

B644 VLSI Design

(Computer Science Department, Indiana University)

Applications for Extended Analog Computers

- *Genetic Algorithm testing for evolving XOR*
- *Genetic Algorithm for Cyclotron Beam Controller*
- *Anomalous Internet Traffic Detection*

Team:

Natarajan Gopalakrishnan
Vess Miller
Kota Nagai
Radu Vlas
Philip Whitener

Advisor:

Jonathan Mills, Ph.D.

Abstract

Analog computing, the area of scientific research that had generated little interest during the last few decades, evolved continuously until starting to attract more scientific interest, which is more appropriate to its wide potential future. Some successful solution had been developed until now. The range of analog computing application is wide. Here, we use analog computing to implement a cyclotron beam controller and an internet traffic controller. In the process of research, we could see its potential advantages and further problems.

Keywords:

analog VLSI
extended analog computer
genetic algorithms
Lukasiewicz logic array
cyclotron beam controller
internet traffic controller

Evolving XOR for the Extended Analog Computer

Introduction

In this section, we attempt to reproduce the work of Ainsley, in evolving through a genetic algorithm, a design for an EAC to compute XOR. We use his code for reproduction.

Test Environment

Compiled code was that produced by Nathan Ainsley. All compilation and execution was performed under the UNIX emulator, Cygwin, on an Intel Pentium III 550Mhz running under Windows XP. Compilation was through Cygwin's GCC version 2.95.3-5.

Determining Success

In order to determine the successful evolution of an individual from a population, it is necessary to establish some general rules for determining when an individual is 'fit enough'. It is also necessary to establish when evolution should be halted, claiming that evolution has failed to produce a fit individual in a sufficient time period. For this experiment, the rules are as follows.

1. Evolution ceases on or before the 500th generation.
2. Evolution ceases when a member of the population reaches a fitness level such that the sum of error in its output is less than a threshold, .01.

Accordingly, failure in this text refers to the case when no fit individual evolved, and success refers to the case when a successful individual did evolve. Further, a population may be said to be more successful if they succeeded in producing a fit individual in fewer generations than another population.

Results

Population of 100

- No tests were successful.

Population of 250

- No tests were successful.

Population of 500

- No tests were successful.

Population of 1,000

<i>Run Number</i>	<i>Successful?</i>	<i>Number of Generations</i>
1	No	-
2	No	-
3	Yes	53
4	Yes	161
5	No	-
6	Yes	358
7	Yes	43
8	Yes	23
9	Yes	216
10	Yes	34
<i>Average:</i>	70%	126.9

Population of 10,000

<i>Run Number</i>	<i>Successful?</i>	<i>Number of Generations</i>
1	Yes	40
2	Yes	4
3	Yes	7
4	Yes	24
5	Yes	25
6	Yes	61
7	Yes	39
8	Yes	5
9	Yes	5
10	Yes	34
<i>Average:</i>	100%	24.4

Discussion of Results

For sufficiently large populations we can see that it is possible to evolve EAC's to produce correct results within .01 for the chosen non-linearly separable function. This confirms the original evolution of the XOR EAC by Ainsley.

Further, we can see through limited testing that genetic algorithms require a certain number of individuals to achieve success regularly. We can hypothesize that this increase in success is correlated to the increase in diversity offered by larger populations. We propose, in keeping with the biological language of genetic algorithms, that populations in genetic algorithms can be viewed as a composition of smaller populations that share common characteristics. That is, a population consists of families, where families consist of groups of individuals who share certain common characteristics. When inbreeding occurs, there is very little, if any advancement, because there is very little diversity to fuel change. However, when members of different families breed, there is a chance for exchange of information, and a synergistic advancement may occur. When populations are too small, they may in fact be forced to inbreed, and never advance. Only more extensive testing and analysis can determine the validity of this hypothesis.

Cyclotron Beam Controller

Indiana University Cyclotron Facility (later referred to as IUCF) currently uses open_loop control of the steering and focusing magnets on the beam line. Sensor data from beam position monitors (later referred to as BPMs) is presented to a human operator, who adjusts the magnetic field of each dipole and quadrupole magnet to maximize beam intensity. There is no direct feedback from the BPMs to the dipole and quadrupole magnets, which means that deviations from the engineered trajectory must be manually tuned each time the beam is restarted for a new experiment. Closed_loop control with a digitally_reconfigurable VLSI EAC provides feedback from a BPM to a specific dipole or quadrupole magnet. The EAC can be adjusted for the beam optics at each magnet's location, as well as operational variations detected by the human beam line operator (Figure 1). Improvements anticipated are reduced time to tune the beam, increased beam temperature, and increased beam stability.¹

¹ "Programmable VLSI Extended Analog Computer for Cyclotron Beam Control" – Mills, Jonathan; [Sep 1995]

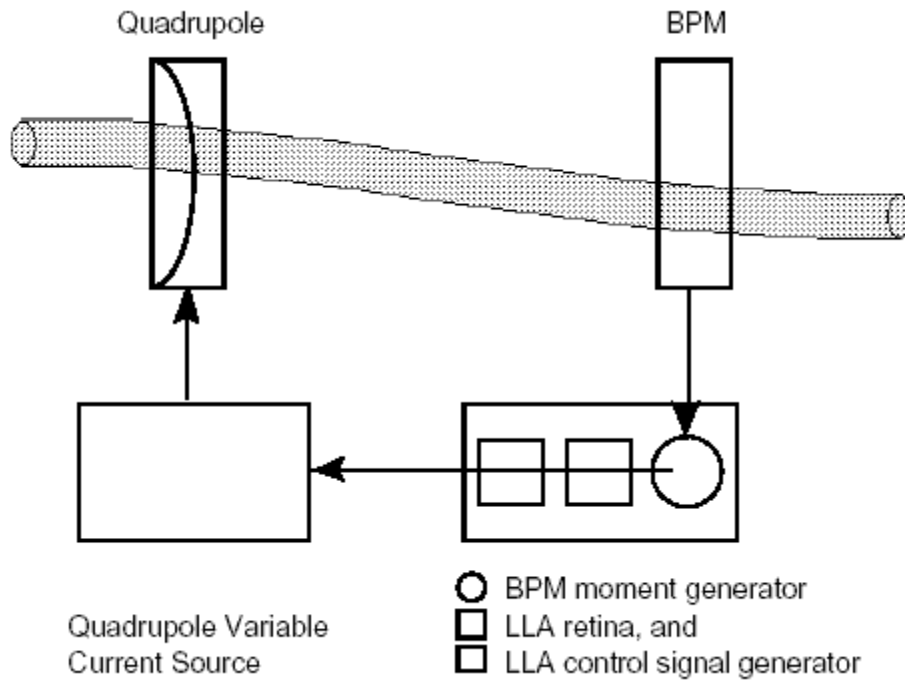
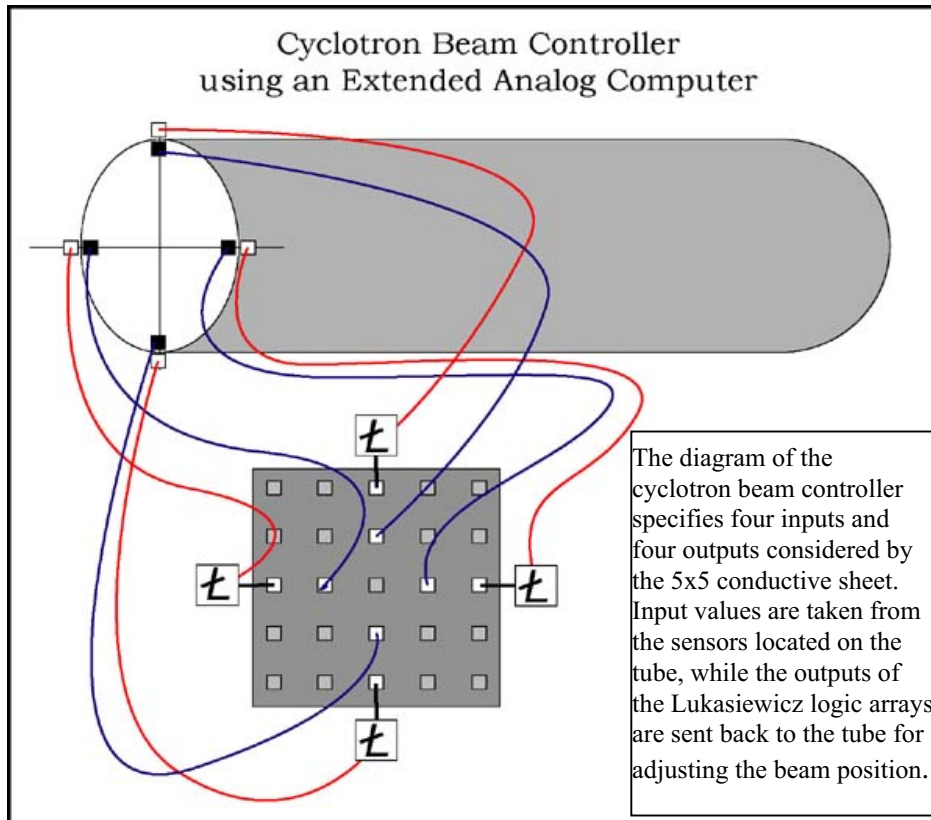


Fig.1

The beam line controller is implemented as a digitally_reconfigurable VLSI EAC. It was designed visually by creating an object analog for the beam cross section, a sensor analog to “view” the beam cross section, and a response analog to generate the control signal based on the difference between the beam's measured and desired positions. The object analog is a conductive sheet that generates a surface that combines the inverse of the beam's moments. The sensor analog is a Lukasiewicz Logic Array (later referred to as LLA) retina. The response analog is a pair of LLAs that generate an adjustable control signal proportional to surface gradients along the X_ and Y_ axis of the beam cross_section reconstructed by the conductive sheet. These gradients are proportional to the beam moments, which need not be computed explicitly to generate a real_time control signal for the dipole and quadrupole magnets in the IUCF beam line.¹

Until now, an extended analog computer was designed as a cyclotron beam line controller. Our team tried to use a 5x5 conductive sheet for adjusting the beam position for a particular section of the beam line. The structure of a section is presented in Fig.2 below:

¹ “Programmable VLSI Extended Analog Computer for Cyclotron Beam Control” – Mills, Jonathan; [Sep 1995]



The tube has a frame made of four magnets which is used for the adjusting the beam position. The magnet's power depends on its input voltage. Higher a magnet's input voltage, higher its repulsion is. Our goal is to provide the (magnets) frame with the right input voltage to move the beam to the desired position.

In the diagram above, the sensors are sending information to four conductive sheet locations through the blue wires. According to these values, we have specific values in the four conductive sheet connections we are using for adjusting the beam position. These four connections are providing the corresponding LLAs with their voltages. The four magnets we are using for adjusting the beam position are provided with the LLA's outputs (through red wires).

Based on the afore-mentioned information, one problem that crops up is what LLA functions to be used in order to be able to provide the frame with the appropriate input voltage. We are using a genetic algorithm approach for solving this problem. A C++ program was first developed by a team in the VLSI class (Fall '02) for implementing a genetic algorithm which provides solutions for the XOR problem. We adapted this code to be used for the Cyclotron Beam Controller. We also used a genome structure that is presented in the following section.

Structure for Genetic Algorithm

The genome is divided up into 4 basic sections or "chromosomes," the sheet inputs, the sheet outputs, the LLAs and a floating point value which represents the voltage of each magnet.

Initially, the genetic algorithm (later referred to as GA) was implemented for 3 _ 3 sheets. We considered a 5 _ 5 conductive sheet in order to achieve better results and have more possible combinations to search through for a result. We also took into consideration the fact that we use four connections of one conductive sheet as inputs from the sensors and four other connections of the same conductive sheet as outputs to magnets.

For the considered 5 _ 5 conductive sheet, we have the following structure of connections (indexed the way represented below):

0	1	2	3	4
5	6	7	8	9
9	10	11	12	13
14	15	16	17	18
19	20	21	23	24

Actually, one dimensional array is used for the convenience of the GA. Actual positions are as above.

Next, the “Evolving XOR” code takes functions like XOR, AND, and IMP. Based on a function, the program runs and returns number of genome, genome size, and average errors of outputs. We have created new function called “CYCEval” stands for Cyclotron Beam Controller evaluation function.

During our work on evolving the cyclotron beam controller, we always had a visual idea of how we thought the optimal solution would be.

I				I
	O		O	
	O		O	
I				I

Our optimal inputs and outputs.

?				?
	?		?	
	?		?	
?				?

Undecided LLAs

We anticipated four inputs and four outputs, for outputs having attached LLAs which would translate the appropriate adjustment to the beam. When we ran the evaluation program (CYCEval.cpp) with Ainslie’s extended analog computer evolution program, slightly modified for our 5x5 conductive sheet (compared to the original 3x3 sheet), we noticed the computer makes random selections on the placement of each of the four inputs and four outputs. In our tests of generations, the computer made many random attempts at finding a solution.

Input and Output

	O			
			I	O
	I		I	
				I
			O	O

Computer randomized inputs and outputs.

LLAs

	0			
				0
			8	16

Computer randomized LLAs.

Although our solution never came up as a chosen solution for the problem, many of the good cases showed signs of symmetry like ours was based off of (as seen above). Possibly a larger population of generations and genomes could prove our optimal output as a good solution.

Derivatives and the Rate of Change (For Internet Traffic Controller Temporal Pattern Matching)

When average traffic patterns are known, it is possible to determine anomalies in current traffic patterns. The detection of irregular traffic is beneficial for many reasons; among these are detecting attacks on servers, and router configuration to compensate for heavy traffic loads. To determine these anomalies in Internet traffic in real time, as it happens, we can look at a current time slice, for example the last ten seconds, and compare it to the same time of day on the average, or the expected traffic pattern.

Changing the graph of the current traffic into a linear function, the derivative of the function will result in an integer, equivalent to the slope of the function (Fig 2). The derivative is the rate of change of the function (with respect to time), and does not compensate for changes in volume by offsetting the function, also true for the slope of a function (Fig 3).

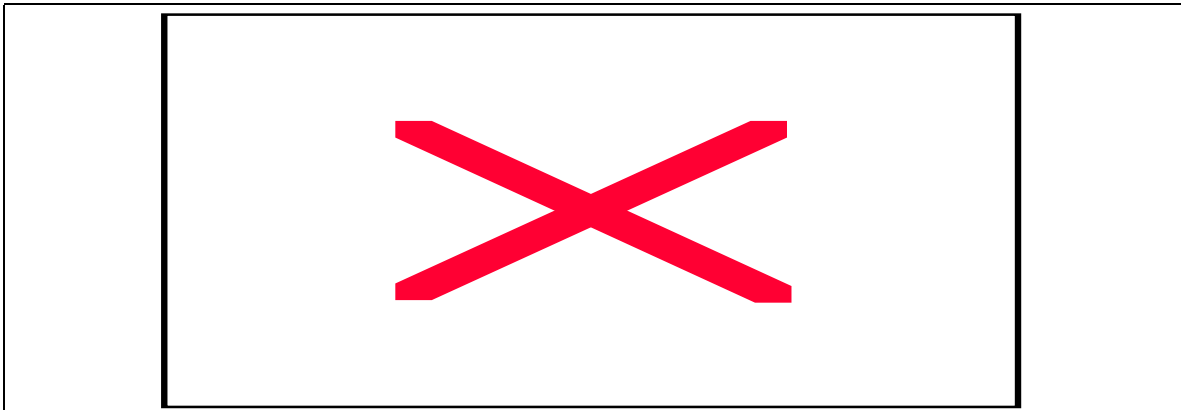


Fig. 2: Linear function, $f(x) = 10x$, and its derivative, $f'(x) = 10$.

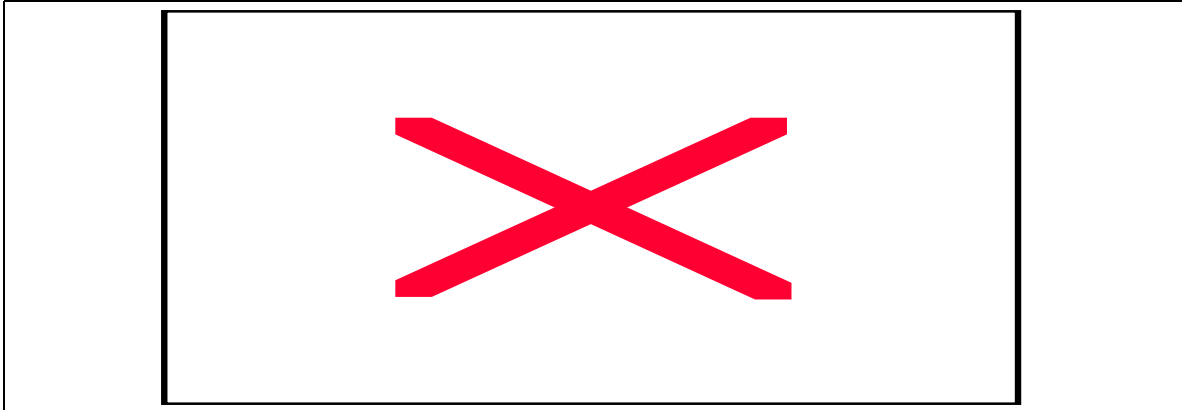


Fig. 3: Linear function, $f(x) = 10x + 20$, and its derivative, $f'(x) = 10$.

In a situation where the derivate of a current time slice is greater than the derivate of the time slice corresponding to the same time of day of the average (or expected) traffic linear function, then an attack may be beginning. The allowed amount of growth, or difference between the average and current would be set for the concern of change; a difference of 1 or 2 may be acceptable, but a difference of 10 or more may be room for concern. A negative derivative would symbolize a decreasing function, or negative slope (Fig 4). It would also be important to know if traffic was not falling off as expected.

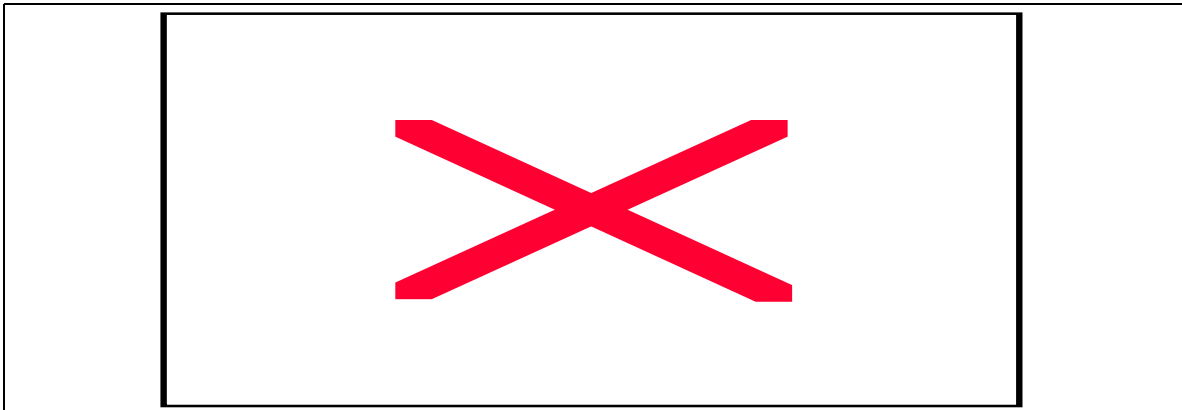


Fig. 4: Linear function, $f(x) = -10x$, and its derivative, $f'(x) = -10$.

References

- [1] Mills,J., " Lukasiwicz' Insect: Continuous-Valued Robotic Control after Ten Years".
- [2] Mills,J., "Programmable VLSI Extended Analog Computer for Cyclotron Beam Control", 1995.
- [3] Ainslie,N., "Evolving XOR" code posted to the VLSI design class newsgroup, Fall 2002
- [4] Team Herbie, Modified header files (genome.h, conductive_sheet.h) for 5x5

Appendices

Appendix A (the code for evolving XOR):

```
/******  
    EVO_EAC  
    Nathan Ainslie  
    nainslie@indiana.edu  
    Oct 2002  
  
    main.cpp  
***** /  
  
#include "ConductiveSheet.h"  
#include "Genome.h"  
#include "LLA.h"  
#include "Eval.h"  
#include "XOREval.h"  
#include "ANDEval.h"  
#include "IMPEval.h"  
#include "BreedingPop.h"  
#include "God.h"  
  
#include <stdio.h>  
#include <string.h>  
  
#define RES 2.0  
  
int NUM_GENS = 100;  
int GEN_SIZE = 10000;  
  
int main(int argc, char* argv[]) {  
    int i = 4;  
    FILE* pOutputFile = NULL;  
    CEval* pEval;  
    if (argc >= 4) {  
        NUM_GENS = strtol(argv[1], NULL, 0);  
        GEN_SIZE = strtol(argv[2], NULL, 0);  
        if (!(strcmp("AND", argv[3]))) {  
            pEval = new CANDEval(RES);  
        } else if (!(strcmp("XOR", argv[3]))) {  
            pEval = new CXOREval(RES);  
        } else if (!(strcmp("IMP", argv[3]))) {  
            pEval = new CIMPEval(RES);  
        }  
    } else {  
        fprintf(stderr, "Usage: %s num_generations generation_size function  
[-f output file] [-m mutation_rate] [-v]\n", argv[0]);  
        exit(0);  
    }  
}
```

```

CGod oGod(NUM_GENS, GEN_SIZE);

while(i < argc) {
    if(!(strcmp("-f", argv[ i ]))) {
        pOutputFile = fopen(argv[ i+1 ], "w");
        oGod.setOutputFile(pOutputFile);
        i += 2;
    } else if (!(strcmp("-m", argv[ i ]))) {
        oGod.setMutationRate(strtod(argv[ i+1 ], NULL));
        i += 2;
    } else if (!(strcmp("-v", argv[ i ]))) {
        oGod.setVerbosity(1);
        i++;
    } else if (!(strcmp("-e", argv[ i ]))) {
        oGod.setMinErr(strtod(argv[ i+1 ], NULL));
        i +=2;
    } else {
        i++;
    }
}

oGod.setEvaluator(pEval);
oGod.go();

delete pEval;
if (pOutputFile)
    fclose(pOutputFile);

return 0;
}

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

    BreedingPop.cpp
*****/

#include "BreedingPop.h"
#include <stdio.h>

CBreedingPop::~CBreedingPop() {
    deleteChildren(m_gs);
}

void CBreedingPop::deleteChildren(GENOMELIST* pList) {
    if (pList->next != NULL)
        deleteChildren(pList->next);
}

```

```

    delete pList;
    return;
}

CBreedingPop::CBreedingPop() {
    m_gs = new GENOMELIST;
    m_gs->next = NULL;
    m_count = 0;
}

int CBreedingPop::addGenome(CGGenome* pG, float error) {
    //fprintf(stderr, "Adding %p.\n", pG);

    return this->addGenomeInOrder(pG, error);
}

int CBreedingPop::addGenomeInOrder(CGGenome* pG, float error) {
    int i = 0;
    //fprintf(stderr, "Adding item %d.\n", m_count);
    if (m_count == 0) {
        //fprintf(stderr, "Adding first item %p.\n", pG);
        m_gs->pGenome = pG;
        m_gs->error = error;
        m_gs->next = NULL;

        m_count++;
        m_maxerror = m_maxerror > error ? m_maxerror : error;
        return 1;
    } else {

        GENOMELIST* pList = m_gs;
        GENOMELIST* pPrev = NULL;

        while ((pList != NULL) && (pList->error <= error)) {
            //fprintf(stderr, "Looping, pList = %p, pList->next = %p\n",
pList, pList->next);
            pPrev = pList;
            //fprintf(stderr, "pPrev assigned.\n");
            pList = pList->next;
            i++;
        }

        //fprintf(stderr, "Insert location found : %d.\n", i);

        GENOMELIST* newItem = new GENOMELIST;
        newItem->pGenome = pG;
        newItem->error = error;
        newItem->next = pList;

        if (pPrev != NULL) {
            pPrev->next = newItem;
        } else {
            m_gs = newItem;
        }
    }
}

```

```

        m_count++;
        m_maxerror = m_maxerror > error ? m_maxerror : error;
        return 1;
    }
}

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

    ConductiveSheet.cpp
*****/

#include "ConductiveSheet.h"
CConductiveSheet::CConductiveSheet(float resistance) {
    m_resistance = resistance;
    m_sources = new SOURCE[ MAX_SOURCES ];
    m_sourceIndex = 0;
}

CConductiveSheet::~CConductiveSheet() {
    delete [] m_sources;
}

void CConductiveSheet::ClearSheet() {
    m_sourceIndex = 0;
}

int CConductiveSheet::AddSource(float x, float y, float voltage) {
    if (m_sourceIndex >= MAX_SOURCES) {
        fprintf(stderr, "Error adding source, too many sources!\n");
        return -1;
    } else {
        //fprintf(stdout, "Adding source at (%f, %f) with V= %f\n", x, y,
voltage);
        SOURCE* pSource = &m_sources[ m_sourceIndex ];
        pSource->x = x;
        pSource->y = y;
        pSource->voltage = voltage;

        m_sourceIndex++;
        return 0;
    }
    return 1;
}

float CConductiveSheet::VoltageAt(float x, float y) {
    float totalVoltage = 0.0;
    float answer = 0.0;
    for (int i = 0; i < m_sourceIndex; i++) {
        totalVoltage += m_sources[ i ].voltage /

```

```

        ((m_sources[ i ].x - x) * (m_sources[ i ].x -x)) + ((m_sources[ i ].y
- y) * (m_sources[ i ].y - y));
    }

    answer = (m_resistance / M_PI) * totalVoltage;

    return (answer >= 0) ? answer : 0.0;
}

#include "Eval.h"

CEval::CEval() {
}

CEval::~~CEval() {
}

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

    Genome.cpp
*****/

#include "Genome.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

#define MIN_SPACING 0.1
#define MAX_SPACING 5.0

float CGenome::m_mutRate = 0.05;

CGenome::CGenome() {
}

CGenome::~~CGenome() {
}

void CGenome::setMutationRate(float rate) {
    CGenome::m_mutRate = rate;
}

float CGenome::getMutationRate() {
    return m_mutRate;
}

```

```

int CGenome::mutate() {

    float temp = ((float) rand() / (float) RAND_MAX);
    // fprintf(stderr, "checking mutation %f \n\n", temp);
    return (temp > m_mutRate) ? 0 : 1;
}

void CGenome::breed(CGenome* other, CGenome* newGenome) {
    int coPt1, coPt2, coPt3;
    float spacingRatio;

    coPt1 = rand() % SHEET_LOCATIONS;
    coPt2 = rand() % SHEET_LOCATIONS;
    coPt3 = rand() % (SHEET_LOCATIONS * 3);

    // fprintf(stderr, "Starting breeding between %p and %p -> %p.\n",
this, other, newGenome);
    for(int i = 0; i < SHEET_LOCATIONS; i++) {
        if (i > coPt1)
            if (mutate())
                newGenome->inputCodons[ i] = other->inputCodons[ i];
            else
                newGenome->inputCodons[ i] = rand() % 2 == 0 ? 0 : 1;
        else
            if (mutate())
                newGenome->inputCodons[ i] = this->inputCodons[ i];
            else
                newGenome->inputCodons[ i] = rand() % 2 == 0 ? 0 : 1;
    }

    for(int i = 0; i < SHEET_LOCATIONS; i++) {
        if (i > coPt2)
            if (mutate())
                newGenome->outputCodons[ i] = other->outputCodons[ i];
            else
                newGenome->outputCodons[ i] = rand() % 2 == 0 ? 0 : 1;
        else
            if (mutate())
                newGenome->outputCodons[ i] = this->outputCodons[ i];
            else
                newGenome->outputCodons[ i] = rand() % 2 == 0 ? 0 : 1;
    }

    for(int i = 0; i < SHEET_LOCATIONS * 3; i++) {
        if (i > coPt3)
            if (mutate())
                newGenome->llaCodons[ i] = other->llaCodons[ i];
            else {
                switch (rand() % 3) {
                case 0:
                    newGenome->llaCodons[ i] = LLA_ZERO_VALUE;
                    break;
                case 1:
                    newGenome->llaCodons[ i] = LLA_HALF_VALUE;
                    break;
                case 2:

```

```

        newGenome->llaCodons[ i ] = LLA_ONE_VALUE;
        break;
    }
}
else
    if (mutate())
        newGenome->llaCodons[ i ] = this->llaCodons[ i ];
    else {
        switch (rand() % 3) {
        case 0:
            newGenome->llaCodons[ i ] = LLA_ZERO_VALUE;
            break;
        case 1:
            newGenome->llaCodons[ i ] = LLA_HALF_VALUE;
            break;
        case 2:
            newGenome->llaCodons[ i ] = LLA_ONE_VALUE;
            break;
        }
    }
}

// blend the two spacings
if (mutate()) {
    spacing = ((float) rand() / (float) RAND_MAX) * (MAX_SPACING -
MIN_SPACING);
    spacing += MIN_SPACING;
} else {
    spacingRatio = (float) rand() / (float) RAND_MAX;
    newGenome->spacing = (spacingRatio * this->spacing) + ((1.0 -
spacingRatio) * other->spacing);
}

//fprintf(stderr, "fixing offspring.\n");

// count the inputs/outputs to make sure that we still have the
// appropriate number
int numInputs = 0;
int numOutputs = 0;

for(int i = 0; i < SHEET_LOCATIONS; i++) {
    if (newGenome->inputCodons[ i ])
        numInputs++;
    if (newGenome->outputCodons[ i ])
        numOutputs++;
}

// fprintf(stderr, "inputs: %d, outputs: %d\n", numInputs,
numOutputs);

// make sure that inputs and outputs didn't randomly end up in the
same place

for(int i = 0; i < SHEET_LOCATIONS; i++) {

```

```

    if (newGenome->inputCodons[ i ] == 1 && newGenome->outputCodons[ i ] ==
1) {
    if (rand() % 2) {
    newGenome->inputCodons[ i ] = 0;
    numInputs--;
    } else {
    newGenome->outputCodons[ i ] = 0;
    numOutputs--;
    }
    }
}

//newGenome->PrintGenome(stderr);

// fprintf(stderr, "deleting inputs/outputs\n");
// we also might need to remove some inputs/outputs
while (numInputs > m_inputs) {
    newGenome->remRandomInput();
    numInputs--;
}

while (numOutputs > m_outputs) {
    newGenome->remRandomOutput();
    numOutputs--;
}

// fprintf(stderr, "adding inputs/outputs\n");
// we might need to add some inputs/outputs
while (numInputs < m_inputs) {
    newGenome->addRandomInput();
    numInputs++;
}

while (numOutputs < m_outputs) {
    newGenome->addRandomOutput();
    numOutputs++;
}

//fprintf(stderr, "offspring fixed.\n");

// a mutant is born!
return;
}

int CGenome::addRandomInput() {
    while (1) {
        int temp = rand() % SHEET_LOCATIONS;
        if(this->inputCodons[ temp ] == 0 && this->outputCodons[ temp ] == 0) {
            inputCodons[ temp ] = 1;
            return temp;
        }
    }
}
}

```

```

int CGenome::remRandomInput () {
    while (1) {
        int temp = rand() % SHEET_LOCATIONS;
        if(this->inputCodons[ temp] == 1) {
            inputCodons[ temp] = 0;
            return temp;
        }
    }
}

int CGenome::addRandomOutput () {
    while (1) {
        int temp = rand() % SHEET_LOCATIONS;
        if(this->outputCodons[ temp] == 0 && this->inputCodons[ temp] == 0) {
            outputCodons[ temp] = 1;
            return temp;
        }
    }
}

int CGenome::remRandomOutput () {
    while (1) {
        int temp = rand() % SHEET_LOCATIONS;
        if(this->outputCodons[ temp] == 1) {
            outputCodons[ temp] = 0;
            return temp;
        }
    }
}

CGenome::CGenome(int numInputs, int numOutputs) {
    // Randomly assign the appropriate number of inputs and outputs to
    // the sheet
    m_inputs = numInputs;
    m_outputs = numOutputs;

    spacing = ((float) rand() / (float) RAND_MAX) * (MAX_SPACING -
MIN_SPACING);
    spacing += MIN_SPACING;

    for(int i = 0; i < SHEET_LOCATIONS; i++) {
        inputCodons[ i] = 0;
        outputCodons[ i] = 0;
    }

    for(int i = 0; i < numInputs; i++)
        this->addRandomInput ();

    for(int i = 0; i < numOutputs; i++)
        this->addRandomOutput ();

    // Randomly assign values for the LLA's

    for(int i = 0; i < SHEET_LOCATIONS * 3; i++) {

```

```

        int temp = rand() % 30;
        int newVal;
        if (temp < 10)
            newVal = LLA_ZERO_VALUE;
        if (temp >= 10 && temp < 20)
            newVal = LLA_HALF_VALUE;
        if (temp >= 20)
            newVal = LLA_ONE_VALUE;

        llaCodons[ i ] = newVal;
    }
}

void CGenome::PrintGenome(FILE* fp) {
    fprintf(fp, "Spacing: %f\n", spacing);

    fprintf(fp, "Sheet Inputs: ");
    for(int i = 0; i < SHEET_LOCATIONS; i++)
        fprintf(fp, "|%d", inputCodons[ i ]);

    fprintf(fp, "\n");

    fprintf(fp, "Sheet Outputs: ");
    for(int i = 0; i < SHEET_LOCATIONS; i++)
        fprintf(fp, "|%d", outputCodons[ i ]);

    fprintf(fp, "\n");

    fprintf(fp, "LLA's: ");
    for(int i = 0; i < SHEET_LOCATIONS * 3; i++)
        fprintf(fp, "|%d", llaCodons[ i ]);

    fprintf(fp, "\n");
}

void CGenome::PrintGenome() {
    this->PrintGenome(stdout);
}

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

    God.cpp
*****/

#include "God.h"

CGod::CGod(int gens, int pop) {
    m_numgens = gens;
    m_gensize = pop;
    m_minErr = 0.0;
    m_reppop = m_gensize / 10;
}

```

```

    m_outputStream = stdout;
    m_verbosity = 0;
}

CGod::~CGod() {

}

void CGod::setMinErr(float err) {
    m_minErr = err;
}

void CGod::setEvaluator(CEval* pEval) {
    m_pEval = pEval;
}

int CGod::setOutputFile(FILE* pFile) {
    m_outputStream = pFile;
    return 1;
}

void CGod::setMutationRate(float rate) {
    CGenome::setMutationRate(rate);
}

void CGod::setVerbosity(int level) {
    m_verbosity = level;
}

void CGod::go() {
    int nIn, nOut;
    CGenome* pPrevGen[m_gensize];
    CBreedingPop oFinalGen;
    int bFound = 0;
    fprintf(m_outputStream, "Running %d generations of size %d\n",
m_numgens, m_gensize);
    fprintf(m_outputStream, "Mutation rate: %f \n\n",
CGenome::getMutationRate());

    srand(time(0));

    nIn = m_pEval->getNumInputs();
    nOut = m_pEval->getNumOutputs();

    // initialize the initial population
    for(int i = 0; i < m_gensize; i++)
        pPrevGen[i] = new CGenome(nIn, nOut);

    if (m_verbosity)
        fprintf(stderr, "Population Initialized\n");

    for(int i = 0; i < m_numgens; i++) {
        // make the new generation from the old
        CBreedingPop oPop;
        CGenome* oBreeders[m_reppop];
        CGenome* newGen[m_gensize];

```

```

if (m_verbosity)
    fprintf(stderr, "Generation %d being bred\n", i);

for(int j = 0; j < m_gensize; j++) {
    m_pEval->setGenome(pPrevGen[ j] );
    oPop.addGenome(pPrevGen[ j] , m_pEval->eval());
}

GENOMEELIST* pList = oPop.m_gs;

if(pList->error <= m_minErr) {
    fprintf(m_outputStream, "Successful organism found in generation
%d\n\n", i);
    break;
}

for(int j = 0; j < m_reppop; j++) {
    oBreeder[ j] = pList->pGenome;
    pList = pList->next;
}

for(int j = 0; j < m_gensize; j++) {
    int dadidx, momidx;
    CGenome* dad = NULL;
    CGenome* mom = NULL;
    CGenome* baby = NULL;
    dadidx = rand() % m_reppop;
    momidx = rand() % m_reppop;
    dad = oBreeder[ dadidx];
    mom = oBreeder[ momidx];
    //fprintf(stderr, "Breeding %d and %d.\n", dadidx, momidx);
    baby = new CGenome(2,1);

    mom->breed(dad, baby);
    newGen[ j] = baby;
}

// clean up the genomes from the old OLD generation
for(int j = 0; j < m_gensize; j++) {
    delete pPrevGen[ j];
    pPrevGen[ j] = newGen[ j];
}

//pPrevGen = newGen;
}

for(int i = 0; i < m_gensize; i++) {
    m_pEval->setGenome(pPrevGen[ i] );
    oFinalGen.addGenome(pPrevGen[ i] , m_pEval->eval());
}

GENOMEELIST* pWinnerList = oFinalGen.m_gs;
for(int i = 0; i < 5; i++) {
    pWinnerList->pGenome->PrintGenome(m_outputStream);
    fprintf(m_outputStream, "Error: %f\n\n", pWinnerList->error);
    m_pEval->setGenome(pWinnerList->pGenome);
}

```

```

        m_pEval->eval(1, m_outputStream);
        fprintf(m_outputStream, "\n\n");
        fprintf(m_outputStream, "-----\n");

        pWinnerList = pWinnerList->next;
    }

}

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

    IMPEval.cpp
*****/

#include "IMPEval.h"

#include <math.h>

#define SHEET_RESISTANCE 1.00

CIMPEval::CIMPEval() {
}

CIMPEval::CIMPEval(float resistance) {
    m_resistance = resistance;
}

CIMPEval::~CIMPEval() {
}

int CIMPEval::getNumInputs() {
    return 2;
}

int CIMPEval::getNumOutputs() {
    return 1;
}

void CIMPEval::setGenome(CGenome* pG) {
    m_pGenome = pG;
}

float CIMPEval::eval(int bVerbose, FILE* outputFile) {
    int inputIdx1 = 0;
    int inputIdx2 = 0;
    int outputIdx = 0;

```

```

int bFirstIndexFound = 0;
int bOutputIndexFound = 0;
float inLoc1x, inLoc1y;
float inLoc2x, inLoc2y;
float outLocx, outLocy;
int i = 0;
CConductiveSheet* pSheet;
CLLA* oLLAs[ SHEET_LOCATIONS];
float volt1, volt2, volt3, volt4;
float test1, test2, test3, test4;
float err1, err2, err3, err4;

while(inputIdx2 == 0) {
    if(m_pGenome->inputCodons[ i] == 1) {
        if(bFirstIndexFound == 0) {
            inputIdx1 = i;
            bFirstIndexFound = 1;
        } else {
            inputIdx2 = i;
        }
    }
    i++;
}

i = 0;

while (!(bOutputIndexFound)) {
    if(m_pGenome->outputCodons[ i] == 1) {
        outputIdx = i;
        bOutputIndexFound = 1;
    } else {
        i++;
    }
}

//fprintf(stderr, "Input idx1: %d, Input idx2: %d\n", inputIdx1,
inputIdx2);
//fprintf(stderr, "Output idx: %d\n", outputIdx);

for(i = 0; i < SHEET_LOCATIONS; i++) {
    float zeroVal, halfVal, oneVal;

    if(m_pGenome->llaCodons[ 3*i + 0] == LLA_ZERO_VALUE)
        zeroVal = 0.0f;
    if(m_pGenome->llaCodons[ 3*i + 0] == LLA_HALF_VALUE)
        zeroVal = 0.5f;
    if(m_pGenome->llaCodons[ 3*i + 0] == LLA_ONE_VALUE)
        zeroVal = 1.0f;

    if(m_pGenome->llaCodons[ 3*i + 1] == LLA_ZERO_VALUE)
        halfVal = 0.0f;
    if(m_pGenome->llaCodons[ 3*i + 1] == LLA_HALF_VALUE)
        halfVal = 0.5f;
    if(m_pGenome->llaCodons[ 3*i + 1] == LLA_ONE_VALUE)
        halfVal = 1.0f;

    if(m_pGenome->llaCodons[ 3*i + 2] == LLA_ZERO_VALUE)

```

```

        oneVal = 0.0f;
        if(m_pGenome->llaCodons[ 3*i + 2] == LLA_HALF_VALUE)
            oneVal = 0.5f;
        if(m_pGenome->llaCodons[ 3*i + 2] == LLA_ONE_VALUE)
            oneVal = 1.0f;

        oLLAs[ i] = new CLLA(zeroVal, halfVal, oneVal);
    }

    pSheet = new CConductiveSheet(m_resistance);

    inLoc1x = (inputIdx1 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
    inLoc1y = (inputIdx1 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

    inLoc2x = (inputIdx2 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
    inLoc2y = (inputIdx2 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

    outLocx = (outputIdx % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
    outLocy = (outputIdx / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

    // test 0,0
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);

    volt1 = pSheet->VoltageAt(outLocx, outLocy);
    test1 = oLLAs[ outputIdx] ->f(volt1);
    err1 = fabs(1.0 - test1);

    if (bVerbose && outputFile)
        fprintf(outputFile, "Test1 (0 -> 0): f(%f) = %f, Error: %f\n",
        volt1, test1, err1);

    pSheet->ClearSheet();

    // test 1,0
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);

    volt2 = pSheet->VoltageAt(outLocx, outLocy);
    test2 = oLLAs[ outputIdx] ->f(volt2);
    err2 = fabs(0.0 - test2);

    if (bVerbose && outputFile)
        fprintf(outputFile, "Test2 (1 -> 0): f(%f) = %f, Error: %f\n",
        volt2, test2, err2);

    pSheet->ClearSheet();

    // test 0,1
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);

    volt3 = pSheet->VoltageAt(outLocx, outLocy);
    test3 = oLLAs[ outputIdx] ->f(volt3);

```

```

err3 = fabs(1.0 - test3);

if (bVerbose && outputFile)
    fprintf(outputFile, "Test3 (0 -> 1): f(%f) = %f, Error: %f\n",
volt3, test3, err3);

pSheet->ClearSheet();

// test 1,1
pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
pSheet->AddSource(inLoc2x, inLoc2y, 1.0);

volt4 = pSheet->VoltageAt(outLocx, outLocy);
test4 = oLLAs[outputIdx] ->f(volt4);
err4 = fabs(1.0 - test4);

if (bVerbose && outputFile)
    fprintf(outputFile, "Test4 (1 -> 1): f(%f) = %f, Error: %f\n",
volt4, test4, err4);

// clean up
delete pSheet;

for (int i = 0; i < SHEET_LOCATIONS; i++) {
    delete oLLAs[i];
}

// return the average error
return (err1 + err2 + err3 + err4); // / 4.0;
}

float CIMPEval::eval(int bVerbose) {
    return this->eval(bVerbose, stdout);
}

float CIMPEval::eval() {
    return this->eval(0, NULL);
}

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

    LLA.cpp
*****/

#include "LLA.h"

CLLA::CLLA(float val0, float valhalf, float vall) {
    m_val0 = val0;
    m_valhalf = valhalf;
    m_vall = vall;
}

```

```

CLLA::~CLLA() {
}

float CLLA::f(float x) {
    if (x == 0)
        return m_val0;
    else if (x > 0 && x < 0.5)
        return ((m_valhalf - m_val0) / 0.5) * x + m_val0;
    else if (x == 0.5)
        return m_valhalf;
    else if (x > 0.5 && x < 1.0) {
        float m = ((m_val1 - m_valhalf) / 0.5);
        return (m * (x - 0.5)) + m_valhalf;
    }
    else if (x == 1.0)
        return m_val1;

    else
        return -1.0;
}

```

```

/*****
    EVO_EAC
    Nathan Ainslie
    nainslie@indiana.edu
    Oct 2002

```

XOREval.cpp

```

*****/

```

```

#include "XOREval.h"

```

```

#include <math.h>

```

```

#define SHEET_RESISTANCE 1.00

```

```

CXOREval::CXOREval() {

```

```

}

```

```

CXOREval::CXOREval(float resistance) {
    m_resistance = resistance;

```

```

}

```

```

CXOREval::~CXOREval() {

```

```

}

```

```

int CXOREval::getNumInputs() {
    return 2;
}

```

```

}

```

```

int CXOREval::getNumOutputs() {
    return 1;
}

void CXOREval::setGenome(CGenome* pG) {
    m_pGenome = pG;
}

float CXOREval::eval(int bVerbose, FILE* outputFile) {
    int inputIdx1 = 0;
    int inputIdx2 = 0;
    int outputIdx = 0;
    int bFirstIndexFound = 0;
    int bOutputIndexFound = 0;
    float inLoc1x, inLoc1y;
    float inLoc2x, inLoc2y;
    float outLocx, outLocy;
    int i = 0;
    CConductiveSheet* pSheet;
    CLLA* oLLAs[ SHEET_LOCATIONS];
    float volt1, volt2, volt3, volt4;
    float test1, test2, test3, test4;
    float err1, err2, err3, err4;

    while(inputIdx2 == 0) {
        if(m_pGenome->inputCodons[ i] == 1) {
            if(bFirstIndexFound == 0) {
                inputIdx1 = i;
                bFirstIndexFound = 1;
            } else {
                inputIdx2 = i;
            }
        }
        i++;
    }

    i = 0;

    while (!(bOutputIndexFound)) {
        if(m_pGenome->outputCodons[ i] == 1) {
            outputIdx = i;
            bOutputIndexFound = 1;
        } else {
            i++;
        }
    }

    //fprintf(stderr, "Input idx1: %d, Input idx2: %d\n", inputIdx1,
inputIdx2);
    //fprintf(stderr, "Output idx: %d\n", outputIdx);

    for(i = 0; i < SHEET_LOCATIONS; i++) {
        float zeroVal, halfVal, oneVal;

        if(m_pGenome->llaCodons[ 3*i + 0] == LLA_ZERO_VALUE)
            zeroVal = 0.0f;
        if(m_pGenome->llaCodons[ 3*i + 0] == LLA_HALF_VALUE)

```

```

        zeroVal = 0.5f;
        if(m_pGenome->llaCodons[ 3*i + 0] == LLA_ONE_VALUE)
            zeroVal = 1.0f;

        if(m_pGenome->llaCodons[ 3*i + 1] == LLA_ZERO_VALUE)
            halfVal = 0.0f;
        if(m_pGenome->llaCodons[ 3*i + 1] == LLA_HALF_VALUE)
            halfVal = 0.5f;
        if(m_pGenome->llaCodons[ 3*i + 1] == LLA_ONE_VALUE)
            halfVal = 1.0f;

        if(m_pGenome->llaCodons[ 3*i + 2] == LLA_ZERO_VALUE)
            oneVal = 0.0f;
        if(m_pGenome->llaCodons[ 3*i + 2] == LLA_HALF_VALUE)
            oneVal = 0.5f;
        if(m_pGenome->llaCodons[ 3*i + 2] == LLA_ONE_VALUE)
            oneVal = 1.0f;

        oLLAs[ i] = new CLLA(zeroVal, halfVal, oneVal);
    }

    pSheet = new CConductiveSheet(m_resistance);

    inLoc1x = (inputIdx1 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
    inLoc1y = (inputIdx1 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

    inLoc2x = (inputIdx2 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
    inLoc2y = (inputIdx2 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

    outLocx = (outputIdx % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
    outLocy = (outputIdx / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

    // test 0,0
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);

    volt1 = pSheet->VoltageAt(outLocx, outLocy);
    test1 = oLLAs[ outputIdx] ->f(volt1);
    err1 = fabs(0.0 - test1);

    if (bVerbose && outputFile)
        fprintf(outputFile,"Test1 (0 XOR 0): f(%f) = %f, Error: %f\n",
        volt1, test1, err1);

    pSheet->ClearSheet();

    // test 1,0
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);

    volt2 = pSheet->VoltageAt(outLocx, outLocy);
    test2 = oLLAs[ outputIdx] ->f(volt2);
    err2 = fabs(1.0 - test2);

    if (bVerbose && outputFile)

```

```

    fprintf(outputFile, "Test2 (1 XOR 0): f(%f) = %f, Error: %f\n",
volt2, test2, err2);

    pSheet->ClearSheet();

    // test 0,1
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);

    volt3 = pSheet->VoltageAt(outLocx, outLocy);
    test3 = oLLAs[outputIdx] ->f(volt3);
    err3 = fabs(1.0 - test3);

    if (bVerbose && outputFile)
        fprintf(outputFile, "Test3 (0 XOR 1): f(%f) = %f, Error: %f\n",
volt3, test3, err3);

    pSheet->ClearSheet();

    // test 1,1
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);

    volt4 = pSheet->VoltageAt(outLocx, outLocy);
    test4 = oLLAs[outputIdx] ->f(volt4);
    err4 = fabs(0.0 - test4);

    if (bVerbose && outputFile)
        fprintf(outputFile, "Test4 (1 XOR 1): f(%f) = %f, Error: %f\n",
volt4, test4, err4);

    // clean up
    delete pSheet;

    for (int i = 0; i < SHEET_LOCATIONS; i++) {
        delete oLLAs[i];
    }

    // return the average error
    return (err1 + err2 + err3 + err4); // / 4.0;
}

float CXOREval::eval(int bVerbose) {
    return this->eval(bVerbose, stdout);
}
float CXOREval::eval() {
    return this->eval(0, NULL);
}

```

Appendix B (the changes done to the code used for evolving):

```

/*****
    EVO_EAC
    Raj, Kota, Vess, Radu, Phil

```

```

main.cpp

***** /

#include "ConductiveSheet.h"
#include "Genome.h"
#include "LLA.h"
#include "Eval.h"
#include "XOREval.h"
#include "ANDEval.h"
#include "IMPEval.h"
#include "CYCEval.h"
#include "RETEval.h"
#include "BreedingPop.h"
#include "God.h"

#include <stdio.h>
#include <string.h>

#define RES 2.0

int NUM_GENS = 100;
int GEN_SIZE = 10000;

int main(int argc, char* argv[]) {
    int i = 4;
    FILE* pOutputFile = NULL;
    CEval* pEval;
    if (argc >= 4) {
        NUM_GENS = strtol(argv[1], NULL, 0);
        GEN_SIZE = strtol(argv[2], NULL, 0);

        if (!(strcmp("AND", argv[3]))) {
            pEval = new CANDEval(RES);
        } else if (!(strcmp("XOR", argv[3]))) {
            pEval = new CXOREval(RES);
        } else if (!(strcmp("IMP", argv[3]))) {
            pEval = new CIMPEval(RES);
        } else if (!(strcmp("CYC", argv[3]))) {
            pEval = new CCYCEval(RES);
        } else if (!(strcmp("RET", argv[3]))) {
            pEval = new CRETEval(RES);
        }

    } else {
        fprintf(stderr, "Usage: %s num_generations generation_size function
[-f output file] [-m mutation_rate] [-v]\n", argv[0]);
        exit(0);
    }

    printf("Making God...\n");
    CGod oGod(NUM_GENS, GEN_SIZE);

    while(i < argc) {
        if(!(strcmp("-f", argv[i]))) {
            pOutputFile = fopen(argv[i+1], "w");

```

```

        oGod.setOutputFile(pOutputFile);
        i += 2;
    } else if (!(strcmp("-m", argv[ i ]))) {
        oGod.setMutationRate(strtod(argv[ i+1 ], NULL));
        i += 2;
    } else if (!(strcmp("-v", argv[ i ]))) {
        oGod.setVerbosity(1);
        i++;
    } else if (!(strcmp("-e", argv[ i ]))) {
        oGod.setMinErr(strtod(argv[ i+1 ], NULL));
        i +=2;
    } else {
        i++;
    }
}

printf("Setting evaluator...\n");
oGod.setEvaluator(pEval);
printf("In main...");
oGod.go();
printf("finished...\n");

delete pEval;
if (pOutputFile)
    fclose(pOutputFile);

return 0;
}

/*****
    EVO_EAC
    Raj, Kota, Vess, Radu, Phil

    CYCEval.cpp
*****/

#include "CYCEval.h"
#include <math.h>

#define SHEET_RESISTANCE 1.00

CCYCEval::CCYCEval() {
}

CCYCEval::CCYCEval(float resistance) {
    m_resistance = resistance;
}

CCYCEval::~~CCYCEval() {
}

int CCYCEval::getNumInputs() {

```

```

    return 4;
}

int CCYCEval::getNumOutputs() {
    return 4;
}

void CCYCEval::setGenome(CGGenome* pG) {
    m_pGenome = pG;
}

float CCYCEval::eval(int bVerbose, FILE* outputFile) {
    int inputIdx1 = 0;
    int inputIdx2 = 0;
    int inputIdx3 = 0;
    int inputIdx4 = 0;
    int outputIdx1 = 0;
    int outputIdx2 = 0;
    int outputIdx3 = 0;
    int outputIdx4 = 0;
    int bFirstIndexFound = 0;
    int bOutputIndexFound = 0;
    float inLoc1x, inLoc1y;
    float inLoc2x, inLoc2y;
    float inLoc3x, inLoc3y;
    float inLoc4x, inLoc4y;
    float outLoc1x, outLoc1y;
    float outLoc2x, outLoc2y;
    float outLoc3x, outLoc3y;
    float outLoc4x, outLoc4y;
    int i = 0;
    CConductiveSheet* pSheet;
    CLLA* oLLAs[ SHEET_LOCATIONS ];
    float volt1, volt2, volt3, volt4;
    float test1, test2, test3, test4;
    float err1, err2, err3, err4, toterr;

    toterr = 0;

    while(inputIdx4 == 0) {
        printf("%d\n", i);
        if(m_pGenome->inputCodons[ i ] == 1) {
            if(bFirstIndexFound == 0) {
                inputIdx1 = i;
                bFirstIndexFound = 1;
            } else if(bFirstIndexFound == 1) {
                inputIdx2 = i;
                bFirstIndexFound = 2;
            } else if(bFirstIndexFound == 2) {
                inputIdx3 = i;
                bFirstIndexFound = 3;
            } else {
                inputIdx4 = i;
            }
        }
        i++;
    }
}

```

```

i = 0;

while(outputIdx4 == 0) {
    if(m_pGenome->outputCodons[ i] == 1) {
        if(bOutputIndexFound == 0) {
            printf("Found 1 @ %d\n", i);
            outputIdx1 = i;
            bOutputIndexFound = 1;
        } else if(bOutputIndexFound == 1) {

            outputIdx2 = i;
            bOutputIndexFound = 2;
        } else if(bOutputIndexFound == 2) {

            outputIdx3 = i;
            bOutputIndexFound = 3;
        } else {
            outputIdx4 = i;

            bOutputIndexFound = 4;
        }
    }
}

i++;
}

//fprintf(stderr, "Input idx1: %d, Input idx2: %d\n", inputIdx1,
inputIdx2);
//fprintf(stderr, "Output idx: %d\n", outputIdx);

for(i = 0; i < SHEET_LOCATIONS; i++) {
    float zeroVal, halfVal, oneVal;

    if(m_pGenome->llaCodons[ 3*i + 0] == LLA_ZERO_VALUE)
        zeroVal = 0.0f;
    if(m_pGenome->llaCodons[ 3*i + 0] == LLA_HALF_VALUE)
        zeroVal = 0.5f;
    if(m_pGenome->llaCodons[ 3*i + 0] == LLA_ONE_VALUE)
        zeroVal = 1.0f;

    if(m_pGenome->llaCodons[ 3*i + 1] == LLA_ZERO_VALUE)
        halfVal = 0.0f;
    if(m_pGenome->llaCodons[ 3*i + 1] == LLA_HALF_VALUE)
        halfVal = 0.5f;
    if(m_pGenome->llaCodons[ 3*i + 1] == LLA_ONE_VALUE)
        halfVal = 1.0f;

    if(m_pGenome->llaCodons[ 3*i + 2] == LLA_ZERO_VALUE)
        oneVal = 0.0f;
    if(m_pGenome->llaCodons[ 3*i + 2] == LLA_HALF_VALUE)
        oneVal = 0.5f;
    if(m_pGenome->llaCodons[ 3*i + 2] == LLA_ONE_VALUE)
        oneVal = 1.0f;
}

```

```

    oLLAs[ i] = new CLLA(zeroVal, halfVal, oneVal);
}

pSheet = new CConductiveSheet(m_resistance);

inLoc1x = (inputIdx1 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
inLoc1y = (inputIdx1 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

inLoc2x = (inputIdx2 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
inLoc2y = (inputIdx2 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

inLoc3x = (inputIdx2 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
inLoc3y = (inputIdx2 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

inLoc4x = (inputIdx2 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
inLoc4y = (inputIdx2 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

outLoc1x = (outputIdx1 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
outLoc1y = (outputIdx1 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

outLoc2x = (outputIdx2 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
outLoc2y = (outputIdx2 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

outLoc3x = (outputIdx3 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
outLoc3y = (outputIdx3 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

outLoc4x = (outputIdx4 % (int) SHEET_ROWS_COLS) * m_pGenome->spacing;
outLoc4y = (outputIdx4 / (int) SHEET_ROWS_COLS) * m_pGenome->spacing;

// tests

printf("Starting tests\n");

for (i = 0; i < 16; i++){
    if (i == 0) {
        pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
        pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
        pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
        pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
    } else if (i == 1) {
        pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
        pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
        pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
        pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
    } else if (i == 2) {
        pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
        pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
        pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
        pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
    } else if (i == 3) {
        pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
        pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
        pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
        pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
    }
}

```

```

} else if (i == 4) {
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 5) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
} else if (i == 6) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
} else if (i == 7) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 8) {
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
} else if (i == 9) {
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 10) {
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 11) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 0.0);
} else if (i == 12) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 0.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 13) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 0.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 14) {
    pSheet->AddSource(inLoc1x, inLoc1y, 0.0);
    pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
    pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
    pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
} else if (i == 15) {
    pSheet->AddSource(inLoc1x, inLoc1y, 1.0);

```

```

        pSheet->AddSource(inLoc2x, inLoc2y, 1.0);
        pSheet->AddSource(inLoc3x, inLoc3y, 1.0);
        pSheet->AddSource(inLoc4x, inLoc4y, 1.0);
    }

    volt1 = pSheet->VoltageAt(outLoc1x, outLoc1y);
    volt2 = pSheet->VoltageAt(outLoc2x, outLoc2y);
    volt3 = pSheet->VoltageAt(outLoc3x, outLoc3y);
    volt4 = pSheet->VoltageAt(outLoc4x, outLoc4y);

    test1 = oLLAs[outputIdx1] ->f(volt1);
    test2 = oLLAs[outputIdx2] ->f(volt2);
    test3 = oLLAs[outputIdx3] ->f(volt3);
    test4 = oLLAs[outputIdx4] ->f(volt4);

    err1 = fabs(0.0 - test1);
    err2 = fabs(0.0 - test2);
    err3 = fabs(0.0 - test3);
    err4 = fabs(0.0 - test4);
    toterr += err1 + err2 + err3 + err4;

    if (bVerbose && outputFile)
        fprintf(outputFile, "Test %d:\n  output1(%f) = %f, Error: %f\n
output2(%f) = %f, Error: %f\n  output3(%f) = %f, Error: %f\n
output4(%f) = %f, Error: %f\n\n", i, volt1, test1, err1, volt2, test2,
err2, volt3, test3, err3, volt4, test4, err4);

    pSheet->ClearSheet();
}

// clean up
delete pSheet;

for (int i = 0; i < SHEET_LOCATIONS; i++) {
    delete oLLAs[i];
}

printf("The end\n");
// return the average error
return (toterr); //4.0
}

float CCYCEval::eval(int bVerbose) {
    return this->eval(bVerbose, stdout);
}

float CCYCEval::eval() {
    return this->eval(0, NULL);
}

```

Appendix C (output sample after running the code for cyclotron beam controller):

100 Gens 10000 Genomes

Test 8:
output1(0.072897) = 0.000000, Error: 0.000000
output2(0.003645) = 0.000000, Error: 0.000000
output3(0.005607) = 0.011215, Error: 0.011215
output4(0.002278) = 0.000000, Error: 0.000000

Test 9:
output1(0.072897) = 0.000000, Error: 0.000000
output2(0.003645) = 0.000000, Error: 0.000000
output3(0.005607) = 0.011215, Error: 0.011215
output4(0.002278) = 0.000000, Error: 0.000000

Test 10:
output1(0.072897) = 0.000000, Error: 0.000000
output2(0.003645) = 0.000000, Error: 0.000000
output3(0.005607) = 0.011215, Error: 0.011215
output4(0.002278) = 0.000000, Error: 0.000000

Test 11:
output1(0.080187) = 0.000000, Error: 0.000000
output2(0.008201) = 0.000000, Error: 0.000000
output3(0.009657) = 0.019315, Error: 0.019315
output4(0.004100) = 0.000000, Error: 0.000000

Test 12:
output1(0.080187) = 0.000000, Error: 0.000000
output2(0.008201) = 0.000000, Error: 0.000000
output3(0.009657) = 0.019315, Error: 0.019315
output4(0.004100) = 0.000000, Error: 0.000000

Test 13:
output1(0.080187) = 0.000000, Error: 0.000000
output2(0.008201) = 0.000000, Error: 0.000000
output3(0.009657) = 0.019315, Error: 0.019315
output4(0.004100) = 0.000000, Error: 0.000000

Test 14:
output1(0.109346) = 0.000000, Error: 0.000000
output2(0.005467) = 0.000000, Error: 0.000000
output3(0.008411) = 0.016822, Error: 0.016822
output4(0.003417) = 0.000000, Error: 0.000000

Test 15:
output1(0.116635) = 0.000000, Error: 0.000000
output2(0.010023) = 0.000000, Error: 0.000000
output3(0.012461) = 0.024922, Error: 0.024922
output4(0.005239) = 0.000000, Error: 0.000000

The end

1000 Generations 1000 Genomes

Spacing: 2.929233

Sheet Inputs: |0|0|0|0|0|0|0|0|0|1|0|0|1|0|1|0|0|0|0|0|0|1|0|0|0|0|0|0
 Sheet Outputs: |0|1|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|1
 LLA's:
 |16|0|0|0|8|16|8|8|16|0|8|0|0|16|8|16|16|0|16|16|16|8|8|8|0|8|8|0|0|0|
 16|0|0|16|16|8|0|0|0|16|8|0|0|8|8|0|16|0|8|0|8|16|0|8|16|0|0|16|0|0|8|
 0|16|0|0|16|8|0|16|0|8|16|0|8|16
 Error: 1.048744

Starting tests

Test 0:

output1(0.000000) = 0.000000, Error: 0.000000
 output2(0.000000) = 0.000000, Error: 0.000000
 output3(0.000000) = 0.000000, Error: 0.000000
 output4(0.000000) = 0.000000, Error: 0.000000

Test 1:

output1(0.014839) = 0.014839, Error: 0.014839
 output2(0.074195) = 0.000000, Error: 0.000000
 output3(0.008244) = 0.008244, Error: 0.008244
 output4(0.007419) = 0.007419, Error: 0.007419

Test 2:

output1(0.018549) = 0.018549, Error: 0.018549
 output2(0.007419) = 0.000000, Error: 0.000000
 output3(0.009274) = 0.009274, Error: 0.009274
 output4(0.005707) = 0.005707, Error: 0.005707

Test 3:

output1(0.018549) = 0.018549, Error: 0.018549
 output2(0.007419) = 0.000000, Error: 0.000000
 output3(0.009274) = 0.009274, Error: 0.009274
 output4(0.005707) = 0.005707, Error: 0.005707

Test 4:

output1(0.018549) = 0.018549, Error: 0.018549
 output2(0.007419) = 0.000000, Error: 0.000000
 output3(0.009274) = 0.009274, Error: 0.009274
 output4(0.005707) = 0.005707, Error: 0.005707

Test 5:

output1(0.033388) = 0.033388, Error: 0.033388
 output2(0.081614) = 0.000000, Error: 0.000000
 output3(0.017518) = 0.017518, Error: 0.017518
 output4(0.013127) = 0.013127, Error: 0.013127

Test 6:

output1(0.033388) = 0.033388, Error: 0.033388
 output2(0.081614) = 0.000000, Error: 0.000000
 output3(0.017518) = 0.017518, Error: 0.017518
 output4(0.013127) = 0.013127, Error: 0.013127

Test 7:

output1(0.033388) = 0.033388, Error: 0.033388
 output2(0.081614) = 0.000000, Error: 0.000000
 output3(0.017518) = 0.017518, Error: 0.017518
 output4(0.013127) = 0.013127, Error: 0.013127

Test 8:

output1(0.037097) = 0.037097, Error: 0.037097
output2(0.014839) = 0.000000, Error: 0.000000
output3(0.018549) = 0.018549, Error: 0.018549
output4(0.011415) = 0.011415, Error: 0.011415

Test 9:

output1(0.037097) = 0.037097, Error: 0.037097
output2(0.014839) = 0.000000, Error: 0.000000
output3(0.018549) = 0.018549, Error: 0.018549
output4(0.011415) = 0.011415, Error: 0.011415

Test 10:

output1(0.037097) = 0.037097, Error: 0.037097
output2(0.014839) = 0.000000, Error: 0.000000
output3(0.018549) = 0.018549, Error: 0.018549
output4(0.011415) = 0.011415, Error: 0.011415

Test 11:

output1(0.051936) = 0.051936, Error: 0.051936
output2(0.089034) = 0.000000, Error: 0.000000
output3(0.026792) = 0.026792, Error: 0.026792
output4(0.018834) = 0.018834, Error: 0.018834

Test 12:

output1(0.051936) = 0.051936, Error: 0.051936
output2(0.089034) = 0.000000, Error: 0.000000
output3(0.026792) = 0.026792, Error: 0.026792
output4(0.018834) = 0.018834, Error: 0.018834

Test 13:

output1(0.051936) = 0.051936, Error: 0.051936
output2(0.089034) = 0.000000, Error: 0.000000
output3(0.026792) = 0.026792, Error: 0.026792
output4(0.018834) = 0.018834, Error: 0.018834

Test 14:

output1(0.055646) = 0.055646, Error: 0.055646
output2(0.022258) = 0.000000, Error: 0.000000
output3(0.027823) = 0.027823, Error: 0.027823
output4(0.017122) = 0.017122, Error: 0.017122

Test 15:

output1(0.070485) = 0.070485, Error: 0.070485
output2(0.096453) = 0.000000, Error: 0.000000
output3(0.036067) = 0.036067, Error: 0.036067
output4(0.024541) = 0.024541, Error: 0.024541

The end

1000 Gens 10000 Genomes

Spacing: 3.932342

Sheet Inputs: |1|1|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|1|0|0|0|0|0|0|0|0|0|

Sheet Outputs: |0|0|0|0|1|0|0|0|1|0|0|0|0|1|0|0|0|0|0|0|0|0|0|0|1|0|0|0

LLA's:

|8|16|8|0|0|0|8|8|8|0|16|0|0|16|8|16|0|16|16|0|16|0|0|16|8|8|8|16|8|16|
|8|0|0|16|16|8|0|0|8|0|16|8|16|8|8|16|0|0|16|8|0|16|8|0|0|16|0|8|16|8|
0|16|8|0|0|16|16|16|8|0|8|8|16|16|0

Error: 0.260741

Starting tests

Test 0:

output1(0.000000) = 0.000000, Error: 0.000000
output2(0.000000) = 0.000000, Error: 0.000000
output3(0.000000) = 0.000000, Error: 0.000000
output4(0.000000) = 0.000000, Error: 0.000000

Test 1:

output1(0.002573) = 0.005146, Error: 0.005146
output2(0.008234) = 0.000000, Error: 0.000000
output3(0.005146) = 0.000000, Error: 0.000000
output4(0.002422) = 0.000000, Error: 0.000000

Test 2:

output1(0.004574) = 0.009149, Error: 0.009149
output2(0.020585) = 0.000000, Error: 0.000000
output3(0.008234) = 0.000000, Error: 0.000000
output4(0.002573) = 0.000000, Error: 0.000000

Test 3:

output1(0.004574) = 0.009149, Error: 0.009149
output2(0.020585) = 0.000000, Error: 0.000000
output3(0.008234) = 0.000000, Error: 0.000000
output4(0.002573) = 0.000000, Error: 0.000000

Test 4:

output1(0.004574) = 0.009149, Error: 0.009149
output2(0.020585) = 0.000000, Error: 0.000000
output3(0.008234) = 0.000000, Error: 0.000000
output4(0.002573) = 0.000000, Error: 0.000000

Test 5:

output1(0.007148) = 0.014295, Error: 0.014295
output2(0.028819) = 0.000000, Error: 0.000000
output3(0.013380) = 0.000000, Error: 0.000000
output4(0.004995) = 0.000000, Error: 0.000000

Test 6:

output1(0.007148) = 0.014295, Error: 0.014295
output2(0.028819) = 0.000000, Error: 0.000000
output3(0.013380) = 0.000000, Error: 0.000000
output4(0.004995) = 0.000000, Error: 0.000000

Test 7:

output1(0.007148) = 0.014295, Error: 0.014295
output2(0.028819) = 0.000000, Error: 0.000000
output3(0.013380) = 0.000000, Error: 0.000000
output4(0.004995) = 0.000000, Error: 0.000000

Test 8:

output1(0.009149) = 0.018298, Error: 0.018298

output2(0.041170) = 0.000000, Error: 0.000000
output3(0.016468) = 0.000000, Error: 0.000000
output4(0.005146) = 0.000000, Error: 0.000000

Test 9:

output1(0.009149) = 0.018298, Error: 0.018298
output2(0.041170) = 0.000000, Error: 0.000000
output3(0.016468) = 0.000000, Error: 0.000000
output4(0.005146) = 0.000000, Error: 0.000000

Test 10:

output1(0.009149) = 0.018298, Error: 0.018298
output2(0.041170) = 0.000000, Error: 0.000000
output3(0.016468) = 0.000000, Error: 0.000000
output4(0.005146) = 0.000000, Error: 0.000000

Test 11:

output1(0.011722) = 0.023444, Error: 0.023444
output2(0.049404) = 0.000000, Error: 0.000000
output3(0.021614) = 0.000000, Error: 0.000000
output4(0.007568) = 0.000000, Error: 0.000000

Test 12:

output1(0.011722) = 0.023444, Error: 0.023444
output2(0.049404) = 0.000000, Error: 0.000000
output3(0.021614) = 0.000000, Error: 0.000000
output4(0.007568) = 0.000000, Error: 0.000000

Test 13:

output1(0.011722) = 0.023444, Error: 0.023444
output2(0.049404) = 0.000000, Error: 0.000000
output3(0.021614) = 0.000000, Error: 0.000000
output4(0.007568) = 0.000000, Error: 0.000000

Test 14:

output1(0.013723) = 0.027446, Error: 0.027446
output2(0.061755) = 0.000000, Error: 0.000000
output3(0.024702) = 0.000000, Error: 0.000000
output4(0.007719) = 0.000000, Error: 0.000000

Test 15:

output1(0.016296) = 0.032593, Error: 0.032593
output2(0.069988) = 0.000000, Error: 0.000000
output3(0.029848) = 0.000000, Error: 0.000000
output4(0.010141) = 0.000000, Error: 0.000000

The end