Final Design Report: Materials List

Kira Gidron

Fall 2012

Part I
Problem Definition

Introduction

During the semester of Fall 2012 I worked on the materials list for the AguaClara plant. The materials list includes both hydraulic design materials and structural materials. Hydraulic design materials include:

- pipes
- pipe fittings
- sheets for flocculator baffles
- sedimentation plate settlers
- chemical dose controller pieces

The structural materials include but are not limited to:

- brick
- rebar
- cement for plant walls
- river rock for entrance tank hoppers

A primary focus of the materials list is to make it useful for the construction teams in Honduras. This is exlained in more detail in Section 2. The materials list should also include pricing so the final materials list can be used as a materials cost estimate for new plants. A final objective of the materials list is adequately is to use it to help reduce the cost of a plant. This is explained in more detail in Section 6.

APP engineers have provided us with two materials lists. One of the materials lists compiled the hydraulic materials for the San Nicolas plant. It has

the list in Spanish and English. It is divided into components: entrance tank, flocculator, sedimentation tank, stacked rapid sand filters and doser. This materials list does not calculate the total number of units of different materials. The second one is a budget for all the materials (not only hydraulic) as well as expenses for labor, travel, training, promotion and more of the Alauca plant. It is also in English and Spanish. This list does calculate the total number of units of different materials. It does not separate the materials by components of the plant. The new materials list can use these material list examples to obtain information about materials, prices, structure, etc. APP has materials list for each of the constructed plants. AguaClara members working on this task can ask for more lists if need be.

Similarly to the two existing lists, the new materials list will present the materials plant componen, it will be in Spanish and English and will include prices and costs. It will not include anything that are not materials such as expenses for labor, etc. However, there is one main difference between the exisiting lists and the new list. The existing lists were made by hand specifically for the plants that were going to be constructed. On the other hand, the new materials list will be generated automatically and simultaneously with the plant design. In other words, the objective is to write code that will automatically create a materials list specific to each design request. This means that the materials list is dynamic and will have a feedback mechanism such that changes to the design of the plant will result in changes in the list.

1 Lances Calculation

Description

The first part of the process to create the materials list is to calculate the number of pipes needed for the construction of the plant. Specifically, I worked on calculating the number of lances needed to build all the pipes in the plant. In Honduras PVC pipes are available in 6.1 meter lengths which are are called lances. The task is to find the lowest number of lances that are needed for piping. The first part of the code consists in finding all the pipe lengths in the plant and categorizing them by nominal diameter and pipe specification. These lengths are stored in a matrix. The second part of the code then optimizes the number of lances needed in order to reduce waste.

Solution Approach

My first step was to create the pipe length matrix. I needed to decide if I would do this manually or automate it. I decided the best approach would be to do it automatically because it speeds up the process, ensures the calculations are accurate and allows for easy repetition each time a new design is requested. The matrix has 4 columns: pipe length, nominal diameter of the pipe, pipe specification and number of pipes. The fourth column referes to the number

of pipes with the same length, nominal diameter and pipe specification. The matrix has as many rows as there are different types of pipes (where at least 1 of the first 3 columns are different). The sum of the fourth column is the total number of pipes in the plant. The code for building the pipe length matrix is written into the PipeF and SUBF drawing functions so that when a new pipe is drawn, its length, ND and specification are inserted into the pipe length matrix.

My second step was to write algorithms that use the pipe length matrix to calculate the number of lances that are needed. There are two possible algorithms.

Method 1

The first method, proposed by Drew Hart, works as follows:

1. find the longest length in the array
2. calculated the unused length of the lance t (where t is the length of the lance L minus the longest length in the array)
3. from the remaining lengths in the array find the next longest length that is under t
   (a) loop through all the lengths
   (b) find all the lengths that are under t
   (c) find the maximum of these lengths
4. repeat steps 2-3 until t is less than a predetermined arbitrary value, x (scrap) such that t<x

This is the code that I have begun to write.

Method 2

The second way, which I propose, works as follows:

1. find all the possible non-repeating permutations of the lengths in the array
2. add up the lengths for each of the permutations until you reach y (length of a lance, L, minus the predetermined buffer value, x)
3. find the lowest sum and use those lengths
4. add a lance
5. start again with the remaining lengths until all the lengths in the array are accounted for

I have not started writing the code for this second method. After the code for the first method is written, the next member working on this task can decide if he/she thinks it is necessary or not to write the code for this second method. I encourage writing the code for both methods for the following reasons:

- I cannot foresee right now which method will be more time-intensive (time for code to process)
- I can check one with the other since ideally they should come up with a similar number of lances
- As I go through the programming for each I may find that one is better than the other, or that one does not work, or that a combination of both is the best method (I have found, in the past, that a general idea I have for an algorithm may flop while I am writing the code- I may realize that it is not possible to write the code or it is overcomplicated). The following explains how I will evaluate the two algorithms:
  - a principal parameter to decide which method is better is the accuracy of output (number of lances)
  - I can test the accuracy of the methods against the San Nicolas plant (for which we have number of lances that were used)
  - this is better explained in the Evaluation section

Constraints and Assumptions

Buffer refers to the amount of pipe that we want to leave untouched in a lance just in case an error is made during the cutting process. Drew thinks I should use a buffer of about 60 cm since "pipes won't always be cut to the exact length the design tool calculates - differences in fitting dimensions and construction error will be significant and in many cases we'll measure how long it needs to be in real life. Therefore we need some extra pipe to play with". They are hoping to have a table saw on site for future projects (rather than hand saws) which should increase precision but we will still use a buffer of 60 cm. This means that the amount of usable lance is 6.1 m - 60 cm which is 5.5 m. Therefore, in the code for lance calculations, we use only 5.5 m of the lance.

I asked Drew about the significant figures needed for the pipe lengths in the cutlist. Drew would like that I give the length in millimeters. He said they currently do not have millimeter construction precision in most cases but that I should include them anyway since for a few places in the plant (like spacer pipes in the plate modules) the engineers need the measurements in millimeters.

I thought that we would need to account for couplings for pipes that are longer than 6m (which is the case of inlet manifolds and exit launders in long sedimentation tanks). However, Drew informed me that the lances include a "campana" where the next pipe slides in so a coupling is not needed. However, the "campanas" are of different lengths so this should be included in the safety factor for those specific pipes.

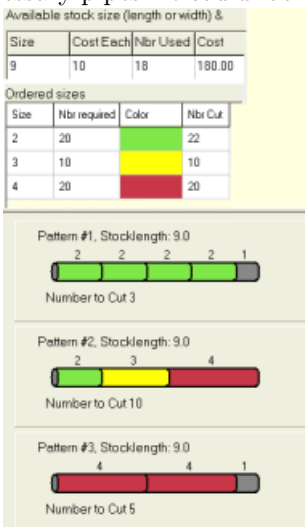
Evaluation

The next step after finishing writing the code will be to test it. First it can be tested with a testing matrix - a matrix with a small number of rows with

example pipes (about 20 pipes in total). The number of lances needed can be calculated by hand and compared to the results generated by the code. Secondly, the code can be tested with the San Nicolas plant. The materials list from the San Nicolas plant will be very useful here. We can run the code for a plant with the same flow rate as the San Nicolas plant. We will want to see if the code is giving values (of number of lances) similar to the number that was used (which can be found from the San Nicolas materials list). It may be necessary to confirm those numbers with APP as well. We will also want to see if one of the two methods is outputting more accurate values than the other. We will want to check if the code is underestimating or overestimating the number of lances and change the code accordingly. If the code is outputting highly inaccurate values we will want to find the causes for this. For example, it could be that the chosen buffer was not large enough since during the construction there were many more cutting errors than originally imagined (when deciding the buffer).

2 Outputting the Cut List

The next step in putting together the materials list is to figure out how to output the information generated by the lances calculation. After finding the best way to find the number of lances needed, we will need to find the best way to output this information for APP in Honduras. Aside from needing to know the number of lances, they need to know how to cut up the lances to create the necessary pipes. It could look something like this:



This would allow the engineers in Honduras to know exactly where to cut the pipes. The output also needs to output which cut up pieces will be used for what part of the plant. In other words, the output needs to detail how the cut up pieces are put together for which pipes and where in the plant. It is important to choose the right program to output the information. We need

to evaluate which programs work best with Mathcad and LabVIEW as well as which programs allow us to create figures such as the one shown above. For example, we know that Microsoft Word works well with LabVIEW: LabVIEW has the ability to do a find-and-replace for variables and put in the specific numbers in a Word template. I was not able to get to this during this semester. Future members working on this task are likely to start working on this after they have finished with the lances calculation.

3 Compiling the Full Materials List

After finalizing everything with the lances calculation (the code and outputting the cutlist), the next step woud be to include the other materials in the material list. Firstly, the other hydraulic materials (elbows, fittings, tees, valves, etc.) will need to be included. Secondly, the structural materials will be included. Lists of some these materials (which are not comprehensive) can be found in the Introduction. The person taking charge of this task will need to analyze the drawings and the code to find the number of each hydraulic material that is used in a plant. They can use the APP material lists for reference (on naming, for example). The materials list will look through the code of the design to find, for example, how many elbows of which type there are. This may require more code to build a matrix for elbows, for example, much like we built the pipe length matrix. To compile the list for structural materials we will need to work a lot with Drew and Santiago, APP's civil engineer, who knows what structural materials are need in the construction of an AguaClara plant. The list may be finally compiled in Excel. However, other options should be considered and evaluated.

4 Including Costs in the Materials List

Costs will also ideally be included in the materials list. We need to find out the prices for the different materials in order to estimate the material costs for the given plant. We will be working off of the Alauca materials list a lot for this part since that list already includes prices. Just like in the Alauca list, we will be including the quantity of a certain type of material, the price for that materials and then multiplying those to obtain the total price for that material. We will also need to consider that AguaClara is not limited to Honduras so the prices of stock will be different in other countries (this will require forethought). We may want to create a price database with the costs of all components of the plant from different countries and then the materials list can select the appropriate column (i.e country) to calculate the total cost.

5 Translating the Materials List

It is imperative that the materials list be translated to Spanish since currently plants are being constructed in Honduras so the engineers and workers that will be using the list speak Spanish. The two existing material lists (described in the Introduction) have English and Spanish versions. The person translating can work with the existing materials lists which have a lot of the technical terms in Spanish already. Additionally, some parts of the translation can be automated: many of the same words can be translated in one go (by using a "find-and-replace" type function, for example).

6 Using Materials List to Reduce Costs

Lastly, the materials list will be able to optimize the plant components to reduce costs. Different parameters in the design could be changed and then the cost estimates (from the materials list) could be compared to find the design modifications that result in the lowest costs. It is of utmost importance that the performance of the plant is maintained. The parameters could be:

- a material (for example, change from steel to an aluminum rod)
  - Grant CES Selector (a materials selection software) could be used to compare the materials [see here for more information: http://www.grantadesign.com/products/ces/]
- the specifications of the material (for example, nominal diameter)

More importantly, changes made in the design will be reflected in the final cost of the materials list. In this way, we can see how changes in the design affect the total cost in the materials list.

Part II
Documentation

Documented Progress

I began this semester by designing the overall structure of the materials list including its purpose, features and parts. I also outlined in this Report what the general steps are to create a useful materials list. I then began working on the lances calculation. Before I began writing the code to calculate the number of needed lances, I needed to create a pipe matrix that would included all the pipes in the plant design. I spent a large part of the semester finding the best way to build the pipe matrix. We wanted a matrix that would include the length of the pipe, its nominal diameter, the pipe specification and the number

of pipes with those same characteristics. Thus, the pipe matrix would have four columns: pipe length, nominal diameter, pipe specification and number of pipes.

Building the Pipe Length Matrix

I held a meeting with Monroe, Tori and Maysoon to discuss the different possibilites to build the pipe matrix.

Options Using LabVIEW

Before talking to Monroe, I had the idea of using LabVIEW. I thought that we could use LabVIEW to extract the information we wanted from the Mathcad files especially since LabVIEW currently does extract variables to input their values in the EtFlocSedFi pdf files. However, after proposing this at the meeting I found out that this would require Monroe to step in and work with LabVIEW (which is overly complicated to use) and, moreover, we would need to display all of the variables in EtFlocSedFi in order to allow for LabVIEW access to them. Therefore, LabVIEW was discouraged from the beginning as being to complicated and time-consuming to use.

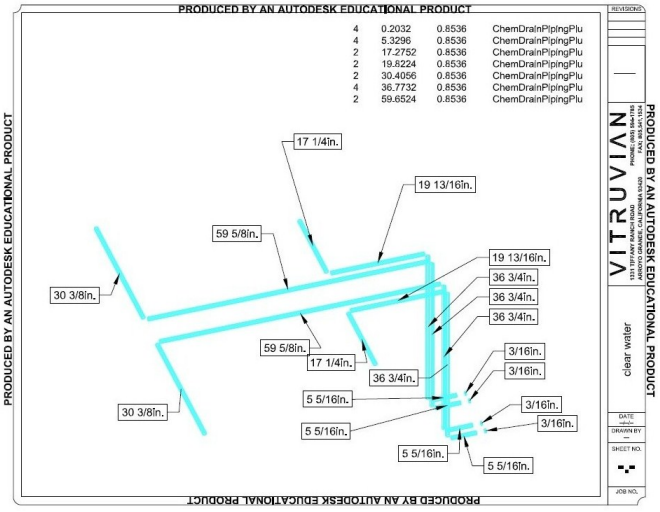
Options Using AutoCAD

After talking about how to build the array with Monroe, we decided to look into the possibilities of using AutoCAD. We thought that since AutoCAD draws out all the pipes, it must, at some point, have all the relevant pipe information I was looking for (pipe length, pipe nominal diameter, pipe spec). I spent several hours searching through forums as well as playing around with AutoCAD to see what possibilities were available. I found that when you look at the "properties" for each object, "length" does not appear; when you use the AREA command for an object it does not give you the length (as it would usually); when you use the LENGTHEN command for an object it does not provide you with the "current length" (as it would usually); it does not let you lengthen the object; it says "object does not have a length"; if you FLATTEN the object, then AutoCAD will provide a length. However, we could run into complications regarding visibility and confusion between objects if we had to flatten the entire plant. Also, we would still have to find a way to differentiate pipes with different pipe specifications. Finally, I think it would also take a long time for AutoCAD to flatten the image. Next, I decided to look at other options that AutoCAD offers such as Bill of Materials, Materials Lists and Cutlists. I found that the standard AutoCAD version we currently use does not include these options (whereas Autodesk Inventor, AutoCAD Mechanical and AutoCAD 3D Plant Design do include these options). In addition, AutoCAD 3D Plant Design has more options for pipe drawings since it is geared to plant design so it might be a good idea to look into these other versions for future upgrades. I did not look at them more in depth nor did I look into each one's advantages and drawbacks

because I decided to move forward with my task of looking for options to build the matrix.

Options Using Smartlister

I analyzed the possibility of using SmartLister which is a an add-on to AutoCAD. Smartlister calculates the dimensions of solids in an AutoCAD file. I contacted the creator of Smartlister, David Wishengrad and he refered to me the current patent-owner of the new version of the program (Vitruvian), Duane Heil, both of whom were very helpful. I learned that Vitruvian imports the plant drawings and cut lists it for you outputting 2D drawings with the dimensions of the different objects and tables with all the dimensional data and then it makes the materials list for you. This is an example of a 2D drawing produced using Vitruvian on AutoCAD:



As you can see, every pipe has a length attached to it. Both David and Duane were willing to give us the program for free because we are students and they were both really appreciative of AguaClara's work. However, we ran into many problems with Vitruvian. Firstly, I found out that it currently only works with 32-bit versions of AutoCAD drawings and we have a 64-bit version. Furthermore, it currently runs only with AutoCAD 2007 (non-student version too) and we are using AutoCAD 2013 Student Version. I run into a lot of trouble trying to run the program since the current version of Vitruvian requires too many additional programs (including AutoCAD 2007, TrueView and LongBow). Initially I wanted to use Vitruvian because I thought it was more logical, smarter and faster to use an existing program instead of spending the time to "re-invent the wheel", per se. However, overtime, it became too troublesome and the whole point of saving time was defeated. I decided that I do not like using external programs because it makes everything more complicated and less efficient and we want to make AguaClara as accessible for everyone as possible. Furthermore, it is better to contain all of AguaClara's design in a few programs (mainly Mathcad

and AutoCAD) to make it easier for everyone on the team and everyone outside that wants to use the AguaClara design tool.

Options Using Mathcad

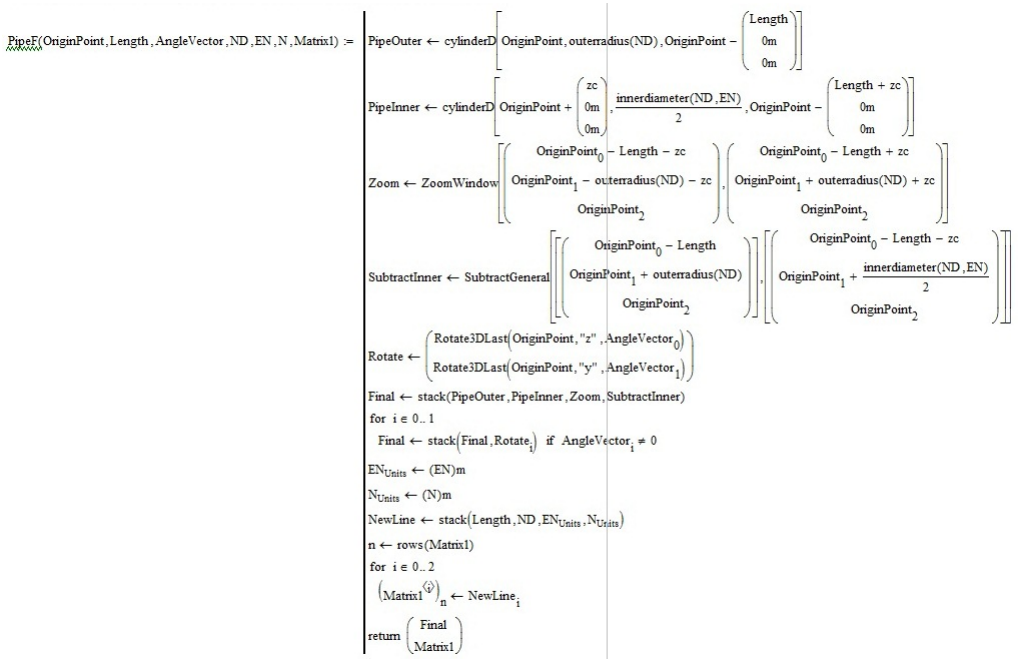
This all leads to one last option for building the matrix: writing the code for it in Mathcad. We could either include code in the Mathcad functions that would either add attribute information to the AutoCAD drawings or create the matrix directly in Mathcad. We decided to go with the second option because we thought this would streamline the process and it would keep it all within Mathcad. Tori and I decided the most straightforward way of having Mathcad output the required information would be through the PipeF function (and the SUBF function) since it is called everytime a pipe is built. This way we would make sure that all the pipes are accounted for in the pipe length matrix. This option would require the least amount of code so it would not be a very big change or addition to the code.

The Code

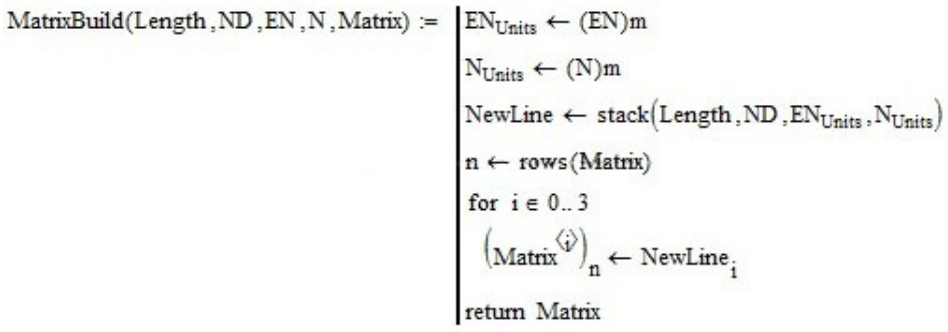
For Building the Pipe Length Matrix

The first part of the code I put together was the addition to the PipeF function that would help generate the matrix. In order to actually build the pipe matrix for the plant, I would have to go in and change PipeF and PipeSUBF in the actual Mathcad function and then go through the code and change every single call to either function. This would be time-consuming and, more importantly, it would require working with every other design team member who is modifying files including calls to PipeF or PipeSUBF to make sure all changes are properly incorporated (so that no one's modifications are lost). Tori and I decided the best approach for this would be to wait until next semester and make the changes at the start of the next semester before new design team members get started on their tasks. This would ensure I would not be conflicting with anyone else's work.

This is what the new code for PipeF looks like:



The only additions to the function are the last 6 lines of code (starting from "ENunits"). In addition, due to these added lines of code, we had to add input parameters to the function which include N and Matrix1. N is the number of pipes with the same characteristics (length, ND and EN) which will depend on the number of times the PipeF function is called with the same inputs or the number of times the SUBF function is arrayed. Here is a shortened version of the new PipeF function that just includes the added parts:



The left-hand side of the bar is just the name of the function (which is different from the real PipeF function). The first line of code on the right-hand side changes the units of the pipe specification from inches to meters. (More information on pipe specifications can be found in "PipeDatabase" under "Mathcad dimension calculations" in the EtFlocSedFi files.) The second line of code adds a unit to the number of pipes because Mathcad requires that everything in a matrix have units. In other words, this is just to ensure that the code works but

it should not change the way you understand the matrix. If you have 8 pipes with the same length, ND and EN, even if we just turned that into 8m, you should still interpret it as 8 pipes - ignore the fact we added units. The third line of code puts all this information into 1 row which is then added into the matrix with the next 3 lines of code. The final line just returns the pipe matrix which now has 1 more row.

For example, if we started of with a matrix that looks like this:

$$\text{Matrix} := (5\text{m} \quad 4\text{m} \quad 3\text{m} \quad 2\text{m})$$

And then we called the function to add a new pipe to the matrix:

$$\text{Matrix} := \text{MatrixBuild}\left(15\text{m}, \frac{1}{8}\text{in}, \text{PS}_{\text{Default}}, 5, \text{Matrix}\right)$$

$$\text{Matrix} = \begin{pmatrix} 5 & 4 & 3 & 2 \\ 15 & 3.175 \times 10^{-3} & 8 & 5 \end{pmatrix} \text{m}$$

We then see the resulting matrix which now has 1 more row. We still have the first row which is what we started with, and then we see an added row with the new pipe. As you can see, the code builds onto the existing matrix. Eventually, we will have a very large matrix that includes all the pipes in the plant.

For Calculating the Lances

After writing the code that will generate the matrix, I started working on the code that will use the matrix to calculate the number of lances needed.

The first step in working with the matrix is to make sure we are working with the matrix we need. For this, we will need to do 2 things to the matrix: firstly, we want to leave out pipes that are longer than the lance length and secondly, we will need to divide the matrix into separate submatrices where all the pipes have the same nominal diameter and pipe specification.

In order to exclude pipes that are longer than the lances I wrote the following code:

```
Lance := 6.1m
LanceUsable := Lance − 0.60m
LanceUsable = 5.5 m
```

Get rid of pipes that are longer than lance length

```
PipesTooLong(MatrixUse) := | n ← 0
                           | for i ∈ 0.. rows(MatrixUse) − 1
                           |   if (MatrixUse^⟨0⟩)_i > LanceUsable
                           |     | Matrix_n ← (MatrixUse^T)^⟨i⟩^T
                           |     | n ← n + 1
                           |     | (MatrixUse^⟨0⟩)_i ← 0
                           | return (Matrix / MatrixUse)
```

I first included the length of the lance we are working with ("LanceUsable"). Then I looked through the matrix (using a for loop) to find all the lengths that exceeded the length of the lance. If any of the pipe lengths were too long, these pipes (or rows) would be added to a new matrix (called "PipesTooLong") and eliminated in the matrix we need (called "MatrixUse" which then becomes "MatrixUseWithGoodPipes"). This PipesTooLong matrix will be used to calculate the number of lances needed for pipes that are longer than the lance length.

I was then working on the code to divide the matrix into submatrices. For now I have written the following code:

```
MatrixUseWithGoodPipes := csort(PipesTooLong(MatrixUse)_1, 2)

DivideMatrix(MatrixUseWithGoodPipes) := | for i ∈ rows(MatrixUseWithGoodPipes) − 1
                                        |   if | (MatrixUseWithGoodPipes^⟨1⟩)_i = (MatrixUseWithGoodPipes^⟨1⟩)_i−1
                                        |      | (MatrixUseWithGoodPipes^⟨2⟩)_i = (MatrixUseWithGoodPipes^⟨2⟩)_i−1
                                        | return Matrix
```

The first line of code sorts the matrix by pipe specification. I then want to split the matrices into separate matrices if they have different pipe specifications by comparing the specification of every new pipe to the one before. So if the specification was the same as the pipe before, it would go into the same matrix. Otherwise, it would go into a new matrix. I was working on this when the semester was ending. This code needs to be finalized. More specifically, the conditional if statement needs to be finished. Everytime the if statements are

true then the row is added to the current matrix, otherwise, the row is added to a new matrix.

After the code that divides the matrix, we finally have the matrices we want to work with. The code that calculates the number of lances needs to be applied to each separate submatrix. That code looks like this:

```
PipeLengthLongest := max(MatrixUseWithGoodPipes^⟨0⟩)

PipeLengthLongest = 5.4 m

PipeLongest := match(max(MatrixUseWithGoodPipes^⟨0⟩), MatrixUseWithGoodPipes^⟨0⟩)

LanceLeft := LanceUsable − PipeLengthLongest
```

```
NextPipeLength :=
    n ← 0
    for i ∈ 0.. rows(MatrixUseWithGoodPipes) − 1
        if (MatrixUseWithGoodPipes^⟨0⟩)_i ≤ LanceLeft
            VectorOfPossiblePipes_n ← (MatrixUseWithGoodPipes^⟨0⟩)_i
            n ← n + 1
    return max(VectorOfPossiblePipes)
```

I first look for the longest pipe in the matrix ("PipeLongest"). I then subtract this length from the length of the lance ("LanceUsable") which gives me "LanceLeft". Next, I loop through the matrix trying to find the longest pipe that is under "LanceLeft"; this pipe becomes "NextPipeLength". This is the traditional way of solving a cutting stock problem and it is the way Drew proposed we do the calculations (method 1). There are other ways of doing these calculations, including using midpoints in the matrix, but we do not think that would save us enough calculation time to be worth coding. Also, we could cater to different situations. For example, if a lance had 6 meters of usable length and we had six 1 meter pipes, then we would want to use those together for 1 lance. However, if we begin to cater to specific situations such as that example, it defeats the purpose of programming a generic code in the first place. The code used above, as mentioned before, is traditionally used for cutting stock problems of 1 dimension (length, in this case). It is simple and straightforward (rather than complicating the code catering to specific situations) and, most importantly, it works. It helps reduce wasted pipe and helps us calculate the number of lances we need. Therefore, we decided to not cater to specific situations. In the future, as the code is developed and tested, future members working on this task might decide differently.

Future Work

Implementing the Code for Building the Pipe Length Matrix

The first step for the person that takes charge of this task is to go into the Mathcad code and add in my code that will build the matrix. Ideally, you will implement these changes before other members of the design team begin their tasks. In this way, you will not conflict by working on the same Mathcad file. Firstly,you must go into the PipeF and SUBF functions and update the code. These functions are found in "Plumbing Functions" under "Drawing Functions" under Pipe Functions in the EtFlocSedFi files in the ADT folder of the Final Designs. Secondly, you must go through the code and find all the calls to PipeF and SUBF. At these calls, you must change the call in the following way:

$$\text{Pipe1} := \text{PipeF}\left[\begin{pmatrix}0\\0\\0\end{pmatrix}, 15\text{m}, \begin{pmatrix}90\\0\\0\end{pmatrix}\text{deg}, \frac{1}{8}\text{in}, \text{PS}_{\text{Default}}, \text{Matrix1}\right]$$

$$\begin{pmatrix}\text{Pipe2}\\\text{Matrix}\end{pmatrix} := \text{PipeF}\left[\begin{pmatrix}0\\0\\0\end{pmatrix}, 15\text{m}, \begin{pmatrix}90\\0\\0\end{pmatrix}\text{deg}, \frac{1}{8}\text{in}, \text{PS}_{\text{Default}}, \text{Matrix1}\right]$$

The calls to PipeF currently look like Pipe1 but need to be changed to look like the bottom line. This is because now we are outputting 2 variables: the pipe itself and the matrix. If you look back at the code that ammended the PipeF function to build the matrix, you see that the function now returns "Final" and "Matrix1". In order to be able to call both things separately (instead of using subscripts) we change the call in the way just shown. In this way, any references to Pipe1 (or Pipe2) somewhere else in the code do not have to be changed (they would otherwise all have to be changed to Pipe2_1). Once all these changes are made and all the code is run to generate a plant design of, for example, Q=20L/s, it should generate a matrix. In order to learn how to generate a new plant design you should talk to Julia, Heidi or May. I do not know how big the resulting matrix will be (it depends on the size of the plant which depends on the flowrate Q) but Tori estimates there should be around 100 pipes (so the sum of all Ns should be around 100). Hopefully, the code will work. Otherwise, changes will need to be made to my code so that the PipeF and SUBF functions work and the matrix is generated.

Finish the Lances Calculation Code for Method 1

First, you need to finish the code that divides the matrix into submatrices for the different pipe specifications. Everytime the if statements are true then the row is added to the current matrix, otherwise, the row is added to a new matrix. The next step in the code is to extend the lances calculation code (shown above) to keep finding the next longest pipe until "LanceLeft" = 0 or until there are no pipes in the matrix that are under "LanceLeft". In the latter case, "LanceLeft" becomes waste. Once either of these things has happened (no lance left or wasted lance left) we start again with a new lance. To do so you will need to use a while loop. We keep doing this until all the pipes in the matrix have been accounted for. In order to keep track of this, we use N. If a pipe has been accounted for, which means that it has at one point been considered "NextPipeLength", then we deduct 1 from the N for that pipe length. So, if there is only 1 pipe with that length, N will become 0. If there is more than 1 pipe with that length, N will be reduced by 1. We know that we have accounted for all the pipes in the matrix when all the Ns are 0. So the condittional for the while loop will be "when the sum of all the Ns is 0". (Another possible way of accounting for all the pipes would have been to get rid of that row from the matrix and eventually there would be no more rows in the matrix, but I decided it would be more simple to reduce N by 1 than to get rid of rows.) Thirdly, you will need to calculate the number of lances needed for the pipes that are longer than the lance length. For this you will need to add the lengths of these pipes and divide by the length of the lances. You will have to consider the campanas and ask Drew to assist you with this considering that the campanas are of different lengths.

If you need help or have doubts about how to continue writing the code for this method you can contact me (Kira Gidron) to get more insight into what my process was. You may also want to contact Drew Hart who originally came up with this idea.

Write the Lances Calculation Code for Method 2

The next step after finishing the code for method 1, will be to write the code for the second method to calculating the number of lances. You may want to test the code for method 1 only (see next section) and in the case that the results are accurate you may decide that it is not necessary to write the code for method 2. In the case that you choose to write the code for method 2, it is explained in full detail under "Method 2" which is under "Solution Approach" in Section 1 in the Problem Definition. You can also contact me if you would like more information about method 2.

Testing the Lances Calculation Code

Once the code is believed to be complete, it should be tested. Firstly, it can be testing using a test matrix. My code includes an example matrix, which you can see here:

$$\text{MatrixUse} := \begin{pmatrix} 15 & 3.175 \times 10^{-3} & 8 & 5 \\ 3 & 3.175 \times 10^{-3} & 8 & 1 \\ 2 & 6.35 \times 10^{-3} & 0 & 1 \\ 5.4 & 0.019 & 1 & 1 \\ 6.1 & 0.203 & 1 & 1 \\ 2.3 & 0.254 & 8 & 3 \\ 4.5 & 3.175 \times 10^{-3} & 8 & 4 \\ 0.7 & 3.175 \times 10^{-3} & 8 & 2 \\ 0.8 & 0.051 & 8 & 15 \\ 0.55 & 0.051 & 8 & 10 \\ 1.2 & 0.127 & 8 & 3 \\ 2.2 & 0.203 & 8 & 2 \\ 2.3 & 0.019 & 8 & 1 \\ 5 & 6.35 \times 10^{-3} & 8 & 1 \\ 5.1 & 9.525 \times 10^{-3} & 2 & 1 \\ 3.2 & 0.203 & 8 & 1 \\ 7.1 & 0.089 & 8 & 1 \\ 9.2 & 6.35 \times 10^{-3} & 2 & 1 \\ 10.4 & 6.35 \times 10^{-3} & 8 & 2 \\ 4 & 0.089 & 1 & 3 \end{pmatrix} \text{m}$$

You can change the numbers in this matrix to test out different scenarios or different specific cases (such as 6 pipes with 1 meter lengths). The number of lances needed can be calculated by hand. Secondly, the code can be tested against the San Nicolas plant material list. This is explained in more detail under "Evaluation" under Section 1 in the Problem Definition. You can use these first two evaluations to decide if method 1 or method 2 works better (in the case that you chose to write the code for method 2).

Once you have tested the codes both ways (with the example matrix and the San Nicolas plant) you can apply it to the real pipe length matrix generated by the plant. The working code can then be added to the other Mathcad files and called everytime a new plant is designed. You will want to talk to May, Tori, Heidi (or other relevant members of the design team) when you are ready

to take this step to learn how to include the code into the EtFlocSedFi final design files.

After these 4 next steps (implementing the code for building the pipe length matrix, finish writing the code for method 1, writing the code for method 2 and testing the lances calculation code) you will have finished the first part of the materials list - "Lances Calculation". I expect this will take at least 1 full semester. You would then move on to figuring out how to output the cut list. The logical progression for the materials list is outlined in the Problem Definition (from lances calculation, to outputting the cut list, adding the other hydraulic materials, adding the structural materials, including prices, translation to Spanish and finally using the list to reduce costs).