PROJECT 3 - ARTIFICIAL INTELLIGENCE

"Artificial Intelligence" (AI) is basically trying to develop computer systems which can do things that if humans did them would be considered "intelligent" behavior, *e.g.*, translate Russian text (or speech!) into English, win at chess, solve a previously unsolved math problem, etc. This project aims for your team to learn and code some fundamental AI techniques which search for a solution to a problem.

The problem is to design a circuit to implement a given logic function, using a genetic algorithm (GA) and another search method of your choice (such as depth-first, A*, etc.), subject to certain constraints, and to display the progress of the search graphically. Each circuit may contain only AND, OR, and NOT gates. There can be any number of AND and OR gates, ***but no more than two NOT gates***. The AND and OR gates may have any number of inputs per gate.

Results of the two search methods (GA and one other) must be demonstrated for two goal circuits:

1. A one-bit full adder. The truth table is

INPUTS			OUTPUTS	
X	Y	Carry In	Carry Out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2. A circuit with three inputs A, B, and C, and three outputs ¬A, ¬B, and ¬C.

For both circuits, use the following simple description language to represent a circuit as lines in a text file:

```
output-line-number   gate-type   input line number(s)
```

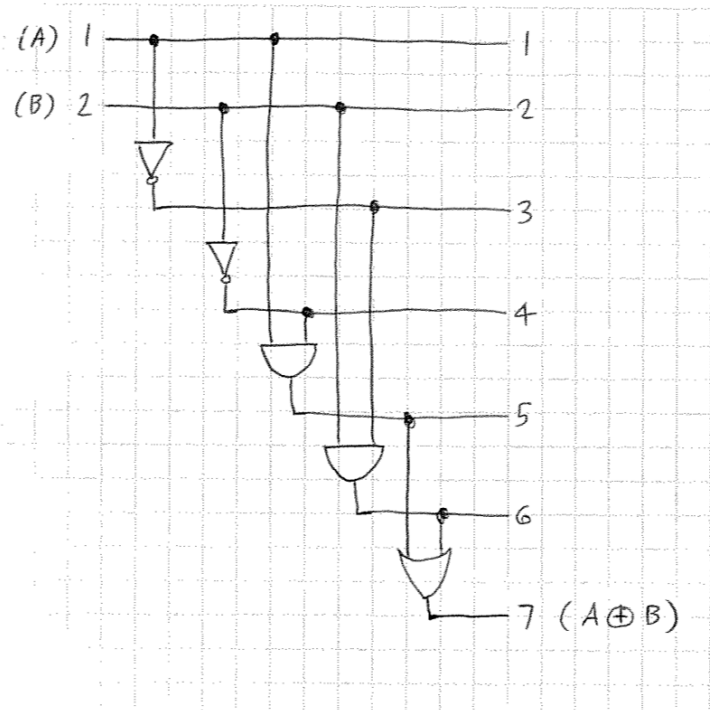
For example, if the problem were to find an XOR, the truth table is

INPUTS		OUTPUT
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

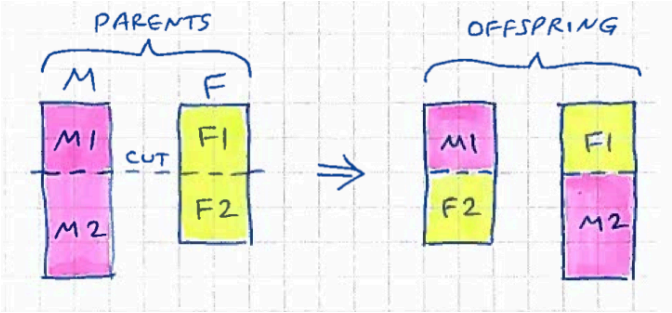
and one solution description text file would be

1 NONE 1 (meaning output line 1 is the output of a wire [no gate] from input line 1)
2 NONE 2
3 NOT 1 (meaning output line 3 is the output of a NOT gate whose input is line 1)
4 NOT 2
5 AND 1 4
6 AND 2 3
7 OR 5 6

to represent



For the GA, generate some large initial population of random circuits using this circuit description language. To "cross" two circuits, cut them at the same random line of the circuit description and resplice the pieces to generate the two "offspring" to add to the population pool:



You will need to create a fitness function to score the initial population pool and then the new offspring at each step, discarding the "least fit" at some rate. Some useful terms in a fitness function (assuming lower values are more fit) might be things like

1000000 * (number of goal output values missing) +
10000 * (number of NOT's) +
10 * (number of AND's and OR's)

This will prefer shorter circuits to longer ones, and also favor circuits with fewer NOT's (since that allows more possibilities for an offspring to have another NOT without exceeding the final limit of 2 NOT's).

Your search programs should display their progress graphically. For example, the GA could represent each individual as a rectangle of fixed width with height proportional to the number of gates and its fitness number inside, highlight the pair chosen to cross in two colors (like the picture above), and add two new rectangles for the offspring showing both parents' colors and the fitness number. Remove a rectangle from the screen when it is discarded. This is just an example; you do not have to do it this way. Another way would be to graph the minimum fitness value (y) versus number of generations (x).

Similarly, do something appropriate for your second search method to show the progress of the search graphically. Warning: Fancy graphics updated every generation might slow down your program; if so, just update the screen periodically, *e.g.*, every 100 generations.

Three possible graphics packages to consider are FLTK, Java Swing, and OpenGL, depending on the skills of your team and what is available on compute.cse.tamu.edu. (This server is where you ***must*** run your programs, since they may consume a lot of resources.) It is supposed to have the same OS configuration as unix.cse.tamu.edu, so if there is some software you need which is on unix.cse.tamu.edu but not installed on compute.cse.tamu.edu, compile on unix.cse.tamu.edu (or sun-new.cse.tamu.edu) and then run on compute.cse.tamu.edu.

WARNING! If you run a resource-intensive program on any CS server other than compute.cse.tamu.edu, your CS account may be terminated for hogging the machine!

Deliverables and due dates:

Design documents	3/15 (no grace period)
Non-GA for circuits 1 and 2	3/29 (no grace period)
GA for circuits 1 and 2	4/5 (no grace period)

Choose clever (but clean) names for your GA program (such as "Gregor's Revenge," "Multiplying Wabbits," or "3 NOT's, You're Out") and your other (non-GA) search program (such as "Inch Wide, Mile Deep," "A-Star-Trekking," etc.).