This is a guide for the use of the GeneralCircuitOptimization.

**ABOUT:**

You must specifying the number of inputs, which gate types you want, the cost of each operation, the maximum cost you want to search up to, the truth value you are looking for (if any), the maximum number of inputs a single gate can have, and a directory. The program will run either until it finds all the circuits that cost less than the specified maxCost, or until it finds all truth values possible given the number of inputs, or until it finds the specified truth value it was told to look for. It will return a HashMap that maps truth values to an ArrayList of Circuits. The list of circuits for each truth value all have the same cost. This cost is the minimum cost of a circuit that can produce the truth value. The list contains all circuits with that cost that give the truth value. At certain checkpoints it will format this HashMap and save it to a file in the specified directory. The directory must contain “/” instead of “\” and it should end with “/” if it is the folder you want to save the file in. It should end with “.json” if it is the folder followed by the name of the file. If no file name is given (the directory doesn’t end with “.json”), it will generate a file name which may overwrite files in that directory if they happen to have the same name as the generated name. The generated file name will include the cost reached, the gates used, and the date. The program will also keep track of how long it took to arrive at the checkpoint when it saves the contents to a file.

**NOTES:**

* The program will overwrite files in the directory if it has the same name as the generated name if no name is specified, but this is unlikely because the date is included in the name. The programs saves a new file for each cost reached so it is recommended that you make a new folder and use that as the directory because multiple json files will be made.
* The program will return null if it is fed a set of allowed operations if all possible truth values for the number of inputs cannot be found with that set of allowed operations.
* The program works for any number of inputs >0 but may not finish in a reasonable time for more than 3 inputs (possibly not for 3 inputs either).
* The program will return null if the cost of one of the gate types is 0.
* Costs for circuits are rounded to two decimal places, but costs for operations are not.
* Setting the maxCost too high may lead to a memory error because too many things will be saved. Suggested setting is around 8 if all costs are 1.0, but depends on what the costs are and which operations are allowed.
* The maxCost is automatically lowered if it is determined that every truth value can be found with a smaller maxCost than the one given.
* If you just want it to run until the maxCost, input null for truthValueToFind.
* Adding operations that are not recognized will not affect the program. It will just remove them.
* I created my own method that can take in an ArrayList of objects and return an ArrayList of ArrayLists of different combinations of the objects. It may speed up the program slightly if you can find an alternative to it and incorporate that one instead. But the speed boost will probably not be very significant unless you are using a large number for maxFanIn.
* If these are the values for a and b, the following table describes the operations that can be used to combine a and b. a=0011, b=0101

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Truth Value** | **Representation** |
| NOT | True when the input is False | 1100 | (~a) |
|  |  |  |  |
| AND | True when both a and b are true | 0001 | (a&b) |
| NAND | Opposite of AND; True when at least one is False | 1110 | (a@b) |
|  |  |  |  |
| OR | True when at least one is true | 0111 | (a+b) |
| XOR | True when only one is True not both | 0110 | (a^b) |
| NOR | Opposite of OR; True when neither are True | 1000 | (a.b) |
| XNOR | True if both are true or both are False | 1001 | (a=b) |

**HOW TO:**

1. Open the java file and navigate to ConstantProperties.java. There will be some example values already present. Leave the first three lines creating approvedOperators, costPerOp, and opNames alone.
2. Decide how many inputs you want and set numInputs.
3. Look at the allowed ops. Inside the parentheses enclosing them list the operators you want to include. Remove the ones you don’t want. Make sure operators are surrounded with double quotation marks and that they are separated by commas.
4. Set the maxCost to what you want it to be. This can be either a decimal or a whole number. It will be converted to a decimal automatically.
5. Leave truthValueToFind as null if you want to run it until it reaches the maxCost. If you want it to stop when it finds a specific truth value, replace the null with a truth value surrounded by double quotation marks. For “don’t cares” just enter an x in that position like “1100xx01”
6. Enter the directory in the format shown as dir.
7. Set shouldSpeedUp to false if you want to find the absolute minimum for the circuits which will run more slowly. Set it to true if you want the program to run slightly faster. It will in most cases still find the smallest circuit, but there will be a chance that it will not be the absolute minimum.
8. Look at the lines that say costPerOp.put("~",1.0); in the setCostPerOp() method. Change the cost of the operations as necessary by replacing the 1.0 with the cost you want it to have. Make sure the number includes a decimal followed by a zero if it is a whole number.
9. Navigate to FindMinimalCircuit.java. Don’t change the line that creates answer, but you can add operations to perform on answer below it. Answer is a hashmap that maps strings of the truth values to a list of circuit objects.
10. Click run. Your files will start saving in the specified directory.

**Important and Useful Properties of Circuit Objects:**

Constructors:

Circuit(String symbols, String values)

Used for making circuits from scratch. Given a name and truth value, it will create a circuit. Circuits should not have parentheses in their names unless you are trying to create a circuit from a circuit string to get its netlist. Otherwise other properties of the circuit may not have the expected behavior.

Circuit (String op, ArrayList<Circuit> circuits)

Takes in a list of circuits and an operator. Combines all circuits in the list of circuits using that operator.

Other methods:

Boolean canBeUsed()

Returns false if it is a bad circuit. A bad circuit is one that is not 0 or 1 or an input that either contains a truth value o

String getTruthValue()

Returns the string that is the truth value of a circuit.

boolean isTruthValue(String tv)

String getName()

Returns the string format of a circuit

HashSet<String> getContainedTruthValues()

Returns a list of all truth values that can be found from pieces of this circuit.

String getOperator()

Returns the operator used for the circuit.

HashMap<String, String> getSubcircuits()

Returns a mapping of subcircuits of a circuit (characterized by parentheses in the name) and maps it to the operator used for the circuit.

ArrayList<String> getNetlist()

Returns the netlist for a circuits as an arraylist of strings.

**double** getCost()

Returns the cost of the circit as a double rounded to two decimal places.

**Other Tricks:**

If you are looking for the minimum circuit for a specific truth value, you can use Logic Friday to find a circuit for it and then use the number of gates needed by Logic Friday as your max cost. This is useful because sometimes circuits may be more optimal than the ones presented in logic Friday.

If you are running out of memory while searching for a specific truth value (like if Logic Friday says it requires 13 gates so you try to set maxCost to 13 and run out of memory there are some things you can try:

Open the Optimizer file and search for the word “Memory” it should appear in 8 places. The line directly under it should be either an ‘if’ or an ‘else if’. At the very end, there should be a maxCost parameter. Change the maxCost to 13(or higher). Then rerun using a lower cost. This will allow the program to print whenever it finds a circuit that gives the truth value even though it won’t save the circuit. It will print the circuit and the cost. If you set the maxCost to something like 8 and allow it to run, you will notice that some circuits are found that satisfy the truth value that will be printed. If you look through these you may notice that some circuits will have 13 some will have less. Scrolling through, look for the circuit with the smallest cost (when it prints the circuit for the opposite truth value look at the expected cost of the actual truth value). You can try running the program again with this as your new max cost. Remember to go back to all the “Memory” markers and change back the number you put in to maxCost.

If you are just looking for a circuit for a truth value you can check the file ending with “FoundButNotBest” for the truth value. If you want the absolute minimum check the other file ending with something like “Ops.~\_MaxCost2.0” with the MaxCost ending with the highest number. If the circuit is minimized in the latter, then the minimal will appear in the former, but if a truth value appears in the former and not the latter, then it may not be the absolute minimum.

Sometimes saving the unsimplified circuits takes a long time. You can choose to not save them either by commenting out any line that says

shouldDoNotSimple = **true**;

or by finding searching for

//Nonsimplified

And commenting out the four lines under it between the

//----------------