\*Note I have the program still running trying to determine the minimum circuits going up to 8 gates.

Last night I looked at everything that was found up to 7 gates and everything that was not found. I made predictions of which ones would be found using 8, 9, 10, 11, and 12 gates using a combination of methods that are in the hand written notes. That will be scanned and added to the folder containing all the notes.

I was able to create a much quicker method using the dictionary of min circuits going up to 7 gates that would solve for everything else. I will refer to this method as “Smart Elimination” while writing about it. I compared it to the Json file and two of them were not minimal and 6 were smaller. I am yet to compare the individual circuits to see what the difference was.

|  |  |  |
| --- | --- | --- |
| Truth Values | Json File | Found |
| 00010110 | 10 | 9 |
| 00101001 | 9 | 8 |
| 01001001 | 9 | 8 |
| 01100001 | 9 | 8 |
| 01101001 | 8 | 12 |
| 10010110 | 8 | 11 |
| 11100101 | 8 | 7 |
| 11101001 | 11 | 10 |

When comparing the number of gates for the remaining things, I found that my predictions for which would use 8 gates were correct, my predictions for what would use 9 gates was incorrect, there was an extra one that used only 9 gates that I predicted would be 10. There was a predicted 11 gate circuit that used 10, and a predicted 12 gate that used 11, and the last circuit used 12 as predicted. These are all in the written notes.

**To go in order, the tasks for today are as followed:**

1. Explain the solution to the question Bryan raised yesterday. What is the minimum number of circuits needed such that by rearranging the inputs you would hit all 256 truth values?
2. Show how I predicted the number of gates that would be needed for the 28/256 not found by checking up to 7 gates.
3. Draw out what the Smart Elimination algorithm does and explain how it works along with possible risks.
4. Explain the functions of get228FromFile(), checkForUse(gateCount,notUsed), and levelMaker15(num, notUsed) because I haven’t done so yet.
5. Check if the accuracy of the Smart Elimination algorithm varies with starting point.
6. Make a circuit diagram for each of the 6 cases above where the Smart Elimination algorithm with a starting point at 7 gates gave a circuit with less gates than what was in the Json file.
7. Create a wrapper function for the Smart Elimination algorithm so you can just say what you want the starting point to be and have it return the dictionary with all 256 truth values and their associated circuits.

For the first point, see **6.24.15\_Supplementary\_Notes.jpg** and **6.24.15\_Supplementary\_Notes2.txt.** The jpg file explains the function used and the goals and reasoning. The txt file shows the groups. The short answer it that 80 circuits are needed and therefore 80 groups made. Notice that there are no overlaps in groups. As mentioned in the description in the jpg file, everything in the same group should use the same number of gates. This will come into play later on. I will refer to these groupings as ‘rearrangement groupings’.

So as I mentioned before, I was able to get 228 of the truth values and their circuits by going up to 7 gates. This took approximately 3 hours and 40 minutes. I copy and pasted them to a file **228TruthValuesAndCircuits.txt**. Which was used for the Smart Elimination algorithm. There were 28 truth values for which circuits were not found. At this point refer to **6.24.15\_Supplementary\_Notes3.jpg**.

Bulletpoints 4 and 5 are both done in **6.24.15\_Supplementary\_Notes4.jpg** .

As for bullet 6, it seems the accuract of the SmartElimination algorithm is affected by starting point. It would also seem that the very next number of gates it builds will be missing some truth values as seen from starting at 4gates and moving on to 7gates. This is also seen in how starting at 7gates skips the two truth values in 8gates and doesn’t find them until the very end. Perhaps there is a better way to differentiate between what is needed and what isn’t. This finding may also imply that the 10 gate circuit and some of the 9 gate circuits may also use less than how many gates we found starting at 7. After completing bullet point 7 I will look at the set of circuits that was eliminated with 4gates starting point and with 7gates starting point and see if there is a pattern in the ones that shouldn’t have been eliminated that were. I will also try and check circuits eliminated from 3gates with a starting point of 5gates and 6gates to see if any patterns become more apparent. If a pattern is found, I can attempt to generalize this pattern and see if it finds the 11 and 12 gate circuits in the 8gate group. Also I can look into keeping a list of minimum circuits instead of just one. This way I can see what is used in anything in that list and keep it. Another thing I should do is convert the 11 and 12 gate circuits that were found to be 8 gate in the Json file from the format in the Json file to the format used here and see when we are throwing out the important circuits.

Before I do those things, I need to convert the 6 circuits that were less gates than the circuits found in the Json file to the netlist format and send it to Bryan. The file is called “Netlist of 6 smaller.txt”. It seems like the 11 gate circuit’s truths in the Json file was actually made from a 9 gate circuit not 8 gate. The one thing is being eliminated that is actually needed is “((((c.b).c).((c.b).b).a)” if this is kept, the 11 and 12 gate circuits will become 9 and 8 gate circuits respectively. Bryan suggested checking to see if any of the isomorphs of things that are unused are used and if so keep that unused thing. Strangely every isomorph of each unused thing is unused. It seems like the things that are needed that are unused yet always end up in the second half of the list of unused when sorted. Maybe there is a reason. I will try to look into this more tomorrow.

(((0.a).b).(0.a))

(((0.a).b).0)

(((0.a).b).c)

(((0.a).c).(0.a))

(((0.a).c).0)

(((0.a).c).b)

(((0.b).a).(0.b))

(((0.b).a).0)

(((0.b).a).c)

(((0.b).c).(0.b))

(((0.b).c).0)

(((0.b).c).a)

(((0.c).a).(0.c))

(((0.c).a).0)

(((0.c).a).b)

(((0.c).b).(0.c))

(((0.c).b).0)

(((0.c).b).a)

(((b.a).0).c)

(((b.a).c).(b.a))

(((c.a).0).b)

(((c.a).b).(c.a))

(((c.a).b).b)

(((c.a).b).c)

(((c.a).c).b)

(((c.b).0).a)

(((c.b).a).(c.b))

(((c.b).a).a)

(((c.b).a).b)

(((c.b).a).c)

(((c.b).b).a)

(((c.b).c).a)