right[5] = 45.4
right[6] = 71.0
right[7] = 96.2
LRdiff[1] = -2.4
LRdiff[4] = 1.2
```

```
LRdiff[5] = 1.6
LRdiff[6] = 6.5
LRdiff[7] = -1.4
absLRdiff[1] = 2.4
absLRdiff[4] = 1.2
absLRdiff[5] = 1.6
absLRdiff[6] = 6.5
absLRdiff[7] = 1.4
C = 13.1
L = 450.5
R = 450.7
D = -0.2
absD = 0.2
```

In [44]:

```
left_side, right_side = get_split(S)
print(f'Scraps on left side: {left_side}')
print(f'Scraps on right side: {right_side}')
```

```
Scraps on left side: [0, 2, 6, 8, 9, 10, 11, 12, 14, 15, 20, 23, 24, 28, 31]
Scraps on right side: [1, 3, 4, 5, 7, 13, 16, 17, 18, 19, 21, 22, 25, 26, 27, 29, 30]
```

In [45]:

```
print(f'Length of left side: {round(compute_side_length(left_side), 2)}')
print(f'Length of right side: {round(compute_side_length(right_side), 2)}')
```

```
Length of left side: 450.5
Length of right side: 450.7
```

In [46]:

```
print(f'Layer unevenness: {C.X}')
```

```
Layer unevenness: 13.1
```

In [46]:

In [46]: