Materials List: Pipes

Annie Ding

Spring 2013

Part I
Problem Definition

Introduction

For the semester of Spring 2013, I will be working on the Materials List code for the entire plant design. There had already been progress made on this task by team member Kira Gidron in Fall 2012. Her report thoroughly explains the details of how she put together all the information needed in order to write the code. Once the entire design challenge has been completed, the materials list code should integrate what we know about the materials available in the region (so far, we only have information about materials in Honduras) and be able to tell the person who requested the code about how much material they will need to build the entire plant, including but not limited to the following hydraulic and structural components:

- Hydraulic design materials
  - Pipes
  - Pipe fittings
  - Flocculator baffle sheets
  - Sedimentation plate settlers
  - Chemical dose controller pieces
- Structural design materials
  - Brick
  - Rebar
  - Cement (for plant walls)
  - River rock (for entrance tank hoppers)

This will primarily be useful to construction teams in the Honduras and for estimating how much a specific plant design would cost. The code should eventually automatically output a list of materials needed to build any plant specified by the user and should include number of items needed, costs and should be in both Spanish and English.

The code that will eventually output the list of pipes had been started by a previous design team member. There were originally two proposed methods:

1. It turns out that this problem is a typical cutting stock problem. The cutting stock algorithm for optimizing the number of lances, which Kira had begun to write, was first proposed by Andrew Hart.

2. Kira also proposed a second method (detailed in her report) for calculating the number of lances needed that involved finding permutations of all the different pipe lengths, but after talking to her I have decided that is it probably not the best use of time to try to implement that method in addition to cutting stock method.

Design Details

Lances Calculation

The first part of the code that was tackled was the calculations for the number and sizes of pipes needed. PVC pipes in Honduras are available as 6.1 meter sections called lances. The completed code, which will be in a new MathCAD file that will be called after the entire plant has been drawn, should find the minimum number of lances needed to build the plant. The first part of the code will find and categorize all the pipes based on their nominal diameter and pipe specification in a series of matrices. This part of the code has been finished. The second part of the code will then use this information to optimize the number of lances needed to reduce waste and therefore cost. This part of the code is still incomplete. After optimization code has been completed, this information needs to be output so that it can be easily read and interpreted by anyone requesting a plant design; the best way to do this will most likely be to output all the variables into a Microsoft Excel file, similar to how the plant design specifications are output into a Microsoft Word file. This Excel file would then be included as an attachment in the same email to the person requesting the design as the AutoCAD file.

Relationship to Existing Code

The code to find the matrices which will store pipe information depends on the current PipeF and PipeSUBF functions, which draw pipes in AutoCad when called. In addition to the original inputs of the drawing functions (OriginPoint, Length, AngleVector, Nominal Diameter (ND) and enumerated pipe schedule type or pipe specification (EN)), the code will also require additional inputs of

N and will output a new variable called PipeMatrix. N is the number of pipes drawn with that particular call of the drawing function; namely, N will depend on how many times a pipe will be arrayed or mirrored. Since N will be based entirely upon the number of times that pipe is arrayed or mirrored, the input N for a specific pipe drawing function (PipeF or PipeSUBF) will be based on this number, which has already been calculated in the code elsewhere. PipeMatrix is a variable that has been created to hold all this information; PipeMatrix will have one new row of values for every time a pipe drawing function is called in the code, each row containing four values: Length, ND, EN and N. To implement this new code, the entire AutoCAD drawing code will have to be searched to find and replace each call of a pipe drawing function with the updated code.

Constraints and Assumptions

Several assumptions were made before starting to write the code to calculate the optimal number of lances, most of which were discussed first with Andrew Hart. First, we are assuming that the construction crews may make mistakes in cutting the pipe and that there will be differences between the design and the actual construction lengths in real life. Because of this, we are building in a buffer of about 60 cm into the code so that our amount and price estimates will take this into account. This will mean that our actual usable pipe length will be $6.1 - .6 = 5.5$m.

Significant figures needed for the pipe lengths should be given in millimeters. Although actual construction crews in the Honduras do not build most parts of the plant with millimeter precision, lengths should be given in millimeters anyway since a few parts of the plant, like the spacer pipes in the plate modules, need to be built with millimeter precision in mind.

Couplings between pipes do not need to be taken into account (for pipes that will need to be longer than the lances) since the lances include a "campana," which allows lances to slide into one another without need for a coupling. However, the buffer will be accounting for the fact that the "campanas" can be of differing lengths.

Solution Approach

Kira has already provided a start to the code for most parts of the lance calculation task. First, she wrote code for making the PipeMatrix within the pipe drawing functions.

1. The code:
   (a) Construct PipeMatrix (see Relationship to Existing Code section)
   (b) There will be lengths of pipe in the PipeMatrix that will be longer than the length of a lance; separate these lengths into a new matrix (LongPipesMatrix). This matrix will be put through all the same dividing codes as the original PipeMatrix (1(c)-(d)).

(c) Separate the PipeMatrix into different matrices that each only contain pipes of a certain nominal diameter and pipe specification.

(d) Group together all pipes with the same length in the same row, adding the appropriate number onto the value of N, the number of times the drawn pipe is arrayed or mirrored.

(e) Optimizing cutting of lances:

i. For each of the rows (pipes) in LongPipesMatrix:

A. Determine the number of full lances needed to construct each pipe in an array (A)

B. Find the length of pipe leftover (t) for each pipe and replace the previous lengths in the arrays with the leftover lengths. Use this edited matrix (e)ii.B as you would use PipeMatrix.

C. Add the number of lances (n) used to a new matrix, LanceMatrix, with the corresponding pipe specification and nominal diameter. LanceMatrix will be used later in the calculation of the cost of all the pipes.

ii. For each of the rows (pipes) in PipeMatrix:

A. Find the longest pipe length (p) for the an array (A) in PipesMatrix.

B. Find $t = l - p$, where l is the length of a lance.

C. Make a new matrix called CutPipesMatrix (just the first time; the same matrix will be used for all following pipes p in all arrays A) to hold the lengths of pipes subtracted from this lance. Each row of CutPipesMatrix should stand for one specific pipe and it will look like LanceMatrix, except instead of n there will be a column that will hold a vector (per pipe) stating the lengths that will be cut out of that lance.

D. Add the length p onto the vector of the appropriate row in CutPipesMatrix.

E. Add 1 onto n in the row in LanceMatrix corresponding to the pipe specification and nominal diameter of A (if there is no existing row, stack a new row with the proper information onto the matrix).

iii. From the remaining lengths in A, find the next longest length that is shorter than the remaining lance t:

A. Loop through all the lengths in A to find the maximum of the lengths (m) that is still less than t.

B. Subtract this number m from the current t to find the new t.

C. Repeat i. through iii. until t is less than the smallest length available in A.

iv. Subtract 1 from the N associated with that particular pipe length in A to make sure we don't accidentally use lances for pipes already accounted for.

v. Repeat (i) though (iv) until all arrays are accounted for. This process is shown in Figure 1.

(f) The cost of each of the lances can be found using costs from previous plants built in Honduras, then the number of lances needed for each ND and EN can be multiplied by these numbers to find the total cost for all the pipes in the plant.

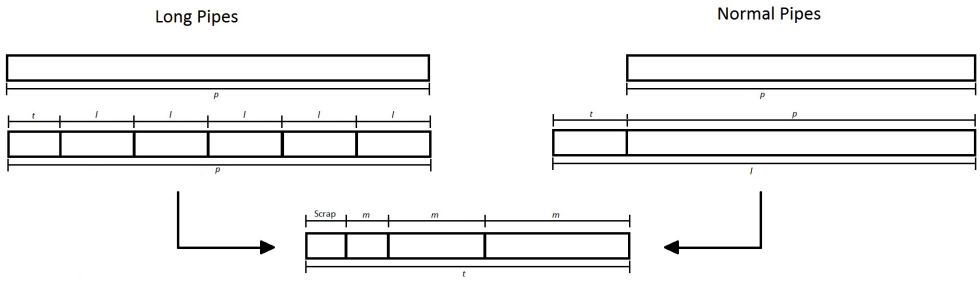

Figure 1:

2. Testing the code:

(a) After the coding has been completed, we will need to test the code by hand to make sure it does what it's supposed to do. A small matrix, with about 20 rows, will be used to test the matrix and the appropriate number of lances needed will be calculated by hand as well in order to see if the code is correctly optimizing the number of lances needed.

(b) We can then test the code on a plant design of the same flow rate as the San Nicolas plant, which we have the pipe data for. If the numbers for the number of pipes (and therefore the cost) don't match up, assumptions may have to be reassessed and the code will have to be thoroughly re-examined.

Part II
Documented Progress

Calculating the Lances

Figure 2 shows the calculations made for finding the length of useable lance. Instead of hardcoding the length of the buffer into the equation for the useable

length, a new variable was made to hold the buffer length, just in case we need to change this variable in the future (for example, if the calculations turn out to be different than hand-calculated numbers). The names of the variables have also been changed to better reflect variable naming conventions. These are fixed variables and will be incorporated into the ExpertInputs MathCAD file.

$$L_{Lance} := 6.1m$$

$$L_{LanceBuffer} := 0.60m$$

$$L_{UseableLance} := L_{Lance} - L_{LanceBuffer}$$

Figure 2:

Long pipe matrix

Figure 3 shows the calculations needed to remove the pipes that are longer than the length of the useable lance. The only changes to this code (apart from variable name changes) is the initialization of the long pipes matrix within the code for the LongPipes function and the consequential stacking of succeeding rows onto the initialized matrix. After testing the code, we found that the output of the LongPipesMatrix ended up being matrices within matrices instead of one table viewable as such in MathCAD. This problem was fixed with this change in the code. The initialized row of four zeros will not affect following code, as it will just register as a null pipe, with no length and no number N.

```
LongPipes(PipeMatrix) :=
    n ← 0
    LongPipesMatrix ← (0  0  0  0)
    for i ∈ 0 .. rows(PipeMatrix) − 1
      if (PipeMatrix^⟨0⟩)_i > L_UseableLance
        LongPipesMatrix ← stack[LongPipesMatrix, (PipeMatrix^T)^⟨i⟩^T]
        (PipeMatrix^⟨0⟩)_i ← 0
        n ← n + 1
    return ( RemoveInitializingZeroRow(LongPipesMatrix)
             RemoveZeroLengths(PipeMatrix) )
```

Figure 3:

Dividing Matrix by Nominal Diameter

The code for separating the pipe matrix (Figure 4) goes through every row of the matrix input, adding rows with the same ND to a previously initialized matrix (called TempMatrix in the code). If the ND values are different, the code will add the temporary matrix that it had been building into a new row of a different matrix, called NDDividedMatrix, which will eventually store all the matrices (each with a different ND value) within it, each to a different row (Figure 5). Note that when viewing the overall matrix, you will not be able to see the data directly; you must view the row of the matrix in order to view the numbers matrix it holds. The format {1,4} is saying that there is a 1×4 matrix in that row.

```
DivideMatrixND(PipeMatrix) :=
    n ← 0
    TempMatrix ← (0 0 0 0)
    MatrixUse ← csort(PipeMatrix, 1)
    for i ∈ 1.. rows(MatrixUse) − 1
        if i = 1
            TempMatrix ← stack[TempMatrix, (MatrixUse^T)^⟨i−1⟩^T]
            TempMatrix ← stack[TempMatrix, (MatrixUse^T)^⟨i−1⟩^T] if (MatrixUse^⟨1⟩)_i = (MatrixUse^⟨1⟩)_i−1
            otherwise
                NDDividedMatrix_n ← RemoveInitializingZeroRow(TempMatrix)
                n ← n + 1
                TempMatrix ← (0 0 0 0)
                TempMatrix ← stack[TempMatrix, (MatrixUse^T)^⟨i⟩^T]
        if (MatrixUse^⟨1⟩)_i = (MatrixUse^⟨1⟩)_i−1
            TempMatrix ← stack[TempMatrix, (MatrixUse^T)^⟨i⟩^T]
            NDDividedMatrix_n ← RemoveInitializingZeroRow(TempMatrix) if i = rows(MatrixUse) − 1
        otherwise
            NDDividedMatrix_n ← RemoveInitializingZeroRow(TempMatrix)
            if i ≠ rows(MatrixUse) − 1
                n ← n + 1
                TempMatrix ← (0 0 0 0)
                TempMatrix ← stack[TempMatrix, (MatrixUse^T)^⟨i⟩^T]
    return NDDividedMatrix
```

Figure 4:

$$\text{DividedNDMatrix} := \text{DivideMatrixND}(\text{NDSortedPipeMatrix}) = \begin{pmatrix} \{3,4\} \\ \{1,4\} \\ \{4,4\} \end{pmatrix} \text{m}$$

$$\text{DividedNDMatrix}_0 = \begin{pmatrix} 4 & 1 & 5 & 4 \\ 3 & 1 & 3 & 24 \\ 1 & 1 & 1 & 8 \end{pmatrix} \text{m}$$

$$\text{DividedNDMatrix}_1 = \begin{pmatrix} 4 & 2 & 2 & 3 \end{pmatrix} \text{m}$$

$$\text{DividedNDMatrix}_2 = \begin{pmatrix} 5.2 & 5 & 4 & 3 \\ 1 & 5 & 2 & 35 \\ 2 & 5 & 5 & 6 \\ 1 & 5 & 4 & 13 \end{pmatrix} \text{m}$$

Figure 5:

Dividing Matrices by EN

Sorting each matrix within the overall matrix (DividedNDMatrix in Figure 5) will use the same function as used in the code in 4 but will be a little more involved because each matrix is contained within DividedNDMatrix. The code that will go into the matrix and sort each smaller matrix is shown in Figure 6.

```
ENSortMatrix(DividedNDMatrix) :=
    for i ∈ 0 .. rows(DividedNDMatrix) − 1
        TempMatrix ← DividedNDMatrix_i
        SortedTempMatrix ← csort(TempMatrix, 2)
        SortedDividedNDMatrix_i ← SortedTempMatrix
    return SortedDividedNDMatrix
```

Figure 6:

Dividing each matrix that was just sorted into even smaller matrices (Figure 7) will use code similar to that used in Figure 4. The output will be a matrix that contains smaller matrices, each of which will only contain rows with the same ND and EN values.

Figure 7:

Combining pipes of the same length

The code for combining rows (pipes) with the same length, ND and EN became overly complicated because I was originally planning on using it at the beginning of the solution approach (between 1(a) and 1(b)), and would therefore had to go through each matrix all the way to see if there were any pipes with the same values. However, in the end the code (Figure 8) only recognizes similar rows if they are adjacent to each other. This is due to the third to last row in the code that assigns a zero to the length of a pipe, but removing it would cause even worse problems. I could find no way to fix the issue and leave the current code largely intact. This section of code will be further discussed in the Future Work section.

Figure 8:

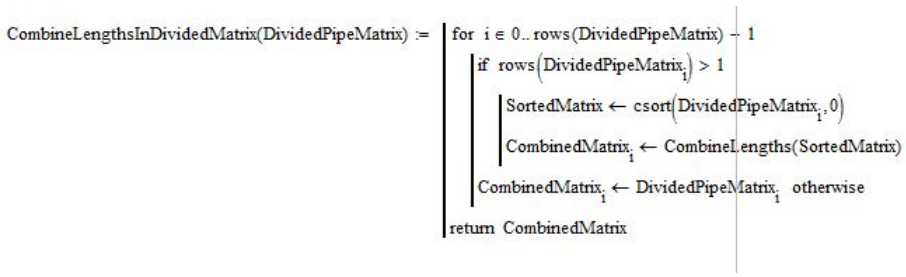

Figure 9:

Making LanceMatrix and LeftoverMatrix with Long Pipes

The divided LongPipesMatrix will be split into two new matrices by the code shown in Figure 10; LanceMatrix and LeftoverMatrix. LanceMatrix contains three columns but otherwise will have all the same dimensions as the original LongPipesDividedMatrix that is the input. The three columns will contain the information for number of lances, ND and EN (as detailed in the solution approach). The number of lances is the full number of 5.5m lengths that fit into

the overall length needed multiplied by the number of pipes needed in the plant, N. The LeftoverMatrix will contain the "leftover" amount of pipe left over after enough full lances are used, as well as the ND, EN and N (which will remain the same as in LongPipesMatrix). If there is no leftover pipe, that row of the matrix is not represented in LeftoverMatrix. LeftoverMatrix will be treated the same way as the divided PipesMatrix when it comes to cutting optimization. The function CombinePipeNumber, used in the return statement, combines the number of pipes with the same ND and EN into one row.

```
CutLongPipes(LongPipesDividedMatrix) :=
    n ← 0
    LanceMatrix ← (0  0  0)
    for i ∈ 0 .. rows(LongPipesDividedMatrix) − 1
        MatrixUse ← LongPipesDividedMatrix_i
        LanceTempMatrix ← (0  0  0)
        CountZeros ← 0
        for j ∈ 0 .. rows(MatrixUse) − 1
            NumberOfLances ← floor( (MatrixUse^⟨0⟩)_j / L_UseableLance )
            LeftoverLength ← (MatrixUse^⟨0⟩)_j − NumberOfLances · L_UseableLance
            TempMatrix ← (0  0  0)
            TempMatrix^⟨0⟩ ← NumberOfLances · (MatrixUse^⟨3⟩)_j
            TempMatrix^⟨1⟩ ← (MatrixUse^⟨1⟩)_j
            TempMatrix^⟨2⟩ ← (MatrixUse^⟨2⟩)_j
            LanceTempMatrix ← stack(LanceTempMatrix, TempMatrix)
            (MatrixUse^⟨0⟩)_j ← LeftoverLength
            CountZeros ← 1  if  LeftoverLength = 0
        if  CountZeros = 1 ∧ rows(MatrixUse) > 1
            LeftoverMatrix_n ← RemoveZeroLengths(MatrixUse)
            n ← n + 1
        if  CountZeros = 0
            LeftoverMatrix_n ← MatrixUse
            n ← n + 1
        LanceMatrix ← stack(LanceMatrix, RemoveInitializingZeroRow(LanceTempMatrix))
    return ( LeftoverMatrix
             RemoveInitializingZeroRow(CombinePipeNumber(LanceMatrix)) )
```

Figure 10:

Cutting Optimization

The actual cutting stock algorithm is implemented in the code shown in Figure 11. The code will take a divided matrix and an existing LanceMatrix (created in Figure 10) and cut each pipe in the divided matrix from the length of a useable lance as described in the solution approach. There are two subfunctions included in the code: RemoveZeroN will take a LanceMatrix as an input and remove any rows with a zero in the N column and AddToLanceMatrix will also take LanceMatrix as an input and add a specified number n to number N in the row with the appropriate input ND and EN. So far, the code does not completely work. The first case (the first if statement, for when the small matrix the code is working with is only one row long) does work as intended, but the other cases do not currently work. I'm not completely certain where the code is going wrong, but it may have to do with the fact that sometimes MathCAD would display unit errors; the code could have errors due to mismatches in variables with units and without that are not apparent with just looking.

```
CuttingOptimization(DividedMatrix, LanceMatrix) :=
    CutMatrix ← (0 0 0)
    for i ∈ 0..rows(DividedMatrix) − 1
        TempCutLengths ← (0)
        TempCutMatrix ← (0 0 0)
        MatrixUse ← DividedMatrix_i
        LanceUse ← L_UseableLance
        for j ∈ 0..rows(MatrixUse) − 1
            LongestLength ← max(MatrixUse^⟨0⟩)
            if rows(MatrixUse) − 1 = 0
                TempCutLengths ← stack(TempCutLengths, MatrixUse^⟨0⟩)
                TempCutMatrix^⟨1⟩ ← (MatrixUse^⟨1⟩)_j
                TempCutMatrix^⟨2⟩ ← (MatrixUse^⟨2⟩)_j
                CutMatrix ← stack(CutMatrix, TempCutMatrix)
                (CutMatrix^⟨0⟩)_rows(CutMatrix)−1 ← RemoveInitializingZeroRow(TempCutLengths)
            if LongestLength ≤ LanceUse
                LanceUse ← LanceUse − LongestLength
                TempCutLengths ← stack(TempCutLengths, LongestLength)
                for k ∈ 0..rows(MatrixUse) − 1
                    (MatrixUse^⟨3⟩)_k ← (MatrixUse^⟨3⟩)_k − 1m  if LongestLength = (MatrixUse^⟨0⟩)_k
                    MatrixUse ← RemoveZeroN(MatrixUse)  if (MatrixUse^⟨3⟩)_k = 0
            otherwise
                TempCutMatrix^⟨1⟩ ← (MatrixUse^⟨1⟩)_j
                TempCutMatrix^⟨2⟩ ← (MatrixUse^⟨2⟩)_j
                CutMatrix ← stack(CutMatrix, TempCutMatrix)
                (CutMatrix^⟨0⟩)_rows(CutMatrix)−1 ← RemoveInitializingZeroRow(TempCutLengths)
                TempCutLengths ← (0)
                TempCutMatrix ← (0 0 0)
                LanceMatrix ← AddToLanceMatrix[1, (MatrixUse^⟨1⟩)_j / 1m, (MatrixUse^⟨2⟩)_j / 1m, LanceMatrix]
    return (RemoveInitializingZeroRow(CutMatrix)
            LanceMatrix)
```

Figure 11:

Part III
Future Work

To finish this design challenge, first the section regarding pipes needs to be completed before we can analyze other materials. The code for cutting optimization must first be completed. The code is already mostly completed, but the second if statement and the otherwise statement need to be debugged. Then, we need combine the LeftoverMatrix and the divided matrix with short pipes into one, unified divided matrix, which will be the matrix to go into the cutting optimization code. We then need to take the costs of lances for each kind of lance and use the numbers in LanceMatrix to find the total cost of pipes in a plant. Then, we need to test the code by hand as described in the solutions approach.

If the code works as expected, we can start implementing it into the existing plant drawing code. PipeMatrix needs to be built as an output of every drawing function in the code that requires pipes. Then the code can be tested for a specified plant size that is equivalent to an existing plant in Honduras to verify if the cost that is calculated is comparable to the actual cost of building the plant.