Low-Cost Evolution in Materio

Robert Lenzie

MSc by Research

University of York

Electronic Engineering

October 2017

1 Abstract

This thesis describes a method of using a Low-Cost computer to program Low-Cost materials to perform a computation. The work demonstrates that an evolutionary algorithm running on a Raspberry Pi can exploit physical properties of graphene and sets of resistors to solve simple travelling salesman problems. The work goes on to investigate the use of the platform to evolve a simple electro-magnetic sensor to show the applicability of the platform to solving other problems which cannot be solved in any other way.

Contents

2 Authors Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

3 Acknowledgements

Julian Miller who encouraged me to start this MSc and was an endless source of ideas and inspiration as my tutor for my first year

Kester Clegg, who lent me a PCI-6259 board and headers, provided me with some helpful IO code samples and provided a welcoming ear and very practical sounding board for ideas and approaches.

Martin Trefzer who was my second-year tutor who patiently guided and encouraged me though what seemed at times like an impossible task of getting the Raspberry Pi to work with the Switches, DACs and ADC.

Pete Turner who produced the Raspberry Pi board design and suggested very practical enhancements on my initial outline design.

Iain Will who provided key insights in how to stabilise the performance of the graphene being used.

4 Introduction

Evolution in Materio EiM is a relatively new area of research. It involves developing an artificial evolution software algorithm, which programs physical materials in some way, such that the material can produce a computational solution to a particular problem. In doing so the algorithm may exploit unknown or undocumented properties of the material being used.

Artificial Evolution includes Genetic Algorithms GA and Evolutionary Algorithms EA. A Genetic Algorithm is a Darwinian based general purpose search technique. A population of individuals are optimized by selecting the ones best capable of solving the problem and enabling their genes to propagate to later generations (1) (2) (3).

A GA starts with a population of random individuals, where every individual encodes a potential solution. The encoding is referred to as a genotype which contains one or more chromosomes. A simple chromosome would be a string of 1s and 0s and or floating-point numbers. The chromosome represents a solution to the problem.

The genotype for every individual, is decoded into its phenotype, which is the individual's physical manifestation represented by its genotype and that phenotype is tested in some way to determine its fitness score. The set of individuals' genotypes, created when the GA starts is called a generation. When all the individuals of a generation have been assigned a fitness score, the one or more best suited with the highest fitness score, survive and pass on their genes to the following generation. Some implementations use elitism, where the fittest individuals from the generation, survive unchanged and pass on their genes to the following generation. Elitism ensures the best solutions are retained across the generations.

To create the next generation a form of sexual reproduction can be used, called crossover, where sections of the chromosome from two high scoring members of a previous generation are combined to form a new member. Also, mutation occurs where the new member is mutated - where a bit or bits in its chromosome may be flipped or floating-point number(s) changed using a gaussian function.

Genetic Algorithms can use both mutation and crossover. In the current work the Evolutionary Algorithm EA described herein uses elitism and mutation only.

The author finds EiM very interesting because in applying an EA to physical matter, evolution needs no knowledge of the problem that is being attempted to be solved. This potentially allows the evolution of solutions to problems which cannot currently be easily solved in any other way.

The first modern demonstration of EiM was done by Adrian Thompson (4) (5). He used an GA which exploited the physical properties of a Xilinx XC6200 FPGA to evolve a simple tone discriminator and a robot controller, constructed by evolution out of components within the FPGA. Julian Miller (6) suggested using a liquid crystal display as an evolable medium, subsequently Simon Harding (7) (8) (9) (10) used an EA which exploited physical properties of a liquid crystal display to derive simple analogue computation. This included reproducing Thompson's tone discriminator and robot controller, this time using only a modified liquid crystal display as the material. More recently Kester Clegg (11) used an EA which exploited properties of carbon nanotubes to solve travelling salesman problems TSP. Maktuba Mohid (12) used an EA and carbon nanotubes to evolve a tone discriminator and in (13) she used an EA and carbon nanotubes to do classifications. Similarly Eléonore Vissol-Gaudin (14) (15) (16) (17) used carbon nanotube / liquid crystal composites to do classifications and evolve electronic circuits.

All of these researchers used a PC, Matlab or Borland Dephi and some sophisticated signal processing hardware and rare or difficult to obtain and configure materials to derive their results.

The existing work recreates the TSP work in (11) using very Low-Cost hardware, readily available Low-Cost materials and is written in Python on the Pycharm Community Edition Open Source platform (18). The work then goes on to explore the possibility of using this platform to attempt other computational tasks.

5 Aims and Architecture

5.1 Hypothesis

5.1.1 Is Low-Cost Hardware Evolution in Materio Possible?

At the start of this MSc it was unknown if it was possible to do EiM using hardware and materials costing less than £200. This thesis shows that it is entirely possible to do EiM research on a very low budget and hopefully this work will help enable the possibility of EiM research to a much wider community.

5.2 Aims

The 3 aims for this MSc were:

1. Identify, interface to and utilise suitable, Low-Cost, readily available EiM material and exploit this material to re-create previous EiM experiments on a PC
2. Research and create a Low-Cost EiM platform, using Low-Cost materials to re-create previous EiM experiments on this Low-Cost EiM platform
3. Analyse the Low-Cost EiM platform and attempt to use it for new experiments

5.3 Re-creating Earlier EiM work with the Travelling Salesman Problem

Work started attempting to re-create Kester Clegg's experiments with the TSP (11), but using much lower cost and more readily available materials. Once lower cost materials had been found and tested on a PC environment a suitable lower cost platform was researched, developed and exploited to reproduce the TSP work on the Low-Cost platform, using Low-Cost materials.

5.4 Architecture

Kester's TSP architecture (11) is summarised in Figure 1 Kester's Architecture. Matlab EA Code runs on the PC and communicates with the National Instruments PCI-6259 multi-function IO device. The PCI-6259 board has 48 Digital Outputs, 4x16bit Analogue Outputs, and 32x16bit Analogue Inputs. In this environment the PCI-6259 had three functions.

3 of the PCI-6259 Digital Outputs were used to configure the AD75019 (19) 16 x 16 cross-point switch.

4 Analogue Outs were available to provide configuration voltages to the material

12 of the 32 Inputs were used to read the voltages coming out of the remaining 8 or 12 pins of the material (depending on if the material had 12 or 16 pins), which represented a possible solution to the travelling salesman problem, for a cross-point switch and configuration voltage montage.

The reason for including the cross-point switch in the architecture was to permit the EA a wider search, whereby a configuration voltage could be routed to any input pin on the material and similarly any output pin from the material could route to any of the available PCI-6259 12x16bit Analogue Inputs ports in use.

A configuration voltage is a voltage sent to the material to change its electrical properties and so perform some form of configuration function. Sending one or more configuration voltages to an EiM material changes the materials' electrical properties and thereby among other things, causes changes to the output voltages of other pins on the material.

In summary this architecture permits the EA to route any of 1 to 4 Config Voltages to any one of the 16 pins on the Material and to read any of the remaining pins on the Material for a potential optimal solution to the TSP Problem being solved.

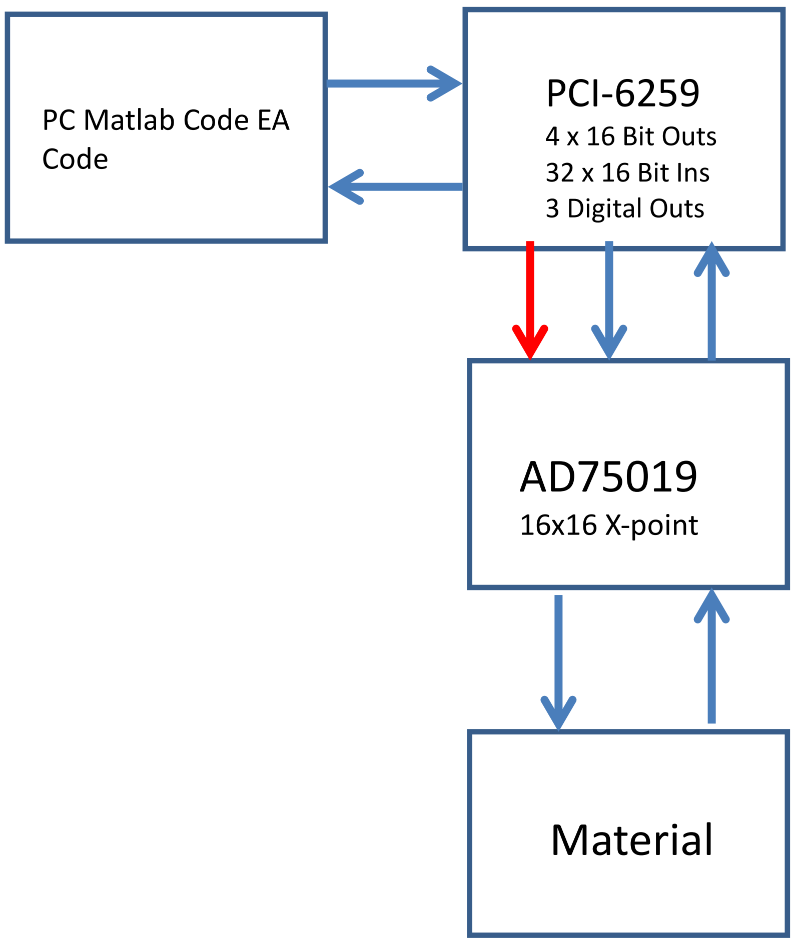


Figure 1 Kester's Architecture

The EA code runs to evolve an Optimal Solution to the Travelling Salesman Problem TSP. It uses TSP 1+4 i.e. it carries forward the elite of the previous generation and creates 4 new members for each generation, by mutating the carried forward elite. Each genome is evaluated by (1) configuring the Cross-point Switch, (2) sending configuration voltages to up to 4 pins on the material and (3) reading the voltages of from the remaining pins on the material the *signal pins*, which represent a potential optimal solution to the TSP.

The *signal pins* are read in sequence and the solution thus represented is costed by calculating the distance travelled by traversing from one pin to the next, with the pins logically arranged on the circumference of a circle.

6 Platform 1: High-Cost PC Based EiM Platform

6.1 Robust Modular Portable Platform.

It was necessary to have a development system that was modular, robust and portable so that it could be moved from the home lab of the author to the Electronics Lab in the University of York some 180 miles away, without having to worry about damage to the board, wires getting disconnected in use, in transit or in setting up or taking down.

6.2 Printed Circuit Boards

A small number of printed circuit boards were required to hold the material, a cross-point switch and some external connectors so that the board, the material and the cross-point switch each could be easily replaced if required.

A suitable circuit board was designed shown in Figure 2 PC PCB Schematic & Figure 3 PC PCB Images, complete with a 16 pin DIP socket for the material, a Cross-point switch socket, two diodes and 3 capacitors, required by the AD75019 and some termination blocks for connecting control and measurement voltages to and from the PCB. It was decided to place the board and the National Instrument header cards in a customised B&Q toolkit box which provided both a secure and portable home for the circuit board and NI Headers, and which permitted these to be simply connected by NI supplied cables to the PCI Board on the host PC.

6.2.1 Simple Interface to Possible Materials

In designing the circuit board, it was necessary to decide how any material used would be securely and safely interfaced to the board. A 16pin dip header socket was chosen as this made sense to use, and would facilitate quick, easy, reliable and secure connection and reconnection of any materials to the board with at worst a bit of soldering.

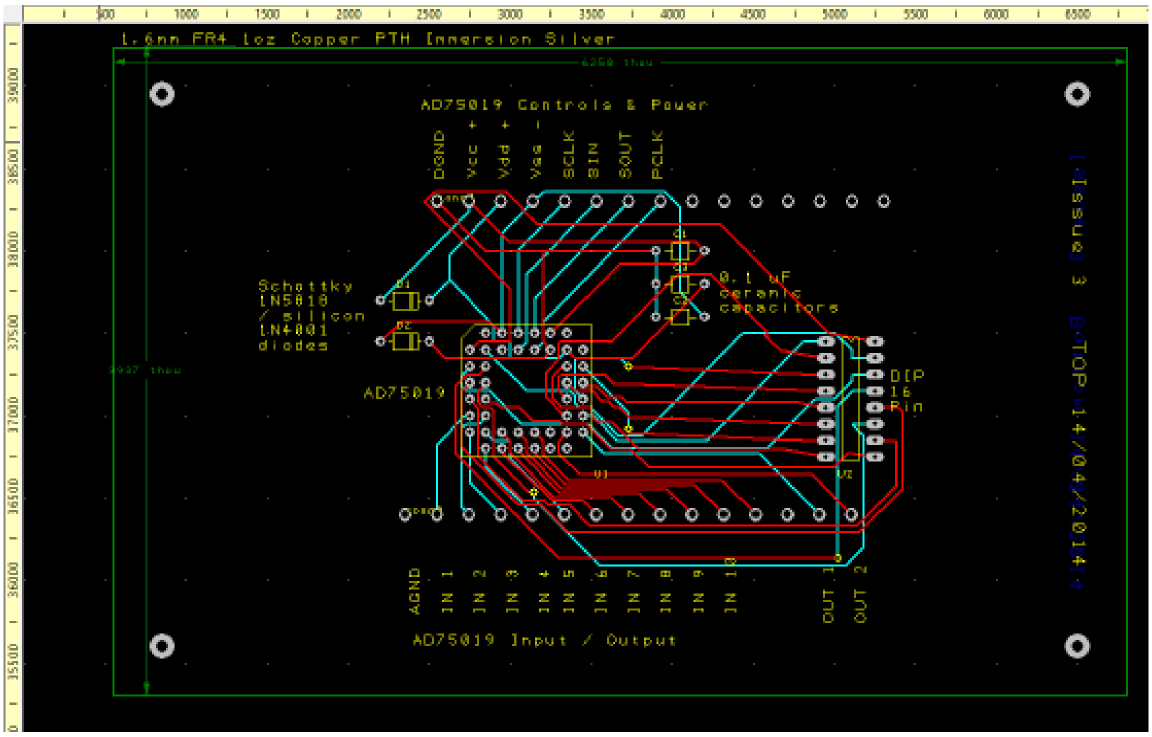


Figure 2 PC PCB Schematic

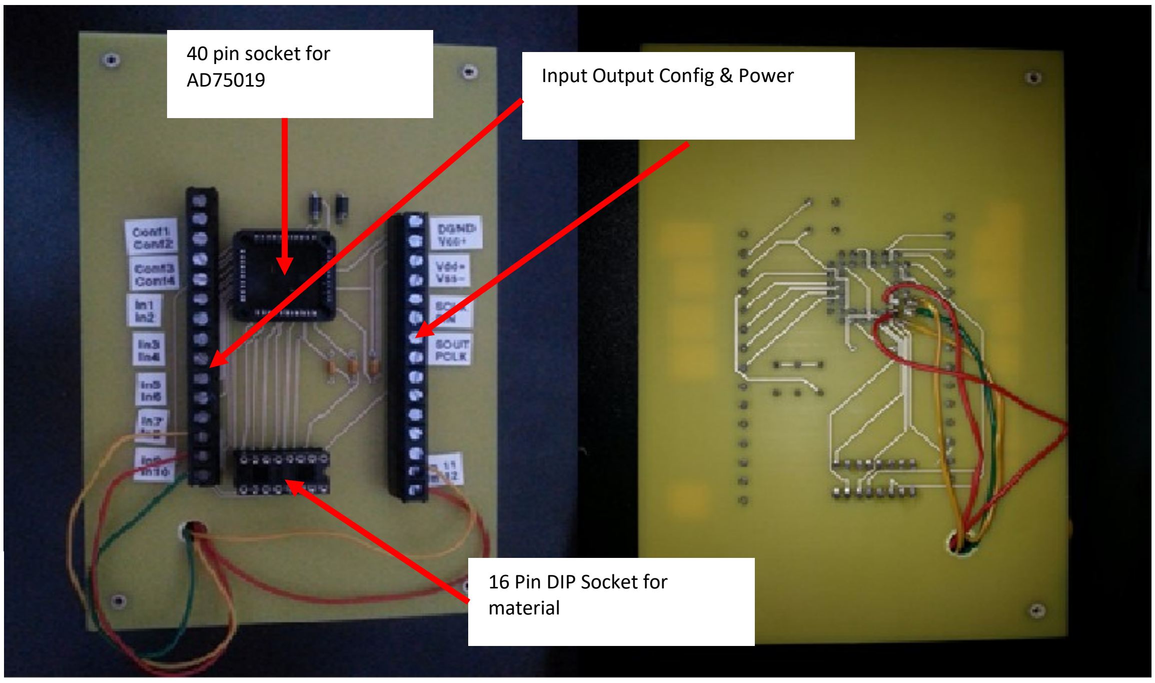
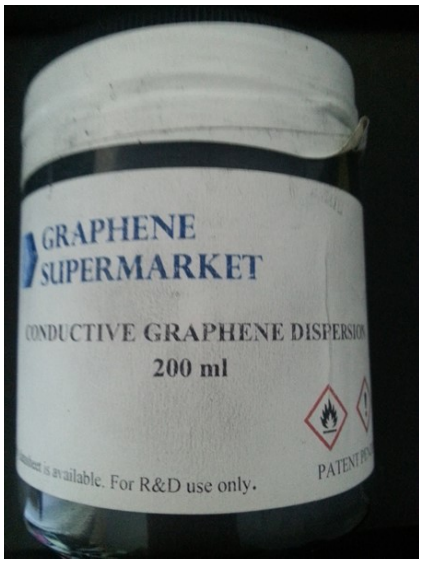

40 pin socket for AD75019
Input Output Config & Power
16 Pin DIP Socket for material

Figure 3 PC PCB Images

6.3 Choice of Graphene as Evolvable Material

Having designed and procured some boards it was necessary to find and discover how to use some material for the experiments. Kester Clegg (11) and Maktuba Mohid (12) had used carbon nanotubes and Eléonore Vissol-Gaudin (14) (15) (16) used carbon nanotube / liquid crystal composites which had been processed and manufactured in a Durham

University cleanroom. This project did not have access to those materials or a cleanroom, or the knowledge or skill to fabricate such material. A determined internet search for some other suitable material led to a US company Graphene Supermarket[1] who sell graphene suspended in a solvent solution, see Figure 4. The problem was then how to get 16 pins to connect securely and reliably into this Graphene?


GRAPHENE
SUPERMARKET
CONDUCTIVE GRAPHENE DISPERSION
200 ml
is available. For R&D use only.
PATENT
Figure 4 Tub of Graphene in Solution

6.4 Interfacing the Evolvable Material to the Circuit Board

Further internet search work led to a 16 pin DIP Header[2], which conveniently fitted snugly into the 16 pin DIP Socket on the circuit board. With some trial and error, it was discovered that the graphene would easily form connections to each pin by bending each pin 180 degrees down so that it faced into the "reservoir" in the header. Then using an eye dropper, some graphene solution was dispensed into the "reservoir" in the header. The graphene solution was then left overnight for the solvent to evaporate and a quick check of the resistance between each pin using a multi-meter, showed that there was a good circuit between each pin – see Figure 5 How to Make a DIP Graphene Header . This provided some very Low-Cost material to start solving TSP's with.

[1] https://graphene-supermarket.com/Graphene-Flakes-in-Solutions
[2] http://www.arieselec.com/products/data/12032-dip-header.htm

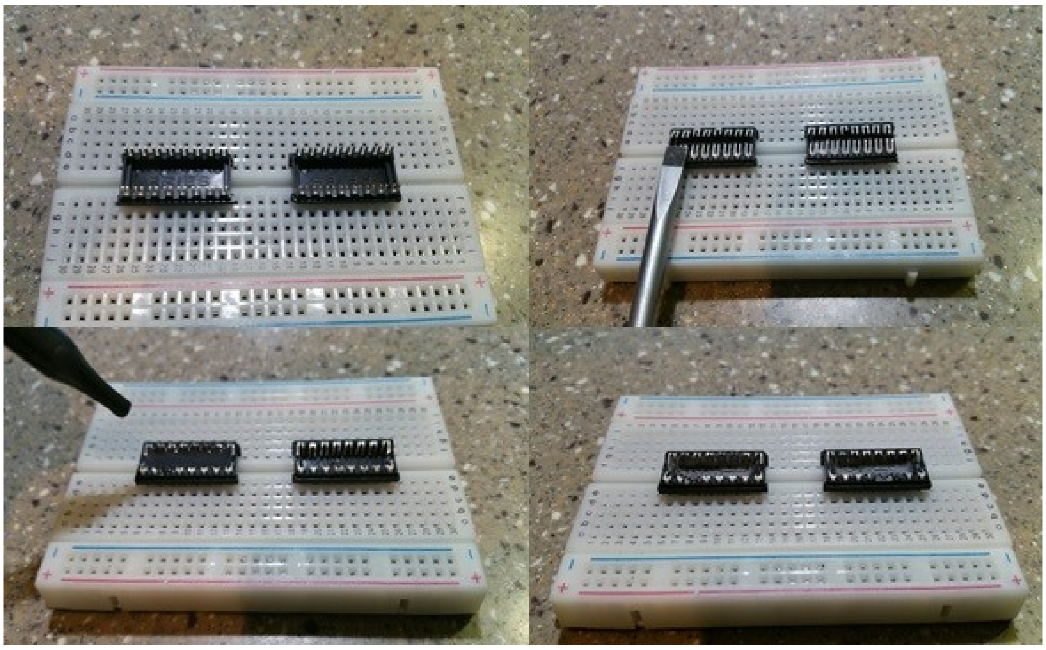


Figure 5 How to Make a DIP Graphene Header

6.5 Software Framework on the PC.

There were several tasks and coding functions which had to be researched and developed before an EA could be written to re-create previous EiM experiments.

- Install and test the National Instruments PCI-6259 board on the development PC
- Install Matlab and gain basic familiarity with the IDE
- Research and develop Matlab code to manage and configure the AD75019 cross-point switch
- Research and develop Matlab code to generate and send configuration voltages to the material and read voltages from the material
- Install and test the National Instruments PCI-6259 board on the development PC

6.5.1 Matlab Software & National Instruments PCI-6259 Installed on PC.

Use of Matlab as the IDE for the project was suggested so a license was purchased, the code downloaded, and some familiarity gained with the Matlab language and the IDE environment. This inevitably took some time. As well as learning a whole new IDE environment it was also necessary to discover how to interface the Matlab code to the physical hardware on the PC and the EIM board. A National Instruments PCI-6259 board had been acquired and installed on the development PC and interfaced to EIM board containing an AD75019 16x16 Cross-point Analogue Switch and evolvable material.

6.5.2 Programming the AD75029

The world of hardware specs and documentation was a completely alien world to the author, whose background is in mainframe database software. To gain an understanding required reading and re-reading of detailed text over and over many times to grasp what the arcane hardware language was attempting to convey.

For example, the AD75019 cross-point switch doc (19) has these terse words of guidance:

"APPLICATIONS INFORMATION Loading Data to control the switches is clocked serially into a 256-bit shift register and then transferred in parallel to 256 bits of memory. The rising edge of SCLK, the serial clock input, loads data into the shift register. The first bit loaded via SIN, the serial data input, controls the switch at the intersection of row Y15 and column X15. The next bits control the remaining columns (down to X0) of row Y15, and are followed by the bits for row Y14, and so on down to the data for the switch at the intersection of row Y0 and column X0. The shift register is dynamic, so there is a minimum clock rate, specified as 20 kHz. After the shift register is filled with the new 256 bits of control data, PCLK is activated (pulsed low) to transfer the data to the parallel latches. Since the shift register is dynamic, there is a maximum time delay specified before the data is lost: PCLK must be activated and brought back high within 5ms after filling the shift register. The switch control latches are static and will hold their data as long as power is applied."

This was eventually interpreted to mean the device is logically represented as follows:

```
    0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
0  Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15 Y15
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
1  Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14 Y14
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
2  Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13 Y13
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
3  Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12 Y12
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
4  Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11 Y11
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
5  Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10 Y10
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
6  Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09 Y09
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
7  Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08 Y08
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
8  Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07 Y07
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
9  Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06 Y06
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
10 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05 Y05
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
11 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04 Y04
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
12 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03 Y03
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
13 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02 Y02
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
14 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01 Y01
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
15 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00 Y00
   X15 X14 X13 X12 X11 X10 X09 X08 X07 X06 X05 X04 X03 X02 X01 X00
```

The following says all switches are open:

```
     0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15

0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
2    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
4    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
5    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
6    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
7    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
8    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
9    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
10   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
11   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
12   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
13   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
14   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
15   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

The following says X0-Y0, X1-Y1, X2-Y2, X3-Y3, X4-Y4, X5-Y5, X6-Y6, X7-Y7 are closed:

```
     0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15

0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
1    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
2    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
3    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
4    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
5    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0
6    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
7    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
8    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
9    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
10   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
11   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
12   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
13   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
14   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
15   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

To set the AD75019 these values are read in reverse order (BACKWARDS) from the above array i.e. from slot 255 through to slot 0 and are presented to pin SIN in this reverse order as described in the AD75019 doc[3]: "*The first bit loaded via SIN, the serial data input, controls the switch at the intersection of row Y15 and column X15. The next bits control the remaining columns (down to X0) of row Y15, and are followed by the bits for rowY14, and so on down to the data for the switch at the intersection of row Y0 and column X0.*"

[3] http://www.analog.com/media/en/technical-documentation/data-sheets/AD75019.pdf

Maintaining the data in the array in as-is order makes the structure much easier to understand and debug where necessary.

6.5.3 Using the Evolutionary Algorithm to Configure the Cross-Point Switch:

It was eventually found that loading a config into the AD75019 is done as follows:

(1) The config data to be loaded is stored in an array AD75019_Array in as-is order.

(2) Pin PCLK is set High

Then for each of the above 256 bits of data (the data is presented to SIN in reverse order), the following is done:

(3) Pin SIN is set to the Current Bit

(4) Pin SCLK is set from Low to High

(5) Pin SIN is held at current value for 40nano seconds.

(6) Pin SCLK is set from High to Low

Once all 256 bits have been sent:

(7) Pin PCLK is set Low within 5ms of the last bit being sent.

How would this be coded this in MATLAB ?

6.5.4 Mapping the AD75019 Cross-Point Switch to Matlab Matrices

It was necessary to understand how the 16x16 array maps to a Matlab matrix and how to get Matlab to copy the 256 bits of config data (the data is presented to SIN in reverse order), to the AD75019 cross-point switch.

The following says X0-Y0, X1-Y1, X2-Y2, X3-Y3, X4-Y4, X5-Y5, X6-Y6, X7-Y7 are Closed.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

However, attempting to read this off as is in reverse order would not work because Matlab *unwinds* the matrix Column by Column, so first we need to transpose csp_mat to be t_csp_mat

```
csp_mat =zeros(16)
csp_mat (1,1)=1
csp_mat (2,2)=1
csp_mat (3,3)=1
csp_mat (4,4)=1
csp_mat (5,5)=1
csp_mat (6,6)=1
csp_mat (7,7)=1
csp_mat (8,8)=1
csp_mat (9,9)=1
csp_mat (10,10)=1
csp_mat (11,11)=1
csp_mat (12,12)=1
csp_mat (13,13)=1
csp_mat (14,14)=1
csp_mat (15,15)=1
csp_mat (16,15)=1
csp_mat (16,16)=1
csp_mat =

     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1
```

Transposing this gives:

```
t_ csp_mat = csp_mat'
```

```
1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    1    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    1    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    1    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    1    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    1    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    1    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    1    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    1    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    1    0    0
0    0    0    0    0    0    0    0    0    0    0    0    0    0    1    1
0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    1
```

t_ csp_mat can now be read off

```
for t = 256: -1: 1
    %set initial config
    s.queueOutputData([1 0 switchConfig(t)]);
    %output one bit of config data
    s.queueOutputData([1 1 switchConfig(t)]);
end
```

6.5.5 Matlab function setUpSwitch to Configure the AD75019 Cross-point Switch

The complete setUpSwitch Matlab function to configure a Cross-point switch using a passed matrix of switchConfig, which contains the config required to be loaded:

```
%%%%%%%%%%%%%%%% SWITCH CONFIGURATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First part of this script configures 16x16 cross point switch
using
% digital I/O from NI DAQ card.  Second part loads the voltage
% configuration and records output.
% Parameters:   switchConfig is 256 bit stream to configure 16x16
switch array

function setUpSwitch(switchConfig)

%start session for DAQ
s =  ('ni');s = daq.createSession ('ni');

%%%%%%%%%%%%%%%%% DIGITAL I/O TO CONFIGURE SWITCHES
%%%%%%%%%%%%%%%%%%%%%%%
% Configure the digital IO rate (min 20Hz)
sclkFreq = 50e3;
s.Rate = sclkFreq;

% create 3 digital output channels: PCLK SCLK SIN
s.addDigitalChannel('dev1', 'Port0/Line0:2', 'OutputOnly');
% Add a dummy analog channel (so we can use its clock)
s.addAnalogInputChannel('Dev1',0,'Voltage');
```

```
% set PLCK high before transfer
s.queueOutputData([1 0 0]);
for t = 256: -1: 1
    %set initial config
    s.queueOutputData([1 0 switchConfig(t)]);
    %output one bit of config data
    s.queueOutputData([1 1 switchConfig(t)]);
end
% pulse PLCK low to transfer
s.queueOutputData([0 0 0]);
s.queueOutputData([1 0 0]);

% Output the queued data at SclkFreq rate
s.startForeground;

%switch is now configured.  We can remove our digital IO lines
s.removeChannel(1:4);
s.release
%%%%%%%%%%%%%%%% END OF SWITCH CONNECTIONS CONFIGURATION
%%%%%%%%%%%%%%%%%%
```

6.5.6 Using Matlab to send Control Voltages to and Read Voltages from Material

Controlling PCI-6259 to send configuration voltages and read the remaining pins of the material was done as in the example code below. Note the following:

The code takes run time parameters **noChannels** and **measuredelapsedtime** which says how many channels to measure voltage for and for how long to measure

The use of the Volts matrix which is loaded from **volts.txt** and will have been setup by the EA – this determines the config voltages sent to the material

Data is read from the number of channels configured with the config voltages in place for the duration required

```
%%%%%% Analogue Output Config to produce Inputs to Switch Array
%%%%%%%
% Setup the Analogue Outputs from NI PCI-6259 (Inputs to Switch
Array)
% Input File :  Volts.txt contains the voltage for each of the
Analogue
%               outputs from the PCI-6259 (Inputs to Switch Array)
% Output File:  capturedData.txt contains the output capturedData
for the
%               6 Analogue Inputs (Outputs from Switch Array)
```

```
function setUpAnalogueIOs(noChannels,measurelapsedtime)

%Create an NI session object
s = daq.createSession('ni');
%add 4 analog output 'Voltage' channels on dev1
ch1 = addAnalogOutputChannel(s,'dev1', 'ao0', 'Voltage');
ch2 = addAnalogOutputChannel(s,'dev1', 'ao1', 'Voltage');
ch3 = addAnalogOutputChannel(s,'dev1', 'ao2', 'Voltage');
ch4 = addAnalogOutputChannel(s,'dev1', 'ao3', 'Voltage');

% Add up to 12 analog input 'Voltage' channels on dev1
% Depending on contents of noChannels
switch noChannels
    case 6
        in1 = addAnalogInputChannel(s,'dev1', 'ai0', 'Voltage');
        in1.InputType='SingleEnded';
        in2 = addAnalogInputChannel(s,'dev1', 'ai1', 'Voltage');
        in2.InputType='SingleEnded';
        in3 = addAnalogInputChannel(s,'dev1', 'ai2', 'Voltage');
        in3.InputType='SingleEnded';
        in4 = addAnalogInputChannel(s,'dev1', 'ai3', 'Voltage');
        in4.InputType='SingleEnded';
        in5 = addAnalogInputChannel(s,'dev1', 'ai4', 'Voltage');
        in5.InputType='SingleEnded';
        in6 = addAnalogInputChannel(s,'dev1', 'ai5', 'Voltage');
        in6.InputType='SingleEnded';
    case 7…
    …
    case 12
        in1 = addAnalogInputChannel(s,'dev1', 'ai0', 'Voltage');
        in1.InputType='SingleEnded';
        in2 = addAnalogInputChannel(s,'dev1', 'ai1', 'Voltage');
        in2.InputType='SingleEnded';
        in3 = addAnalogInputChannel(s,'dev1', 'ai2', 'Voltage');
        in3.InputType='SingleEnded';
        in4 = addAnalogInputChannel(s,'dev1', 'ai3', 'Voltage');
        in4.InputType='SingleEnded';
        in5 = addAnalogInputChannel(s,'dev1', 'ai4', 'Voltage');
        in5.InputType='SingleEnded';
        in6 = addAnalogInputChannel(s,'dev1', 'ai5', 'Voltage');
        in6.InputType='SingleEnded';
        in7 = addAnalogInputChannel(s,'dev1', 'ai6', 'Voltage');
        in7.InputType='SingleEnded';
        in8 = addAnalogInputChannel(s,'dev1', 'ai7', 'Voltage');
        in8.InputType='SingleEnded';
        in9 = addAnalogInputChannel(s,'dev1', 'ai8', 'Voltage');
        in9.InputType='SingleEnded';
        in10 = addAnalogInputChannel(s,'dev1', 'ai9', 'Voltage');
        in10.InputType='SingleEnded';
        in11 = addAnalogInputChannel(s,'dev1', 'ai10', 'Voltage');
        in11.InputType='SingleEnded';
        in12 = addAnalogInputChannel(s,'dev1', 'ai11', 'Voltage');
        in12.InputType='SingleEnded';
end

%Set Voltage Range for each channel
ch1.Range=[-5.0,5.0];
ch2.Range=[-5.0,5.0];
ch3.Range=[-5.0,5.0];
```

```
ch4.Range=[-5.0,5.0];

%Load the voltages for the Analogue Channels from volts.txt
%into array volts
load volts.txt

%Set the Voltage for each Channel using data from array volts
ao0_volts=volts(1);
ao1_volts=volts(2);
ao2_volts=volts(3);
ao3_volts=volts(4);

%Create one set of data to output for each channel:
outputData(:,1) = linspace(ao0_volts, ao0_volts,
measurelapsedtime)';
outputData(:,2) = linspace(ao1_volts, ao1_volts,
measurelapsedtime)';
outputData(:,3) = linspace(ao2_volts, ao2_volts,
measurelapsedtime)';
outputData(:,4) = linspace(ao3_volts, ao3_volts,
measurelapsedtime)';

%Queue the output data:
queueOutputData(s,outputData);

%Get the duration for reference
duration = s.DurationInSeconds;

%Collect the Data:
[capturedData,time]=startForeground(s);

%Write capturedData to capturedData.txt
save capturedData.txt -ascii capturedData

%Unreserve the NI PCI-6259 DAQ so it can be used again
release(s)

end
```

6.6 Implementation of the Evolutionary Algorithm EA

Having got the major IO components working:

The AD75019 cross-point switch could be configured, some potentially evolvable material had been found, Configuration Voltages in the form of Analog Outs could be generated and Analogue Inputs – representing Solutions to the TSP was working.

6.6.1 Logging the Evolutionary Algorithm Progress for Debugging and Analysis

For diagnostics, debugging and analysis the following columns needed to store be stored as a minimum to identify and characterise every individual generated by the EA:

DeviceNo	Integer
RunNo	Integer
GenNo	Integer
GenMemNo	Integer - 1+4 - values in range 1 to 5, one for the elite and 4 for mutations.
Score	Float – Cost of Traversing the list of Cities
VoltsActive	Integer – Number of Configuration Voltages
Volt1Active	Bool
Volt2Active	Bool
Volt3Active	Bool
Volt4Active	Bool
ConfigVolts	Varchar(64) – 4 x 16bits for each Configuration Voltage
Csp_mat	Varchar(256) – 256 x 1bit for each switch in the cross-point array
Analogue_Inputs	Varchar(192) - 12 x16bits for each Non Config Voltage read

The above allows representation of the following in the genome/database:

The 256 bit Cross-point Array

The voltage on each of the 4 PCI_6259 Analogue Outs which are configuration voltages sent to the evolvable material.

The voltage of each of the 12 PCI_6259 Analogue Ins which represent solutions to the TSP.

The EA Needs to operate with the following constraints:

1. There is only ever a 1 to 1 relationship between the *x* pins and the *y* pins on the Cross-point switch
2. Only one Voltage and one Connection are randomly selected to be mutated for each Offspring
3. Voltages are mutated by Gaussian Mutation, which means using the Normal Distribution to generate the floating-point numbers
4. A single PCI_6259 Voltage will only go to a single electrode pin on the Material.
5. The next generation consists of 1+4[4], i.e. the Elite from the earlier generation is carried forward as is and is also mutated 4 times to produce the subsequent generation of 5 members.

If a mutation's score is as good as the previous Elite's that mutation parents the next generation.

A run is limited to 1500 generations (about 14 hours elapsed).

1. is the most difficult coding constraint to enforce. It was important to ensure that when the Cross-point Matrix was mutated, this NEVER resulted in more than 1 *x* connected to more than 1 *y* and vice versa – i.e. there was only ever a 1 to 1 relationship between the *x* pins and the *y* pins on the Cross-point switch.

[4] I asked Julian Miller where the idea for TSP 1+4 algorithm came from. He said it was developed with Cartesian Genetic Programming CGP – Bio Inspired Computation. He tried several different approaches and TSP 1+3, 1+4, 1+5 etc and 1+4 seemed the most effective from a CGP standpoint. Its use in TSP came about because of a student looking for things to apply CGP to.

6.6.2 Observation of Voltage Artifacts

Before coding the EA some analysis was done to discover what happens when voltages are applied to the material. The cross-point switch was setup in Matlab using the following:

```
csp_mat=eye(16)

l_csp_mat=logical(csp_mat)

t_csp_mat = l_csp_mat'
```

& then the following command was issued:

```
setUpSwitch(t_csp_mat)
```

which sets up cross-point switch to have the following connected x1-y1, x2-y2…x16-y16. Then some voltages were added to `volts.txt`:

```
   3.0000000e+00  -2.9992676e+00   3.0000000e+00   -3.0000000e+00
```

& the following command was issued:

```
setUpAnalogueIOs(6,4000)
```

setUpAnalogueIOs writes `capturedData.txt` from 6 signal lines containing 4 seconds of data. There appears to be mains hum in every signal line, so to eliminate that a 200 bar moving averages of the signal data was taken and plotted to see the graph in Figure 6 x-axis shows time in ms y-axis shows voltage from material:

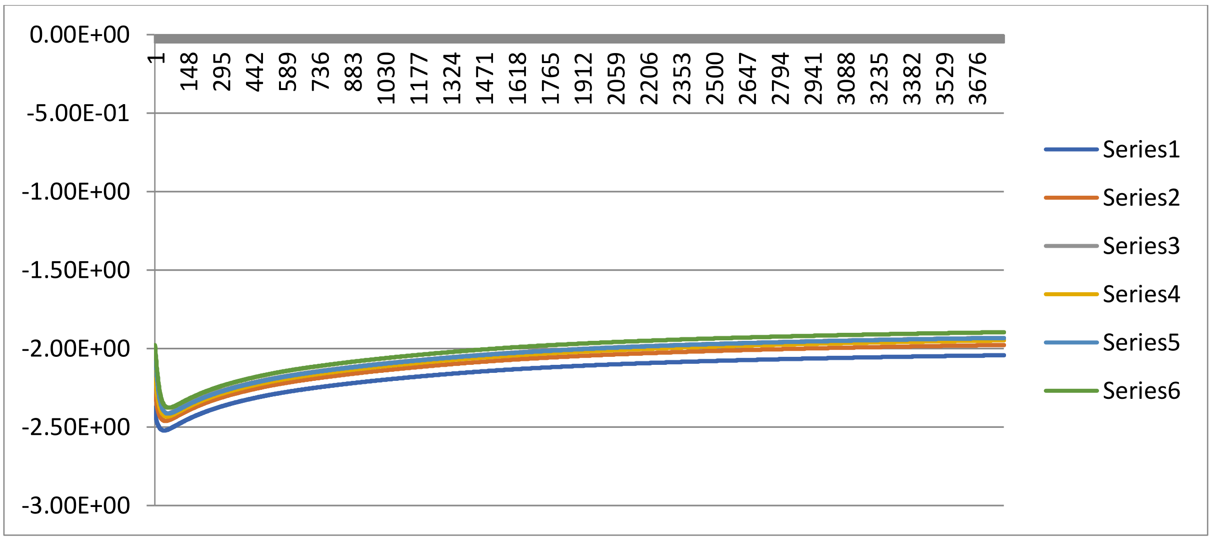


Figure 6 x-axis shows time in ms y-axis shows voltage from material

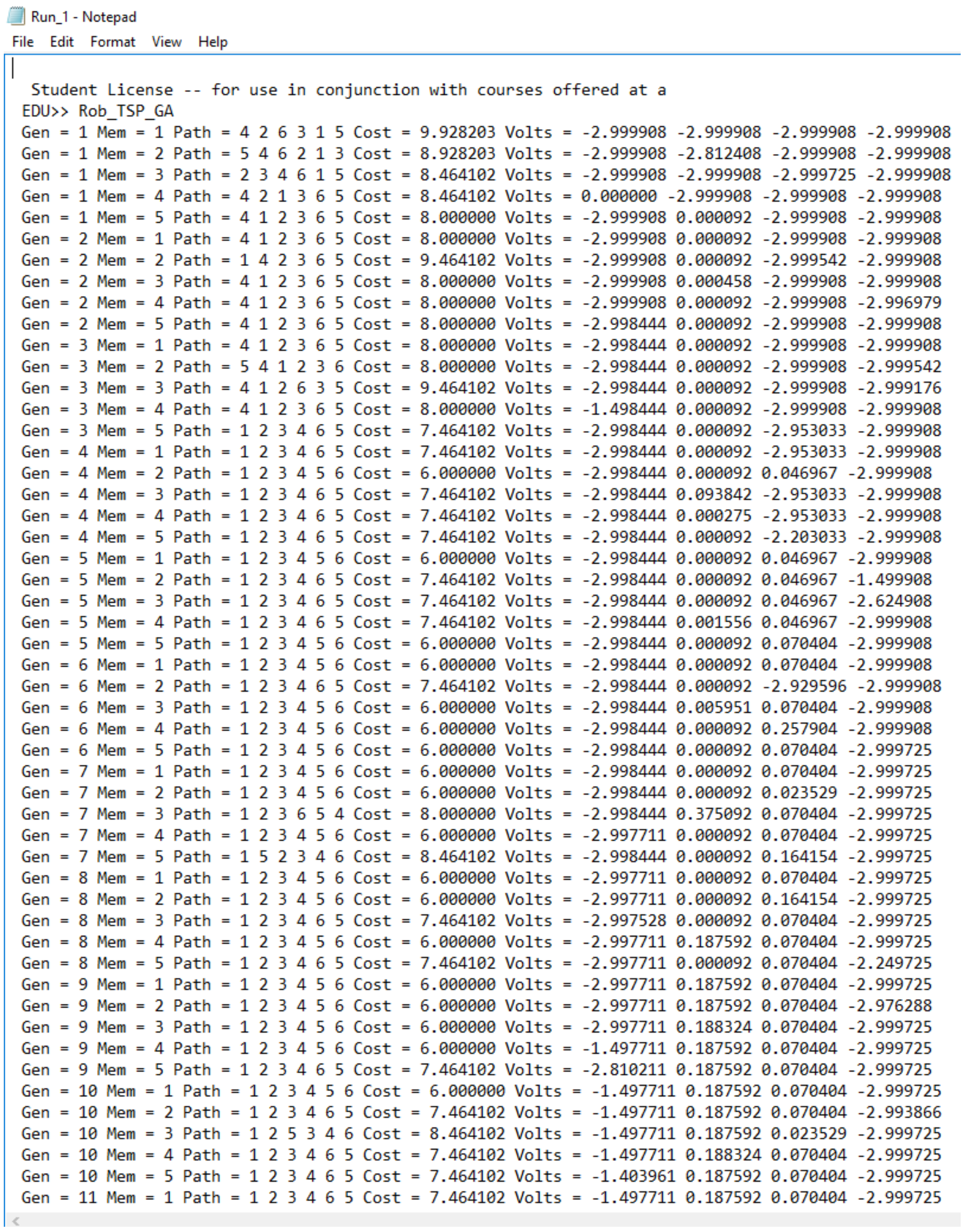
Notice the strange curve in the voltage coming from the material. It is believed that this is an artefact of either the PCI_6259 DAC or ADC hardware. After about 250ms the signals settle to a level and remain at that level.

6.6.3 Sample Interval Length

From the measurements in Figure 6 x-axis shows time in ms y-axis shows voltage from material, there was some concern about how long a sample period was necessary, but looking at the signals in the Figure and many others it was clear that each signal line stayed in the same relative position to all the others over the duration of measurement. It was decided to an average of 250 measurements from 250ms onwards, when the signals had settled. Since the Output Voltages from the Material were being used to represent solutions to a Travelling Salesman problem, then the fact that each pins' voltages retains its relative value over times means this approach is sound.

6.6.4 First GA Run

The output from the first run is shown in Figure 7 :



```
 Student License -- for use in conjunction with courses offered at a
EDU>> Rob_TSP_GA
Gen = 1 Mem = 1 Path = 4 2 6 3 1 5 Cost = 9.928203 Volts = -2.999908 -2.999908 -2.999908 -2.999908
Gen = 1 Mem = 2 Path = 5 4 6 2 1 3 Cost = 8.928203 Volts = -2.999908 -2.812408 -2.999908 -2.999908
Gen = 1 Mem = 3 Path = 2 3 4 6 1 5 Cost = 8.464102 Volts = -2.999908 -2.999908 -2.999725 -2.999908
Gen = 1 Mem = 4 Path = 4 2 1 3 6 5 Cost = 8.464102 Volts = 0.000000 -2.999908 -2.999908 -2.999908
Gen = 1 Mem = 5 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -2.999908 0.000092 -2.999908 -2.999908
Gen = 2 Mem = 1 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -2.999908 0.000092 -2.999908 -2.999908
Gen = 2 Mem = 2 Path = 1 4 2 3 6 5 Cost = 9.464102 Volts = -2.999908 0.000092 -2.999542 -2.999908
Gen = 2 Mem = 3 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -2.999908 0.000458 -2.999908 -2.999908
Gen = 2 Mem = 4 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -2.999908 0.000092 -2.999908 -2.996979
Gen = 2 Mem = 5 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -2.998444 0.000092 -2.999908 -2.999908
Gen = 3 Mem = 1 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -2.998444 0.000092 -2.999908 -2.999908
Gen = 3 Mem = 2 Path = 5 4 1 2 3 6 Cost = 8.000000 Volts = -2.998444 0.000092 -2.999908 -2.999542
Gen = 3 Mem = 3 Path = 4 1 2 6 3 5 Cost = 9.464102 Volts = -2.998444 0.000092 -2.999908 -2.999176
Gen = 3 Mem = 4 Path = 4 1 2 3 6 5 Cost = 8.000000 Volts = -1.498444 0.000092 -2.999908 -2.999908
Gen = 3 Mem = 5 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000092 -2.953033 -2.999908
Gen = 4 Mem = 1 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000092 -2.953033 -2.999908
Gen = 4 Mem = 2 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.046967 -2.999908
Gen = 4 Mem = 3 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.093842 -2.953033 -2.999908
Gen = 4 Mem = 4 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000275 -2.953033 -2.999908
Gen = 4 Mem = 5 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000092 -2.203033 -2.999908
Gen = 5 Mem = 1 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.046967 -2.999908
Gen = 5 Mem = 2 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000092 0.046967 -1.499908
Gen = 5 Mem = 3 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000092 0.046967 -2.624908
Gen = 5 Mem = 4 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.001556 0.046967 -2.999908
Gen = 5 Mem = 5 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.070404 -2.999908
Gen = 6 Mem = 1 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.070404 -2.999908
Gen = 6 Mem = 2 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.998444 0.000092 -2.929596 -2.999908
Gen = 6 Mem = 3 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.005951 0.070404 -2.999908
Gen = 6 Mem = 4 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.257904 -2.999908
Gen = 6 Mem = 5 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.070404 -2.999725
Gen = 7 Mem = 1 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.070404 -2.999725
Gen = 7 Mem = 2 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.998444 0.000092 0.023529 -2.999725
Gen = 7 Mem = 3 Path = 1 2 3 6 5 4 Cost = 8.000000 Volts = -2.998444 0.375092 0.070404 -2.999725
Gen = 7 Mem = 4 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.000092 0.070404 -2.999725
Gen = 7 Mem = 5 Path = 1 5 2 3 4 6 Cost = 8.464102 Volts = -2.998444 0.000092 0.164154 -2.999725
Gen = 8 Mem = 1 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.000092 0.070404 -2.999725
Gen = 8 Mem = 2 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.000092 0.164154 -2.999725
Gen = 8 Mem = 3 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.997528 0.000092 0.070404 -2.999725
Gen = 8 Mem = 4 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.187592 0.070404 -2.999725
Gen = 8 Mem = 5 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.997711 0.000092 0.070404 -2.249725
Gen = 9 Mem = 1 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.187592 0.070404 -2.999725
Gen = 9 Mem = 2 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.187592 0.070404 -2.976288
Gen = 9 Mem = 3 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -2.997711 0.188324 0.070404 -2.999725
Gen = 9 Mem = 4 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -1.497711 0.187592 0.070404 -2.999725
Gen = 9 Mem = 5 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -2.810211 0.187592 0.070404 -2.999725
Gen = 10 Mem = 1 Path = 1 2 3 4 5 6 Cost = 6.000000 Volts = -1.497711 0.187592 0.070404 -2.999725
Gen = 10 Mem = 2 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -1.497711 0.187592 0.070404 -2.993866
Gen = 10 Mem = 3 Path = 1 2 5 3 4 6 Cost = 8.464102 Volts = -1.497711 0.187592 0.023529 -2.999725
Gen = 10 Mem = 4 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -1.497711 0.188324 0.070404 -2.999725
Gen = 10 Mem = 5 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -1.403961 0.187592 0.070404 -2.999725
Gen = 11 Mem = 1 Path = 1 2 3 4 6 5 Cost = 7.464102 Volts = -1.497711 0.187592 0.070404 -2.999725
```

Figure 7 First PC GA Run Output

It worked very well and arrived at an optimal solution at the 6th Generation ! And this was using Graphene Supermarket Graphene dispensed into a modified 16 pin DIP header. Note at this point the EA was not Logging what it was doing to disk, it was just displaying messages to the Matlab command window. Hence the Image in Figure 7.

The meaning of the fields in Figure 7 is as follows:

Gen is generation

Mem is the genome member of a generation, Mem = 1 is the elite from the previous generation

Path is the Order of Cities – they want to be in sequence for an optimal solution,

Cost is the distance travelled traversing the sequence of cities – which are in these experiments arranged in a circle

Volts are the 4 Configuration Voltages Applied to pins on the Material

6.6.5 Noise in Data and Cost Function when Re-evaluating Elites

After the initial Euphoria of seeing the EA setup rapidly produce a TSP solution, problems with the data soon began to appear. After a time of using a particular piece of graphene as material, it would begin to not produce the same consistent results from one generation to the next. This happened where an elite montage of cross-point switch settings and configuration voltages carried forward as is to a subsequent generation, no longer got the same Fitness Score.

To identify precisely what was going wrong it required externalising Cost, Config and Material Output Voltages to disk for each genome of each generation.

Getting Matlab to write output to a CSV was not trivial. This was done as follows:

```
%write results to screen & Log.txt
fprintf('Gen = %d ', GenNo);
fprintf('Mem = %d ', GenMemNo);
fprintf('Try = %d ', TryNo);
fprintf('Samp = %d ', SampNo);
fprintf('Path = '); fprintf('%d ', OriginalPositions);
fprintf('Cost = %f ', cost);
fprintf('Ivar = %f ', inputvar);
fprintf('Iskw = %f ', inputskew);
fprintf('Ikts = %f ', inputkurtosis);
fprintf('Volts = '); fprintf('%f ', volts);
fprintf('AvInputs ='); fprintf('%f ', AvInputs);
fprintf('\n ')

%For first record write a header
if GenNo == 1 && GenMemNo == 1 && TryNo == 1 && SampNo == 1
    fprintf(fid,'GenNo');
```

```
            fprintf(fid,',GenMemNo');
            fprintf(fid,',TryNo');
            fprintf(fid,',SampNo');
            fprintf(fid,',Pos');
            fprintf(fid,',cost');
            fprintf(fid,',Ivar');
            fprintf(fid,',Iskw');
            fprintf(fid,',Ikts');
            volt_len=length(volts);
            for v = 1:volt_len
                fprintf(fid,',V%d',v);
            end
            AvI_len = length(AvInputs);
            for v = 1:AvI_len
                fprintf(fid,',I%d',v);
            end
            fprintf(fid,'\n ');
        end

        fprintf(fid,'%d',GenNo);
        fprintf(fid,',%d',GenMemNo);
        fprintf(fid,',%d',TryNo);
        fprintf(fid,',%d',SampNo);
        fprintf(fid,','); fprintf(fid,'%d ',OriginalPositions);
        fprintf(fid,',%f',cost);
        fprintf(fid,',%f',inputvar);
        fprintf(fid,',%f',inputskew);
        fprintf(fid,',%f',inputkurtosis);
        volt_len=length(volts);
        for v = 1:volt_len
            fprintf(fid,','); fprintf(fid,'%d',volts(v));
        end
        AvI_len = length(AvInputs);
        for v = 1:AvI_len
            fprintf(fid,','); fprintf(fid,'%d',AvInputs(v));
        end
        fprintf(fid,'\n ');
    end
```

The above code segment was preceded earlier in the code by an Open:

```
%open log.txt
fid = fopen('log.txt','wt');
```

The point of doing this was to gather some data. The code had been modified so that it tried the same config 9 times and each time it took 9 samples from the measured AVInputs to get a better idea of what was happening. The intention was to devise an algorithm to determine if a set of AVinputs and their score were stable across several measurements.

The code used to sample was as follows:

```
%take several samples of the data
for SampNo = 1:Samples
    % Calc the Average Voltage measured for each pin used.  I take
    % the period 250ms to 500 ms as the signals all seem to stay
    % in the same relative position, even though they've not yet
    % stabilised
    for i = 1 : columns
        avolt=0;
        measure_start=(250+(SampNo-1)*100);
        Measure_end=(500+(SampNo-1)*100);
        for j = measure_start:Measure_end
            avolt=avolt+capturedData(j,i);
        end
        avolt=avolt/250;
        AvInputs(i)=avolt;
    end
    % Transpose AvInputs for use by evaluateTSP
    AvInputt=AvInputs';

[cost,OriginalPositions,inputvar,inputskew,inputkurtosis]=evaluateTS
P (AvInputt);
                cost=cost;
```

6.6.6 Observations from Analysing the EA Log

(1) There are many samples where the first and last costs are different numbers, typically the first 4 will be the same and the rest are the same. This would suggest the data is being read too soon before the voltages have stabilised. The first measurement starts @ 250ms – & typically after 4 or 5 measurements i.e. from 650 to 750ms the voltages are more stable.

It was believed that fixing this would make a big difference to the variability in the measurements obtained.

(2) Using the same Config Volts & Cross-point Matrix settings repeatedly does seem to get different results sometimes, e.g. when a new Best_cost_score is found, but when that config is tried again in a subsequent generation, it does not always give the same Score.

6.6.7 Adding Rollbacks to the EA

After some analysis of the data produced by logging the progress of the GA, it was thought that perhaps the GA had moved into a region of Config Voltages for the Material which was for some reason unstable. To cope with this, it was decided to add some code which would Rollback Evolution to a point where re-trying an *elite* was stable.

So, the following code segment was added to regress back to an earlier config if a new Generation gets a higher score with an existing config:

```
%check that a cost has not degraded i.e. worst score than when
%first got.  If it is we regress back to the best cost before
%that cost was got
SortedCostArray=sort(CostArray);
if GenNo > 1
    if GenMemNo == 1
        if SortedCostArray(CostsToEvaluate) > log_cost(1)
            csp_mat_use=log_csp_mat_use(:,:,2);
            config_volts_use=log_config_volts(:,:,2);
            cost=log_cost(2);
            log_cost(1)=log_cost(2)
            New_Best_csp_mat = csp_mat_use;
            New_Best_config_volts = config_volts_use;
            New_Best_cost = cost;
            %write results to screen & Log.txt
            fprintf('Gen = %d ', GenNo);
            fprintf('Mem = %d ', GenMemNo);
            fprintf('Adjustment');
            fprintf(fid,'Gen = %d ', GenNo);
            fprintf(fid,'Mem = %d ', GenMemNo);
            fprintf(fid,'Adjustment');
        end
    end
end
```

The existing code segment was also modified to store several versions of config and cost – only if all the Tries and the Samples produce the same cost value. This seemed initially to make the costs much more stable as expected.

```
        SortedImproveCostArray=sort(ImproveCostArray);
%did all costs equal or improve ?
if ImproveCostcounter == CostsToEvaluate
    %are all new costs the same ?
    if SortedImproveCostArray(1) ==
SortedImproveCostArray(CostsToEvaluate)
```

```
        Prev_Best_Cost = cost;
        GenNoScore(GenNo,2) = cost;
        BestMemNo=GenMemNo;
        New_Best_csp_mat = csp_mat_use;
        New_Best_config_volts = config_volts_use;
        New_Best_cost = cost;
        save New_Best_csp_mat.txt New_Best_csp_mat -ascii;
        save New_Best_config_volts.txt New_Best_config_volts -ascii;
        save New_Best_cost.txt New_Best_cost -ascii;
        %Save to arrays to allow regression to earlier setting
        %when costs are no longer as good
        for lg = 2:3
            log_csp_mat_use(:,:,lg)=log_csp_mat_use(:,:,(lg-1));
            log_config_volts(:,:,lg)=log_config_volts(:,:,(lg-1));
            log_cost(lg)=log_cost(lg-1);
        end
        log_csp_mat_use(:,:,1)=csp_mat_use;
        log_config_volts(:,:,1)=config_volts_use;
        log_cost(1)=cost;
    end
```

In practise this meant Evolution would now spend lots of time Rolling back to an earlier generation, which in practise, would often itself prove to be unstable – i.e. retrying the Config Volts and Switch settings did not produce the same results either.

6.6.8 Adding Resistors to Limit the Current Supplied to the Material

There was clearly something going wrong with the material. Perhaps its properties were being changed in some way by having the Config Voltages continuously applied to it. This caused a recall of Simon Harding's Phd Thesis Evolution in Materio (8) where he "*connected 4.7kOhm resistors in series with every connection from the PCB to the LCD to limit any current flowing through the display*". It was decided to try connecting resistors in series from the cross-point switch to each pin of the material.

Figure 8 shows a first attempt at this.

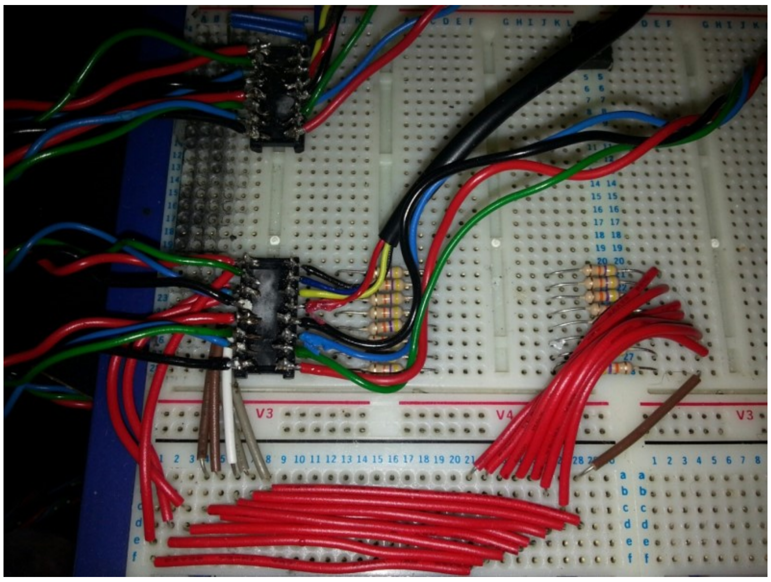


Figure 8 First Resistor Network to limit current to the Material

Running the EA through these resistors in series with the material did not make much difference to the results of the GA, which kept Rolling Back to an earlier successful configuration.

Julian Miller suggested talking to Materials Scientist specialist Dr Iain Will based at York. Iain suggested trying a range of resistor values in addition to the one already tried, which gave rise to the sets of resistors shown in Figure 9 A range of Resistors in Series with Material. Iain also suggested that the material may be becoming affected by atmospheric conditions, either moisture absorption or evaporation and suggested somehow protecting the material from direct contact with air.

The idea was that the resistors would plug into the DIP header socket on the EIM board shown in Figure 9 and then the material plugs into the top of the resistors, which conveniently had been made with a 16 pin DIP socket on top.

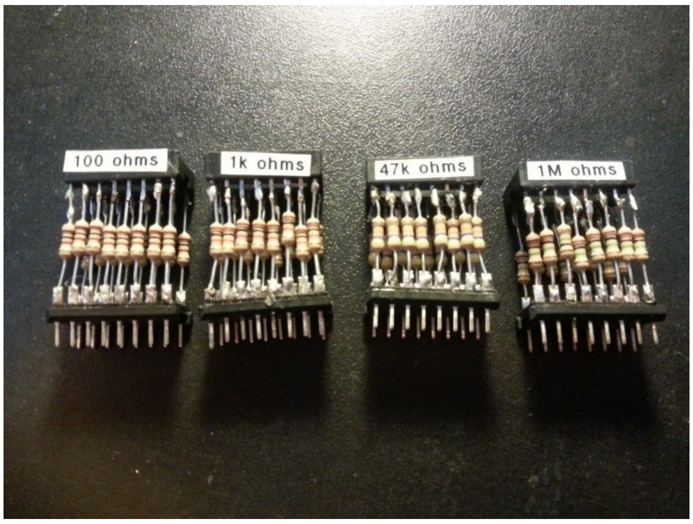


Figure 9 A range of Resistors in Series with Material

None of these sets of resistors made much difference to the stability of the material or performance of the EA, which continued Rolling Back to an earlier successful configuration, because a carried forward elite would no longer get the same score as it did in the previous generation. One impact of using any of these set of resistors was that evolution took much longer to arrive at an optimal solution if it did at all - see Figure 23 for examples.

6.6.9 Saved by Cling Film – Issues with long-term Stability of Materials

Following Iain Will's suggestion of protecting the material from direct contact with air and a little thought, it was decided to try wrapping several dip headers containing some freshly added graphene solution, which had been left for the solvent to evaporate overnight, in cling film. This proved to work very well, showed stability over several months' use and the Rollback processing, which had been added earlier to the code, was no longer invoked. See Figure 10 Graphene in DIP Header wrapped in cling film.

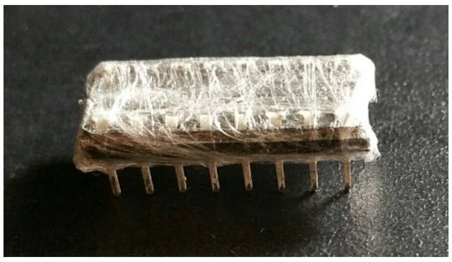


Figure 10 Graphene in DIP Header wrapped in cling film

6.6.10 Alternative Materials Used

Access to a 12 pin carbon nanotube slide used in (11) was provided and so with a connector, some wires, a 16 pin dip header and a bit of soldering, the carbon nanotube slide was easily connected to the board shown in Figure 3 PC PCB Images. Some runs done to enable comparison of graphene and carbon nanotubes as much for use as a sanity check to ensure the graphene was working in a similar way to the carbon nanotube slides used in (11) – see Figure 11 DIY Connector for 12 Pin Nanotube Slide set with a connector.

Similarly, two sets of standard resistors were tried as material – a set of 16 mixed ohm resistors and a set of 16 x 47k ohm resistors see Figure 12 Maplin Resistors Used as Evolvable Material. Both sets of resistors produced solutions to the TSP - see Results sections for further details.

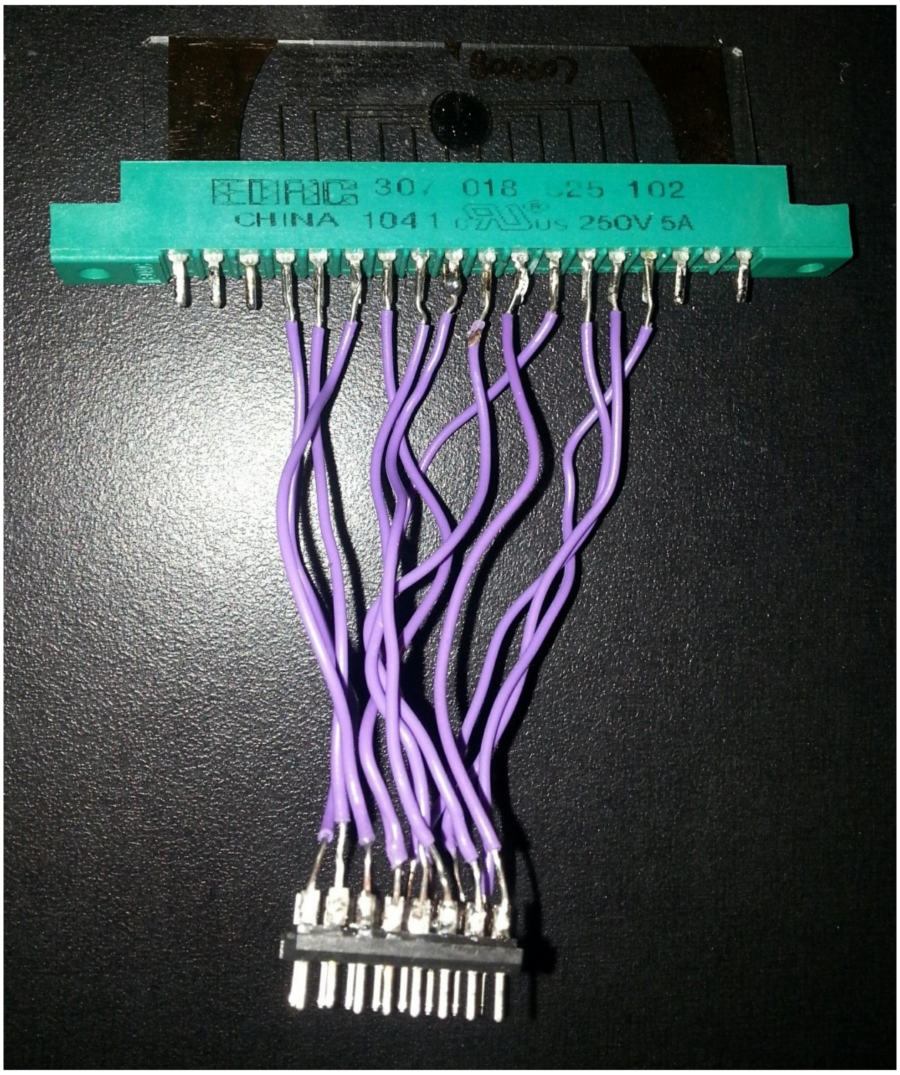


Figure 11 DIY Connector for 12 Pin Nanotube Slide

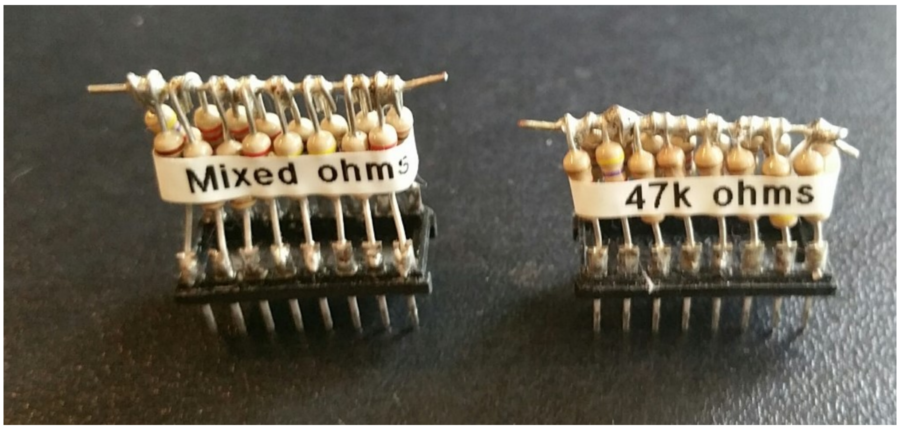


Figure 12 Maplin Resistors Used as Evolvable Material

6.6.11 Summary

An easily portable platform was developed using Matlab, a National Instruments PCI-6259 and custom printed circuit boards, housed in a toolbox. Materials were investigated, and Graphene was chosen as the evolvable material and a means was devised to easily interface the Graphene to the printed circuit boards. Matlab routines were written to configure the AD75019 cross-point switch, and to control and operate the PCI-6259, so that the material could be fed with configuration voltages and measurements taken from the material because of the supplied configuration voltages.

An EA was then implemented using the Matlab routines developed and noise was found to be present in the cost function when re-evaluating elites. This was thought to be due to changing atmospheric conditions and the material was sealed in cling film which made the material more stable.

Alternative materials were investigated including carbon nanotubes and standard resistors, which both proved to be usable for solving TSP Problems.

7 Platform 2: Low-Cost Raspberry Pi Based Platform

7.1 Low-Cost EIM Platform System Choice

Having examined the available Low-Cost development processor options, it was decided to use a Raspberry Pi for the following reasons:

- It has many GPIO pins
- The Pi supports SPI[5] & I2C[6] meaning there are many ADC/DAC devices potentially of use with a Pi
- The Pi runs Python which is similar to Matlab – meaning less of a learning curve and there are tools which claim to be able convert Matlab code to Python - SMOP[7]
- There are several well-established Python IDEs which run native on the Pi such as Pycharm, which provide a complete debugging/development environment
- There is a large educational and enthusiast base with many websites providing help and support
- It has quad core processor running at 1GHz with 1Gigabyte of memory
- A Pi is very cheap - it costs £30
- From reviews, the Pi Platform appeared stable and robust and not prone to random failures.

[5] http://whatis.techtarget.com/definition/serial-peripheral-interface-SPI
[6] http://i2c.info/
[7] https://github.com/victorlei/smop

7.2 Low-Cost EIM ADC/DAC/Cross-Point Switches

7.2.1 Low-Cost EIM ADC/DAC Initial Steps

The next step was to decide what Physical devices were required to run EIM on a Pi. The PCI_6259 had the following Spec:

- 4 x 16-bit Analog outputs (2.8 MS/s); 48 digital I/O; 32-bit counters
- 16-Bit, 1 MS/s (Multichannel), 1.25 MS/s (1-Channel), 32 Analog Inputs

It was clear that for Low-Cost EIM work such a device was a little over specified and costs £1,700.

Researching available Low-Cost ADC/DAC devices for use with the Pi inevitably led to the Adafruit website[8], where several easy to use off the shelf Pi compatible devices were found and the following were purchased:

4 x Adafruit ADS1115 – a 16 Bit ADC with 4 differential Inputs

4 x Adafruit MCP4725 - which has a single 12bit output.

1 x Adafruit Pi Cobbler – a ribbon cable attached Pi GPIO expander for a breadboard.

The idea was to get a feel for prototyping an EIM on the Pi without going down the whole design route before getting any results.

A power supply was required for the Adafruit components so a YwRobot was found on Ebay. YwRobot is a little power supply providing 3.3 and 5V from a USB or 6.5 – 12V via a barrel socket. Getting the YwRobot to fit on a breadboard with the rest of the Adafruit components was a bit of a fiddle – the pins had to be bent at the bottom as it did not quite fit the breadboard. The YwRobot appeared to stop working after a little while – the green LED went out. A bit of fiddling revealed the LED had come unsoldered – so it was soldered back together and it worked fine again. YwRobot avoids the hassle of wiring up L7805s as it has a couple of AMS1117s – one does 5V and the other does 3V. YwRobot is a handy little power device.

[8] www.adafruit.com

The devices were placed on a breadboard, wired together, powered up and connected to a Raspberry Pi's 40 pin GPIO header. All the devices were visible from the Pi via the Adafruit Pi Cobbler using:

```
Sudo i2cdetect  –y  1
```

I2cdetect showed the 4 x ADS1115s at addresses 48, 49 4A & 4B and the 2 x MCP4725s at addresses 62 & 63 – you can only have 4 ADS1115s & 2 x MCP4725s connected to a single I2C bus. See Figure 13 below.

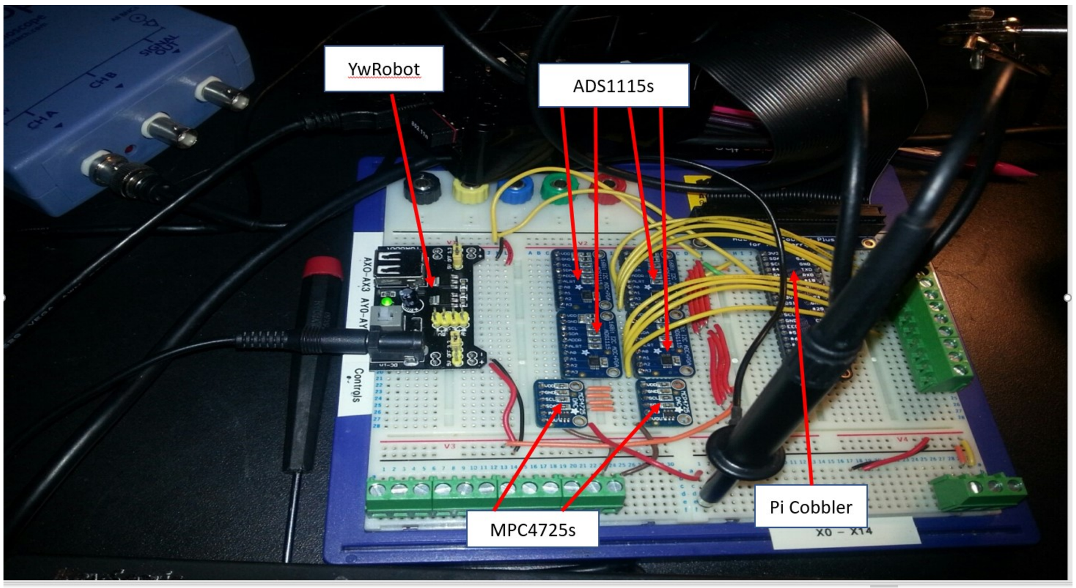


Figure 13 Breadboard with Adafruit ADCs/DACs & YwRobot

7.2.2 Low-Cost EIM ADC/DAC Custom Development Board

In parallel with experimenting with Adafruit devices, it was decided to request the build of a custom development board which would interface to the Pi via a ribbon cable to its GPIO pins. Pete Turner of The University of York Electronics Department suggested having an additional AD75019 cross-point switch, arranged so that the Evolvable Material could be presented with up to 16 Inputs fed by the 2 x LTC2657 12bit DACs (20) or drive up to 16 Outputs read by the 24bit LTC2499 (21), or have some intermediate number of inputs and outputs all configurable and controllable by the Python EIM code running on the Raspberry Pi. This was proposed in part to reduce the amount of potential interference, from having wires running from terminal block to terminal block, but also to provide the maximum flexibility and software configurability. Figure 14 Pi Development Board Logical Summary and Figure 15 Pi Development Board Schematic shows a logical Summary of the Pi and devices and the Schematic shows the interconnections between the Pi GPIO Header and the devices' signalling lines and interconnections. Figure 16 Development Board with Pi Mounted shows an image of the finished board complete with a mounted Raspberry Pi, indicating the Pi, DACs, cross-point switches and ADC.

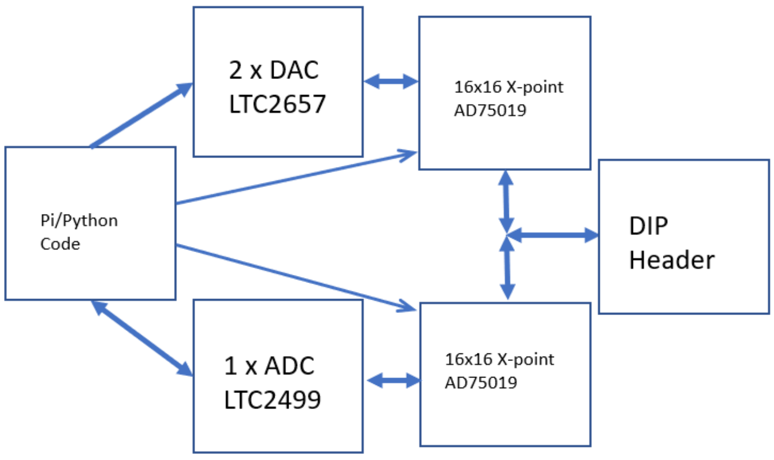


Figure 14 Pi Development Board Logical Summary

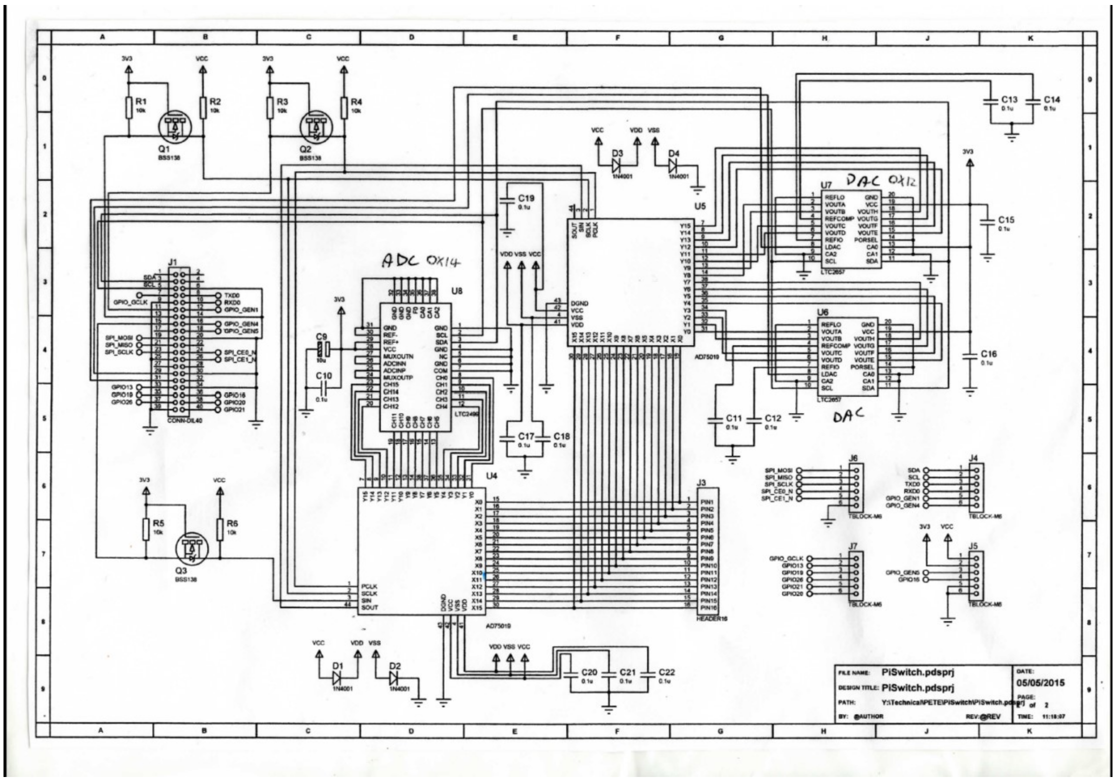


Figure 15 Pi Development Board Schematic

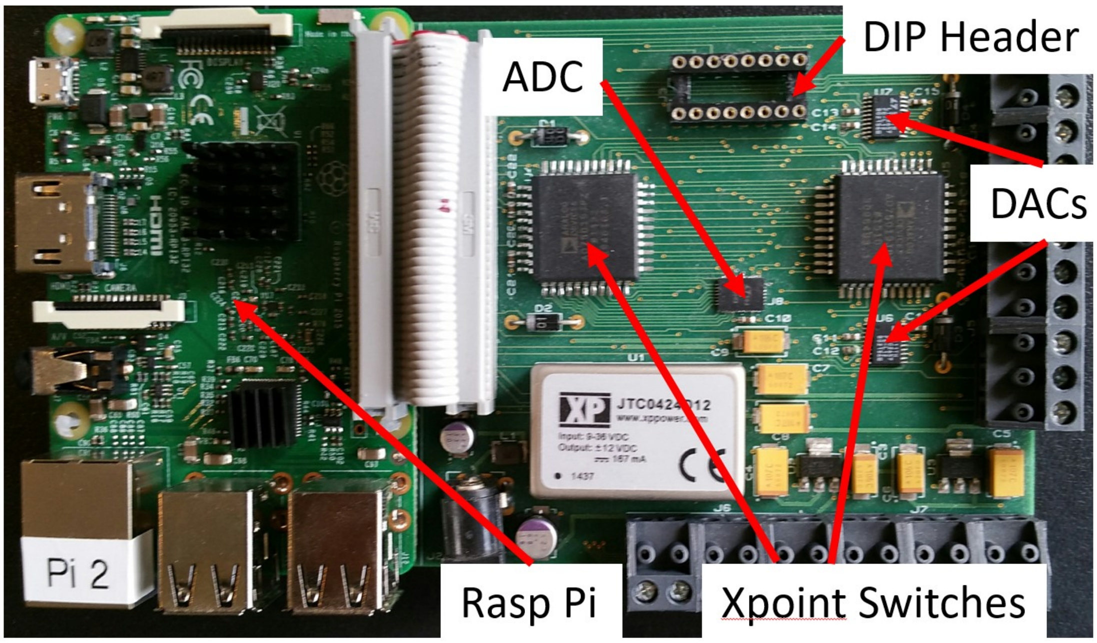


Figure 16 Development Board with Pi Mounted

7.2.3 Low-Cost EIM ADC/DAC Implications

The National Instruments PCI_6259 DAQ board is a high-performance board. The components being used in the Low-Cost EIM Project here are much lower specification. The PCI_6259 can process 16 Analogue Inputs in parallel at 1.25 Megabytes per second, whereas the LTC2499 only samples 24bits at approximately 3.5 samples per second and has a single ADC which is multiplexed between the 16 inputs. Also, the PCI_6259 can generate up to 4 Analogue outputs at 16 bits, whereas the two LTC2657DACs can generate up to 16 Analogue outputs at 12bit resolution. The key question here was, could such low spec, slow devices achieve comparable results to a PC running Matlab using a PCI_6259?

7.2.4 Necessary Things for Setting Up a Raspberry Pi to Develop this Python EIM Application

Note; it's not completely trivial getting a Pi to do this. Here is a list of things that are required to be done on a Pi once the operating system is initially installed:

sudo apt-get update

sudo apt-get upgrade

sudo apt-get dist-upgrade

sudo apt-get install python-tk

sudo apt-get install i2c-tools

sudo apt-get install python-smbus

sudo apt-get install python-numpy

sudo apt-get install python-opencv

sudo apt-get install python-scipy

sudo apt-get install ipython

Install the Latest version of Pycharm (18) – a Python IDE

Install numpy into Pycharm

Install Quick2wire[9]:

The following was key to getting the LTC2499 to work with Pi using Quick2wire[10] - this took many months to find and was very valuable so it is included here[11]:

"by BudBennett » Tue Oct 11, 2016 2:06 pm
Quick2wire is still available on GitHub[12]
I don't use their install script but simply put the quick2wire-python-api-master folder in a convenient location and then add the two export lines to my PYTHONPATH as they recommend.

The following I2C video is invaluable since this project uses I2C protocol to communicate with the ADC & DACs, it provides a deep dive into I2C, should you need to do any debugging[13]

The Saleae I2C Logic Analyzer is invaluable for debugging & checking what is happening at the I2C level in any project[14] – considerable use was made of it in debugging problems with the LTC2499.

[9] www.github.com/quick2wire

[10] www.raspberrypi.org/forums/viewtopic.php?f=37&t=64503https://www.raspberrypi.org/forums/viewtopic.php?t=64503&p=583062

[11] https://www.raspberrypi.org/forums/viewtopic.php?t=64503&p=477232

[12] *https://github.com/quick2wire/quick2wire-python-api*

[13] www.youtube.com/watch?v=kxaFbDY-wH0

[14] https://www.amazon.co.uk/gp/product/B00ISTG89C/ref=oh_aui_search_detailpage?ie=UTF8&psc=1

7.2.5 Cross-Point Switch Debugging Technique

An invaluable debugging technique suggested by Pete Turner for the cross-point switch code is to load a cross-point switch setting into the switch, generate a waveform on a Picoscope or similar and feed that waveform into one of the material pin sockets e.g. Pin16 which corresponds to X15 (Switch 1 - U4) on J3 HEADER16 in the Figure 15 Pi Development Board Schematic and then check that this signal is really coming from the expected Y pin on the cross-point switch. Then repeat this for each of the other pins on the header. Note; there will be a small amount of cross talk between adjacent signal paths within the switch, but these interference signals are significantly attenuated.

7.2.6 Configuring & Debugging the Cross-Point Switch Code

The cross-point switches on the board are daisy chained with U4 first followed by U5, as pin SOUT from U4 goes to pin SIN on U5. This means to configure both devices, the config for U5 must be send first followed by the config for U4.

In writing the cross-point switch Python code a bug had been coded which took a couple of weeks to solve. This section illustrates the cross-point switch debugging technique above.

Setup_CSPs.py code was as follows:

```
import RPi.GPIO as GPIO
import smbus
import time
import numpy as np

SIN = 22
SCLK = 27
PCLK = 17
bus = smbus.SMBus(1)

# set up gpio
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(SIN, GPIO.OUT)
GPIO.setup(SCLK, GPIO.OUT)
GPIO.setup(PCLK, GPIO.OUT)

# Init the Cross Point Switch Arrays
csp_mat1 = np.eye(16, dtype=int)
csp_mat2 = np.eye(16, dtype=int)

# Transpose them
csp_mat1_t=np.transpose(csp_mat1)
csp_mat2_t=np.transpose(csp_mat2)
```

```
# Copy and redefine from 16x16 to 256x1
csp_mat1_r=np.reshape(csp_mat1_t,(1,256))
csp_mat2_r=np.reshape(csp_mat2_t,(1,256))

# set SIN low, SCLK low and PCLK high ready to init the XpointSwitches
GPIO.output(SIN, 0)
GPIO.output(SCLK, 0)
GPIO.output(PCLK, 1)

#Populate XpointSwitch U4
# AD75019 doc says you need to send the CSP matrix bits in reverse order
than in the matrix
# so you send Y15 X15 first then the remaining X values down to X0,
followed by bits for row
# Y14 and so on down to Y0 X0.
for x in range(255,0,-1):
    # AD75019 doc says 1=CLOSE 0=OPEN
    if csp_mat1_r[0,x] == 0:
        GPIO.output(SIN, 0)
        print(x,0)
    else:
        GPIO.output(SIN, 1)
        print(x,1)
    GPIO.output(SCLK, 1)
    time.sleep(0.00001)  # wait for value to take
    GPIO.output(SCLK, 0)

#Populate XpointSwitch U5
for x in range(255,0,-1):
    # AD75019 doc says 1=CLOSE 0=OPEN
    if csp_mat1_r[0,x] == 0:
        GPIO.output(SIN, 0)
        print(x,0)
    else:
        GPIO.output(SIN, 1)
        print(x,1)
    GPIO.output(SCLK, 1)
    time.sleep(0.00001)  # wait for value to take
    GPIO.output(SCLK, 0)

## toggle PCLK to load data
time.sleep(0.000001)  # wait for value to take
GPIO.output(PCLK, 0)
time.sleep(0.000001)  # wait for value to take
GPIO.output(PCLK, 1)
```

The bug was discovered by configuring the cross-point switch with the setting `csp_mat1 = np.eye(16, dtype=int)`, generating an Analogue Signal from the Picoscope see Figure 17 Example Picoscope Diagnostic Waveform and feeding it in turn into each of the pins on the 16 way DIP Connector and tracing that signal through the U4 16 way switch checking that:

PIN01 signal goes to X00 on U4
PIN02 signal goes to X01 on U4
PIN03 signal goes to X02 on U4
PIN04 signal goes to X03 on U4
PIN05 signal goes to X04 on U4
PIN06 signal goes to X05 on U4
PIN07 signal goes to X06 on U4
PIN08 signal goes to X07 on U4
PIN09 signal goes to X08 on U4

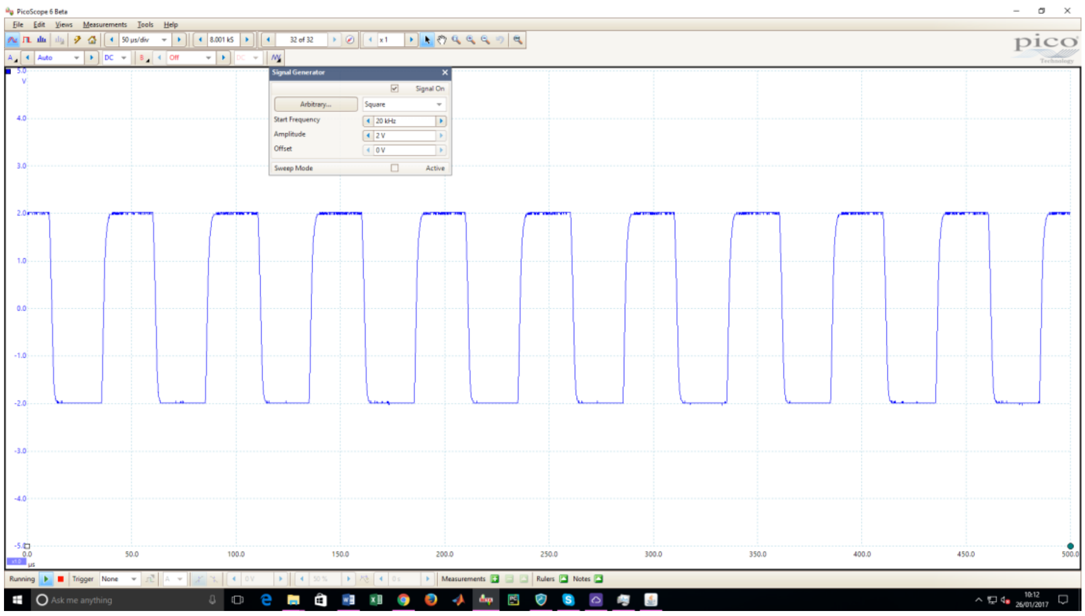
PIN10 signal goes to X09 on U4
PIN11 signal goes to X10 on U4
PIN12 signal goes to X11 on U4
PIN13 signal goes to X12 on U4
PIN14 signal goes to X13 on U4
PIN15 signal goes to X14 on U4
PIN16 signal goes to X15 on U4

These all proved to be fine.

But then when on checking which pin the X's go to, the Y's are as:

X00 → Y01
X01 → Y02
X02 → Y03
X03 → Y04
X04 → Y05
X05 → Y06
X06 → Y07
X07 → Y08
X08 → Y09
X09 → Y10
X10 → Y11
X11 → Y12
X12 → Y13
X13 → Y14
X14 → Y15
X15 → ??

This was not good at all and illustrates that the code to configure the switches needed to be carefully tested or it can appear to work fine, but end up doing something quite different.



Signal Generator

Signal On ✓

Arbitrary... Square

Start Frequency 20 kHz

Amplitude 2 V

Offset 0 V

Sweep Mode Active

Figure 17 Example Picoscope Diagnostic Waveform

It was eventually discovered that the Setup_CSPs.py code had the following bug which is what was causing it all to be out by 1:

```
for x in range(255,0,-1):
```

This was changed to

```
for x in range(255,1,-1):
```

This fixed the problem and following that fix the Setup_CSPs.py code which populates two 16x16 Arrays was executed. This sets both AD75019s, which when checked where the X's route to, the Y's were as they should be:

X00 → Y00
X01 → Y01
X02 → Y02
X03 → Y03
X04 → Y04
X05 → Y05
X06 → Y06
X07 → Y07
X08 → Y08
X09 → Y19
X10 → Y10
X11 → Y11
X12 → Y12
X13 → Y13
X14 → Y14
X15 → Y15

This was verified by feeding the Picoscope Signal Out to each of the pins on the DIP header J3 and then checking that the signal appears on the associated Y pins on U4 & U5, but not to any of the other X or Y Pins on either chip. The same is true for X15 → Y15, X14 → Y14… X0 → Y0. Figure 18 Checking Signal Routing on AD75019s shows an illustration of feeding a signal into the DIP header and reading it out on the appropriate AD75019 Y Pin.

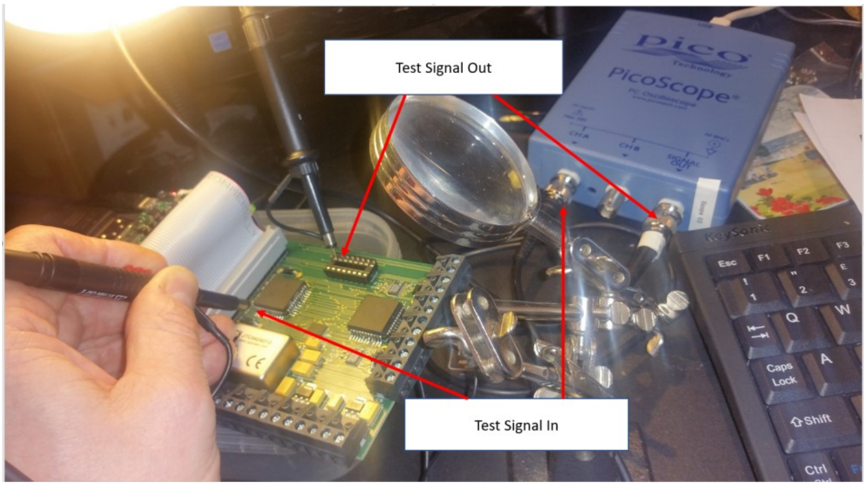


Figure 18 Checking Signal Routing on AD75019s

7.2.7 Simplifying the Code to Ensure Both Cross-Point Switches are Logically consistent

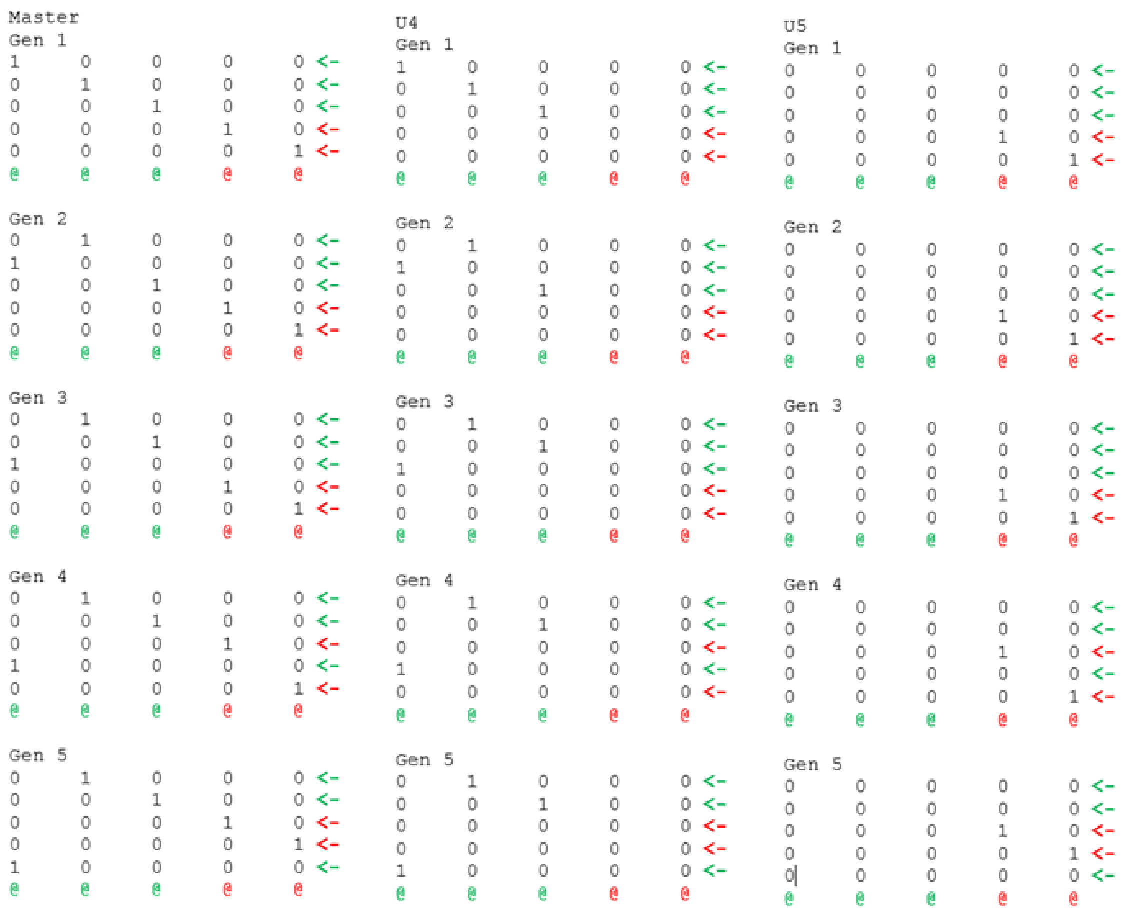
Designing this part was one of the author's greatest causes for concern, until the solution became apparent.

A simple way of maintaining the ADC75019 cross-point switches in step with each other was needed such that complicated checking logic was not required to ensure that the switches were doing what was expected over hundreds or thousands of generations.

It was an EA requirement that 1 to many relationships are not allowed between the logical x and y pins and a given J3 Header pin will only ever be a DAC or an ADC pin and never both a DAC and an ADC pin. The two-separate cross-point switches U4 & U5 could be considered as a logical single cross-point switch with an ADC and a DAC component, where the ADC component is U4 and the DAC component is U5.

A simple example of a 5x5 switch illustrates how a single 5x5 switch array could represent two other 5x5 arrays U4 and U5, where the **red** represents DACs and the **green** represent

ADCs and apply a single mutation each time, where the DACs and the ADCs ONLY connect to the AD75019 Y pins. The example below in Figure 19 Illustrating How to manage two Switches as one shows how a set of mutations to a single cross-point switch could represent the contents of two cross-point switches and how those definitions might be separated.



```
Master
Gen 1
1      0      0      0      0 <-
0      1      0      0      0 <-
0      0      1      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 2
0      1      0      0      0 <-
1      0      0      0      0 <-
0      0      1      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 3
0      1      0      0      0 <-
0      0      1      0      0 <-
1      0      0      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 4
0      1      0      0      0 <-
0      0      1      0      0 <-
0      0      0      1      0 <-
1      0      0      0      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 5
0      1      0      0      0 <-
0      0      1      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
1      0      0      0      0 <-
@      @      @      @      @
```

```
U4
Gen 1
1      0      0      0      0 <-
0      1      0      0      0 <-
0      0      1      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
@      @      @      @      @

Gen 2
0      1      0      0      0 <-
1      0      0      0      0 <-
0      0      1      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
@      @      @      @      @

Gen 3
0      1      0      0      0 <-
0      0      1      0      0 <-
1      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
@      @      @      @      @

Gen 4
0      1      0      0      0 <-
0      0      1      0      0 <-
0      0      0      0      0 <-
1      0      0      0      0 <-
0      0      0      0      0 <-
@      @      @      @      @

Gen 5
0      1      0      0      0 <-
0      0      1      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
1      0      0      0      0 <-
@      @      @      @      @
```

```
U5
Gen 1
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 2
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 3
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 4
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      1      0 <-
0      0      0      0      0 <-
0      0      0      0      1 <-
@      @      @      @      @

Gen 5
0      0      0      0      0 <-
0      0      0      0      0 <-
0      0      0      1      0 <-
0      0      0      0      1 <-
0      0      0      0      0 <-
@      @      @      @      @
```

Figure 19 Illustrating How to manage two Switches as one

So, using using a virtual AD75019, representing 2 x AD75019s, having mutated it, the following needs to be done:

Identify which X cords correspond to Y12 – Y15 – *xi*, *xj*, *xk*, *xl*, assuming there are 4 Config lines, initialise a csp_mat_U5 to zeros and set the following coordinates to 1:

csp_mat_U5(*xi*, Y12)=1
csp_mat_U5(*xi*, Y13)=1
csp_mat_U5(*xi*, Y14)=1

csp_mat_U5(*xi*, Y15)=1

And similarly Identify Which X cords correspond to Y0 – Y11 – *xm*, *xn*…*xw*, *xx*, assuming (16 – 4) 12 result pins are being measured, initialise a csp_mat_U4 to zeros and set the following coordinates to 1:

csp_mat_U5(*xm*, Y0)=1
csp_mat_U5(*xm*, Y1)=1
…
csp_mat_U5(*xw*, Y10)=1
csp_mat_U5(*xx*, Y11)=1

This was coded in setupcsps.py as follows:

```
import RPi.GPIO as GPIO
import smbus
import time
import numpy as np

# Setup the Two AD75019 16x16 Cross Point Switches

def setupcsps(n_control_lines, csp_mat):

    SIN = 22
    SCLK = 27
    PCLK = 17
    bus = smbus.SMBus(1)

    # set up gpio
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(SIN, GPIO.OUT)
    GPIO.setup(SCLK, GPIO.OUT)
    GPIO.setup(PCLK, GPIO.OUT)

    # Now construct the individual Switch Arrays for U4 & U5
    #  U5 connects to the DACs
    csp_size = len(csp_mat)
    # U4 connects to the ADC
    csp_mat_u4 = np.zeros((csp_size,csp_size), dtype=int)
    # U5 connects to the DACs
    csp_mat_u5 = np.zeros((csp_size, csp_size), dtype=int)
    # now buid csp_mat_U5 based on the contents of csp_mat_use

    if n_control_lines == 0:  # Special Case of Calibration we want 2 x
16x16 eye arrays to route ALL signals
        csp_mat_u5 = csp_mat
        csp_mat_u4 = csp_mat
        #print(csp_mat_u5)
        #print(csp_mat_u5)
    else :                    # Normal Case Calibration we want 2 x 16x16
eye arrays to route ALL signals
        r1 = csp_size-n_control_lines
        for y_coord in range(r1, csp_size):
            # oldrow=find_(csp_mat_cpy[:,y1])
            # find the x coordinate of the y_coord value
            for oldrowi in range(0, csp_size):
                if csp_mat[oldrowi, y_coord] == 1:
                    oldrow = oldrowi
            csp_mat_u5[oldrow, y_coord] = 1
        #print(csp_mat_u5)
        # now buid csp_mat_U4 based on the contents of csp_mat_use
        r2 = csp_size - n_control_lines
        for y_coord in range(0, r2):
            # oldrow=find_(csp_mat_cpy[:,y1])
            # find the x coordinate of the y_coord value
            for oldrowi in range(0, csp_size):
                if csp_mat[oldrowi, y_coord] == 1:
                    oldrow = oldrowi
            csp_mat_u4[oldrow, y_coord] = 1
        #print(csp_mat_u4)

    # Transpose them
    csp_mat_u4_t = np.transpose(csp_mat_u4)
    csp_mat_u5_t = np.transpose(csp_mat_u5)

    # Copy and redefine from 16x16 to 256x1 for downloading to AD75019
    csp_mat_u4_r = np.reshape(csp_mat_u4_t, (1, 256))
    csp_mat_u5_r = np.reshape(csp_mat_u5_t, (1, 256))

    # set SIN low, SCLK low and PCLK high ready to init the XpointSwitches
    GPIO.output(SIN, 0)
    GPIO.output(SCLK, 0)
    GPIO.output(PCLK, 1)

    #Populate XpointSwitch U4
    # AD75019 doc says you need to send the CSP matrix bits in reverse
order than in the matrix
    # so you send Y15 X15 first then the remaining X values down to X0,
followed by bits for row
    # Y14 and so on down to Y0 X0.
    # for x in range(255,-1,-1):   Note the (255, -1, -1) - originally had
(255, 0, -1) took weeks to
    # figure out why this did not work so appreciate :)
    # NOTE we do U5 first because that is daisey chained onto U4, which is
done second
    for x in range(255, -1, -1):
        # AD75019 doc says 1=CLOSE 0=OPEN
        if csp_mat_u5_r[0,x] == 0:
            GPIO.output(SIN, 0)
            #print(x,0)
        else:
            GPIO.output(SIN, 1)
            #print(x,1)
        GPIO.output(SCLK, 1)
        #time.sleep(0.00001)  # wait for value to take
        GPIO.output(SCLK, 0)
    for x in range(255, -1, -1):
        # AD75019 doc says 1=CLOSE 0=OPEN
        if csp_mat_u4_r[0,x] == 0:
            GPIO.output(SIN, 0)
            #print(x,0)
        else:
            GPIO.output(SIN, 1)
            #print(x,1)
        GPIO.output(SCLK, 1)
        #time.sleep(0.00001)  # wait for value to take
        GPIO.output(SCLK, 0)

    ## toggle PCLK to load data
    time.sleep(0.000001)  # wait for value to take
    GPIO.output(PCLK, 0)
    time.sleep(0.000001)  # wait for value to take
    GPIO.output(PCLK, 1)
```

7.3 Interfacing the ADC (LTC2499) in Python

Writing and debugging the ADC code in Python for the LTC2499 took the most amount of effort of the whole project. This was in part because the python smbus routines appear not to work well for block reads. After trying lots of proposed solutions in[15] [16] and other sites, Quick2wire was used instead of smbus because this was the only proposed solution that worked in this case and it works very well.

The problem with smbus occurred when the following command was issued more than once:

Bus.write_i2c_byte_block_data(DEVICE_ADDRESS_ADC, ADC_MODE1, ADC_MODE2)

Python would return "Errno 5", which can mean device does not exist. The device did exist before the second or more Bus.write commands were issued, but would no longer appear on the list of I2C devices shown available by an i2cdetect command after the second or more Bus.write command. Strangely then after issuing a subsequent i2cdetect command the device would magically re-appear.

A significantly modified version of the Quick2wire ADC code presented in thread[17] was used which writes out the required material pin voltages measured to a flat file LTC2499_volts.txt:

```
#!/usr/bin/env python3

# must use python 3 because of quick2wire interface

# LTC2499 I2C address is 7'b0100110 = 0x14 when all address pins tied low
import sys
import time
import numpy as np
from Q2W_i2c import *
import math

class LTC2499:

    # define variables
    __LTC2499_config = 0b10100000 # set ADC for external input, 60Hz reject
    # define channel list
    __LTC2499_channels = [0b10110000 # Input CH 0: 1-end, norm Polarity
                         ,0b10111000 # Input CH 1: 1-end, norm Polarity
                         ,0b10110001 # Input CH 2: 1-end, norm Polarity
                         ,0b10111001 # Input CH 3: 1-end, norm Polarity
                         ,0b10110010 # Input CH 4: 1-end, norm Polarity
```

[15] www.stackoverflow.com
[16] www.raspberrypi.org/forums/
[17] https://www.raspberrypi.org/forums/viewtopic.php?f=37&t=64503

```
                                ,0b10111010 # Input CH 5: 1-end, norm Polarity
                                ,0b10110011 # Input CH 6: 1-end, norm Polarity
                                ,0b10111011 # Input CH 7: 1-end, norm Polarity
                                ,0b10110100 # Input CH 8: 1-end, norm Polarity
                                ,0b10111100 # Input CH 9: 1-end, norm Polarity
                                ,0b10110101 # Input CH 10: 1-end, norm Polarity
                                ,0b10111101 # Input CH 11: 1-end, norm Polarity
                                ,0b10110110 # Input CH 12: 1-end, norm Polarity
                                ,0b10111110 # Input CH 13: 1-end, norm Polarity
                                ,0b10110111 # Input CH 14: 1-end, norm Polarity
                                ,0b10111111 # Input CH 15: 1-end, norm Polarity
                                ]

    # Constructor
    def __init__(self, address=0x14,sample_count = 1):
        self.address = address
        self.sample_count = sample_count

    def get_adc_conversion(self, channel, config):
        """
        This configures the LTC2499 for a conversion type on a channel.
Then it will perform n
        conversions on that same channel. This is more efficient than
performing a config and
        read for every conversion.
        """
        result_array = []
        time.sleep(0.135) # wait for previous conversion to end
        # set adc channel to convert
        Q2Wwrite8(self.address,channel,self.__LTC2499_config)

        # convert n times on the channel
        for i in range(self.sample_count):
            time.sleep(0.135)     # allow time for the conversion (Fconv ~
7.5Hz)
            # read result into most significant Byte ... least significant
Byte
            msB2, msB1, msB0, lsB = Q2WreadListNoReg(self.address,4)
            # the result is in two's complement, strip off sign bit and
ms_bit for conversion to integer
            # the sign_bit is used to check for ADC overrange - implement
this later
            sign_bit = msB2
            sign_bit = sign_bit >>7 # extract the sign bit
            ms_bit = msB2
            ms_bit = (ms_bit >> 6) & 0x01 #mask off the ms bit
            msB2 =   0x3F & msB2 # remove sign bit and ms_bit from msB2

            # shift the bytes by appropriate power and add together to get
result
            ms_bit = ms_bit << 24
            result = (msB2 << 24) + (msB1 << 16)+ (msB0 << 8) + lsB
            result = result >> 7 # Shift the noise bits out of the result
            # convert to integer from two's complement and check for adc
overflow
            if (ms_bit > 0 and sign_bit > 0):
                # this is an ADC overflow condition
                result_array.append(16777216/2 +1)
            elif(ms_bit > 0 and sign_bit == 0):
                result_array.append(result - 16777216/2)
            elif(ms_bit == 0 and sign_bit == 0):
                # this is an ADC overflow condition
                result_array.append(-16777216/2 -1)
            else:
                result_array.append(result)
        return result_array

    def get_adc_voltage(self, channel):
        v = self.get_adc_conversion(self.__LTC2499_channels[channel], 
self.__LTC2499_config)
        v[:] = [float(3.3 * x/16777216.0) for x in v]
        return v

    def meanstdv(self,x):
        """
        Calculate mean and standard deviation of data x[]:
        mean = {\sum_i x_i \over n}
        std = math.sqrt(\sum_i (x_i - mean)^2 \over n-1)
        """
        n, mean, std = len(x), 0, 0
        for a in x:
            mean = mean + a
        mean = mean / float(n)
        for a in x:
            std = std + (a - mean)**2
        if(n > 1):
            std = math.sqrt(std / float(n-1))
        else:
            std = 0.0
        return mean, std

    def measure_voltage(self,channel):
        """

        :rtype: object
        """
        x = self.get_adc_voltage(channel)
        mean,std = self.meanstdv(x)
        return mean

    def read_all_channels(self,howmanyadcchannels):
        volts=[]
        for ch_nr in range(howmanyadcchannels):
            mean = self.measure_voltage(ch_nr)
            #print("Channel {0} Voltage: {1:.7f} V".format(ch_nr,mean))
            volts.append(mean)
            #print(volts)
        # Save voltages from volts to disk
        np.savetxt('/home/pi/GA/LTC2499_volts.txt', volts)

    def Get_Volts(self,address,sample_count,howmanyadcchannels):
        adc = LTC2499(address)   # initialize
        adc.read_all_channels(howmanyadcchannels)
        #print(self.volts)

  # test code:1
if __name__=="__main__":

    adc = LTC2499(0x14) #initialize
    adc.read_all_channels()
```

7.4 Interfacing the DACs (LTC2657) in Python

Compared to the ADC getting the DACs code to work in Python was much more straight forwards but not without its own challenges.

The first issue was the fact that there are two 12bit LTC2657 DACs, which each produce voltages on 8 separate channels or pins. For a coding perspective these are better dealt with as a single logical device. Hence the DAC_DETAILS array in the example setup_dacs.py has the I2C addresses and device mode which identifies which DAC pin is associated with that request. This enables the code to treat both devices as a single entity.

The next issue was related to the fact that the LTC2657 device is a 12bit device and the parameters being passed to it are 16bit. Many of the values passed are not on a 12bit boundary and would often, but not always, cause a TypeError exception so logic was added to round down dac_value1 to the nearest 12bit boundary. Despite doing this, there were still some TypeErrors and the cause of these was never identified or resolved so the logic in the except routine either decremented or incremented DAC_VALUE(1) according to its content. Don't knock it - it works fine.

The setup_dacs routine is shown here:

```
#  Code to configure LTC2657 over I2C

import RPi.GPIO as GPIO
import smbus
import time
import numpy as np


def setup_dacs(dac_first_pin, dac_values):

    # datashape = dac_values.shape
    # DAC_COLS = datashape[0]
    DAC_COLS = len(dac_values)

    bus = smbus.SMBus(1)

    # set up gpio
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)

    # define Array to contain DAC Addresses and the device_mode which determines
which pin we are turning on Thus :
    # set up i2c for U7 device_address = 0x12
    # set up i2c for U6 device_address = 0x10
    # device_mode = 0x3F #Turn on all DACs - NOT used in this program
    DAC_DETAILS = np.array([[0x12, 0x37],  #devaddr=0x12 & devmode=0x37 Turn on DAC H
                            [0x12, 0x36],  #devaddr=0x12 & devmode=0x36 Turn on DAC G
                            [0x12, 0x35],  #devaddr=0x12 & devmode=0x35 Turn on DAC F
                            [0x12, 0x34],  #devaddr=0x12 & devmode=0x34 Turn on DAC E
                            [0x12, 0x33],  #devaddr=0x12 & devmode=0x33 Turn on DAC D
                            [0x12, 0x32],  #devaddr=0x12 & devmode=0x32 Turn on DAC C
                            [0x12, 0x31],  #devaddr=0x12 & devmode=0x31 Turn on DAC B
                            [0x12, 0x30],  #devaddr=0x12 & devmode=0x30 Turn on DAC A
                            [0x10, 0x37],  #devaddr=0x10 & devmode=0x37 Turn on DAC H
                            [0x10, 0x36],  #devaddr=0x10 & devmode=0x36 Turn on DAC G
                            [0x10, 0x35],  #devaddr=0x10 & devmode=0x35 Turn on DAC F
                            [0x10, 0x34],  #devaddr=0x10 & devmode=0x34 Turn on DAC E
                            [0x10, 0x33],  #devaddr=0x10 & devmode=0x33 Turn on DAC D
                            [0x10, 0x32],  #devaddr=0x10 & devmode=0x32 Turn on DAC C
                            [0x10, 0x31],  #devaddr=0x10 & devmode=0x31 Turn on DAC B
                            [0x10, 0x30]]) #devaddr=0x10 & devmode=0x30 Turn on DAC A

    # device_mode = 0x3F #Turn on all DACs
    # device_mode = 0x3F #Turn all 8 Channels on

    # dac_values = [0xFF, 0xFF]

    # For each DAC Address we want a signal on from the first Pin to the Last set it
up
    for x in range(dac_first_pin, DAC_COLS, 1):
        device_address = DAC_DETAILS[x, 0]
        device_mode = DAC_DETAILS[x, 1]
        dac_value = dac_values[x]
        dac_value1 = dac_value[1]
        if dac_value1 <= 16:
            dac_value1 = 0
        elif dac_value1 <= 32:
            dac_value1 = 16
        elif dac_value1 <= 48:
            dac_value1 = 32
        elif dac_value1 <= 64:
            dac_value1 = 48
        elif dac_value1 <= 80:
            dac_value1 = 64
        elif dac_value1 <= 96:
            dac_value1 = 80
        elif dac_value1 <= 112:
            dac_value1 = 96
        elif dac_value1 <= 128:
            dac_value1 = 112
        elif dac_value1 <= 144:
            dac_value1 = 128
        elif dac_value1 <= 160:
            dac_value1 = 144
        elif dac_value1 <= 176:
            dac_value1 = 160
        elif dac_value1 <= 192:
            dac_value1 = 176
        elif dac_value1 <= 208:
            dac_value1 = 192
        elif dac_value1 <= 224:
            dac_value1 = 208
        elif dac_value1 <= 240:
            dac_value1 = 224
        elif dac_value1 <= 256:
            dac_value1 = 240
        else:
            dac_value1 = 128

        dac_value[1] = dac_value1

        if dac_value1 < 129:
            dv = "low"
        else:
            dv= "hig"
        trys = 0
        # sometimes the bus.write fails with a TypeError so I increment the second
value in dac_value
        while True:
            try:
                #print(x, device_address, device_mode, dac_value[0], dac_value[1])
                bus.write_i2c_block_data(device_address, device_mode, dac_value)
                break
            except TypeError:
                # get the second parm and increment it by one
                # if that make it > 256 set it to 255
                value= dac_value[1]
                if trys == 0:
                    dac_value[1] = value
                else:
                    value = dac_value[1]
                    if dv == "low":
                        value = value - 1
                    if dv == "hig":
```

```
                    value = value + 1
                if value > 256:
                   value = 254
                if value < 1:
                    value = 0
                dac_value[1] = value
            trys = trys + 1
            #print(x, device_address, device_mode, dac_value)
            #bus.write_i2c_block_data(device_address, device_mode, [0, 1])
    dac_value = dac_value
return
```

7.5 Calibrating the DAC / ADC

Because of ignorance on the part of the author, the DAC and the ADC devices chosen were not matched in terms of their resolution or their voltage ranges. To cope with this a calibration routine was coded which stepped through the range of possible DAC values each of which were read by the ADC to find what practical range of voltages should be imposed on the DAC so that its signals always fell within a range of values measurable by the ADC.

The Calibration.py module below generated a set of DAC Values on each DAC pin which were then read by the ADC on each of its 16 input channels and the result of each measurement written to disk. Figure 20 DAC ADC Calibration Curve shows the voltage range of the DAC, readable by the ADC was 0 to 1.65V.

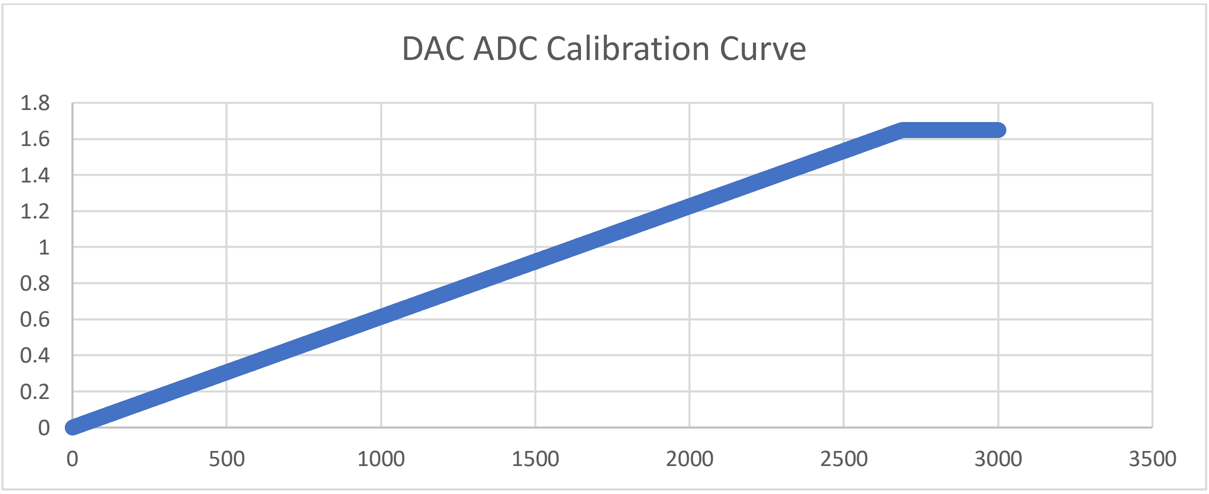


Figure 20 DAC ADC Calibration Curve

Note the Calibration.py module makes use of the other modules illustrated in this text and gives a basic function test that the cross-point switch, the DACs and the ADC are all working correctly before going on to do any EA work. Running Calibration.py was also the first real continuous processing test on the Raspberry Pi – A version of Calibration.py was left to run for 36 hours and the Pi handled it all beautifully with no problems or errors. Perhaps the heat sinks on the processor and graphics card which can be seen Figure 16 Development Board with Pi Mounted in may have helped. Seeing the Pi being able to easily handle hours of continuous processing like this was a confidence builder and is a tribute to the quality and design of this little £30 credit card sized machine.

The Calibration.py routine is shown here:

```
# Calibrate the ADC & DAC Devices
#
#
import datetime
import numpy as np
#from SetUp_CSPs import *
from setup_dacs import *
from setupcsps import *
from LTC2499_test_v5 import *

# Define and Open for output the Calibration Filename
logname = ('/home/pi/GA/Calibration.txt')
fidlog = open(logname, 'w')

# Setup Crosspoint Switches
#SetUp_CSPs
n_control_lines = 0
csp_mat = np.eye(16, dtype=int)
setupcsps(n_control_lines, csp_mat)

#Setup DAC Values used to define the LTC2657 Voltages on all 16 Pins of U6
& U7
for x1 in range(0x00, 0x0100, 0x01):
    for x2 in range(0x00, 0x0100, 0x10):
        #Setup DAC
        dac_values = [[x1, x2],[x1, x2],[x1, x2],[x1, x2],[x1, x2],[x1,
x2],[x1, x2],[x1, x2],[x1, x2],[x1, x2],[x1, x2],[x1, x2],[x1, x2],[x1,
x2],[x1, x2],[x1, x2]]
        dac_first_pin = 0
        setup_dacs(dac_first_pin, dac_values)
        address=0x14
        sample_count=1
        howmanyadcchannels=4
        adc = LTC2499(address)
        adc.Get_Volts(address,sample_count,howmanyadcchannels)
        # Get the measurements got by setUpAnalogueIOs in captureddata.txt
        captureddata = np.loadtxt('/home/pi/GA/LTC2499_volts.txt',
delimiter=',')
        datashape = captureddata.shape
        rows = datashape[0]
        # build a logline with the measurement data
        logline = ''
        ts = '{:%Y-%m-%d,%H:%M:%S}'.format(datetime.datetime.now())
        logline = ts + ',' + str(x1) + ',' + str(x2)
        for v in np.arange(0, rows):
            logline = logline + ',' + str(captureddata[v])
        logline = logline + '\n'
        fidlog.write(logline)
        p = str(x1) + ' ' + str(x2) + ' ' + str(captureddata[0]) + ' ' +
str(captureddata[1])
        print(p)
fidlog.close()
```

7.6 Implementation of the EA

An attempt was made to use SMOP[18] to automagically convert the Matlab code developed for the High Cost platform into Python code to run on the Pi. This was a useful exercise in that the Python code started out broadly like the PC Matlab code, but because of limitations in SMOP, every line of code had to be checked and significant changes had to be made manually to many aspects of the code, to ensure Python arrays provided similar functionality to Matlab matrices, that the function and parameter passing worked properly and that the code had the same overall functionality.

The functional building blocks described earlier, in this chapter – DAC/ADC/cross-point switch code were incorporated in the Python EA code and once done, the two EAs were very similar. A conscious decision was made to NOT include the Rollback functionality, as the problems with material stability appeared to have been solved by wrapping the material in cling film.

In execution the Python code behaved very well on the Pi and the Pycharm IDE provided all the functionality needed in testing, debugging and running the code.

7.7 Summary Low-Cost Raspberry Pi Platform

The Raspberry Pi was chosen as the processor for the Low-Cost platform, Pycharm (18) Python chosen as the IDE and the ADC/DACs were identified along with 2 cross-point switches allowing 1-16 configuration voltages and 1-16 Outputs. A custom development

[18] https://github.com/victorlei/smop

board was designed and produced by the York Electronics Lab housing a Raspberry Pi. By its nature this system was a much lower specification and power than a PC with a PCI-6259 running Matlab.

Interface code was developed to control the AD75019 cross-point switches, the LTC2499 ADC and the LTC2657 DACs. The ADC and DACs were calibrated on the development board so that the EA constrained the DACs' output voltages to be between 0 and 1.65V, which was the usable range of voltages the ADC could read from the DACs.

8 Results from the High-Cost Platform

8.1 Commentary

Much of the time spent working with the High Cost Platform was spent attempting to get the material to perform reliably and produce consistent results. An example of a run using unstable material is shown in Figure 21. This run included use of Rollbacks to attempt to keep evolution within areas where more reliable stable results were obtained. Its shows no sign of getting to an optimal fitness of 6. Note that this and subsequent graphs of EA runs shows EACH evaluation of every genotype. This explains the noise in the curve as evolution tries different mutations each generation to arrive at the optimal fitness value of 6.

Similarly, Figure 22 shows a set of runs where the material started out behaving fine for the 6, 7 City problems, but then degraded when attempting to do 8, and for 9 & 10 Cities did not arrive at a solution.

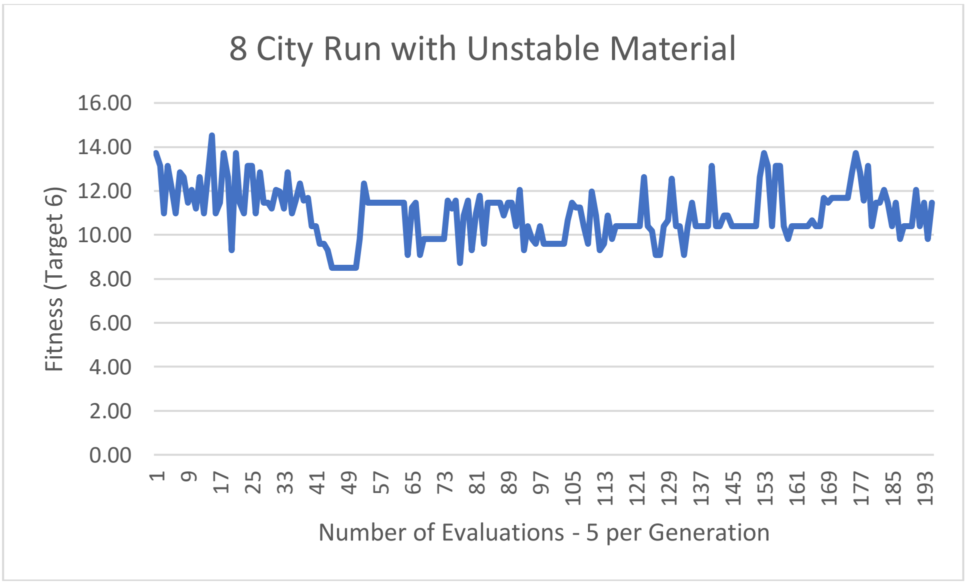


Figure 21 8 City Run with Unstable Material No Hope of Reaching Fitness Target of 6

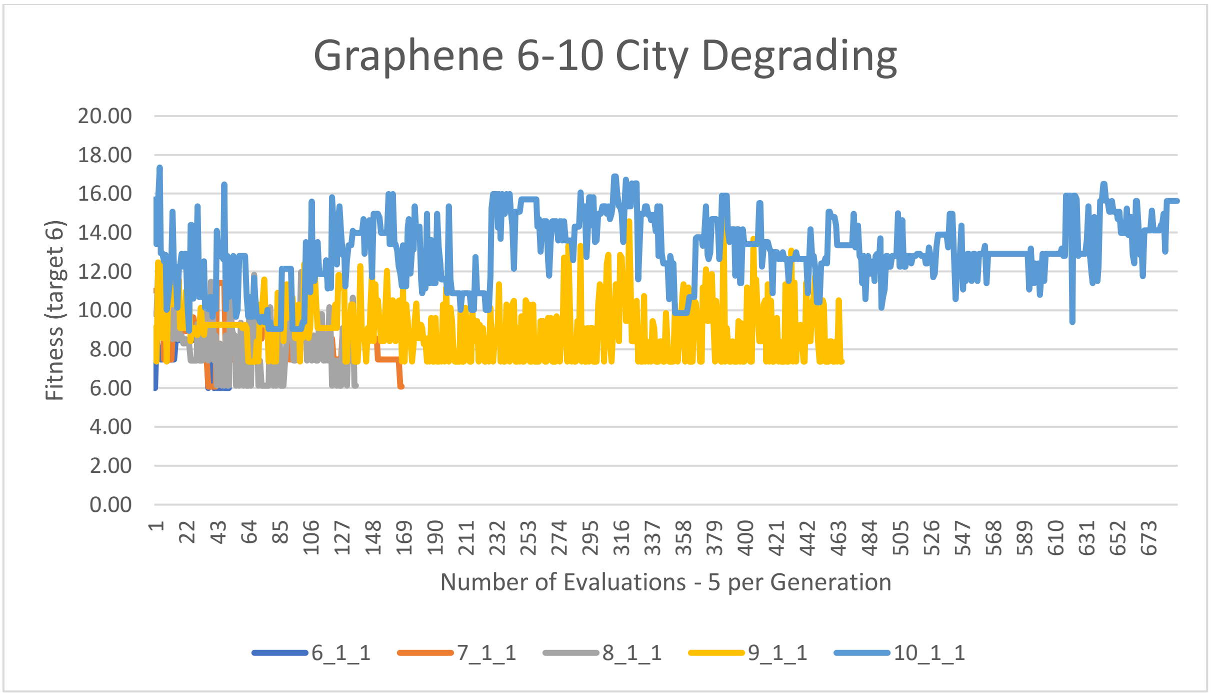
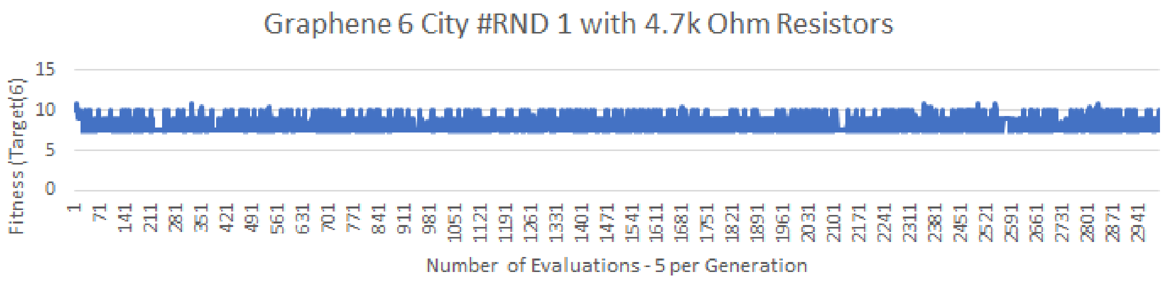
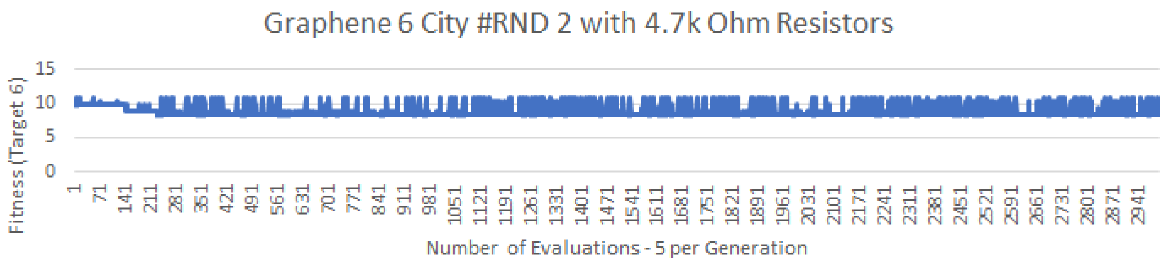


Figure 22 Partially Degraded Material

Attempts to use resistors in series Figure 9 with new material tended to produce results like Figure 23 where evolution would progress very slowly, but even after prolonged periods of time – 5 or 6 hundred generations did not arrive at the optimum score of 6. Each of these 6 City TSP runs were the same, differing only in random number seed of 1, 2 or 3.





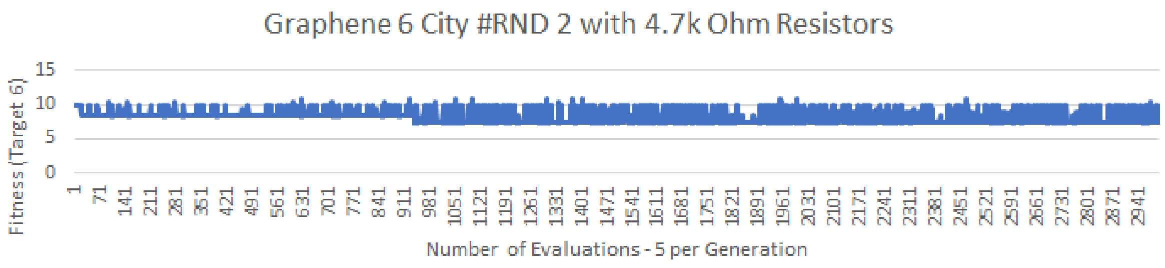


Figure 23 Examples of 6 City Runs with 4.7k Resistors Unsuccessfully meeting the Fitness Target

Some carbon nanotube slide results are shown in Figure 24 to give a sense of perspective as to how well this performs compared to Graphene. It was a delight to work with the carbon nanotube slides – they just worked consistently and reliably.

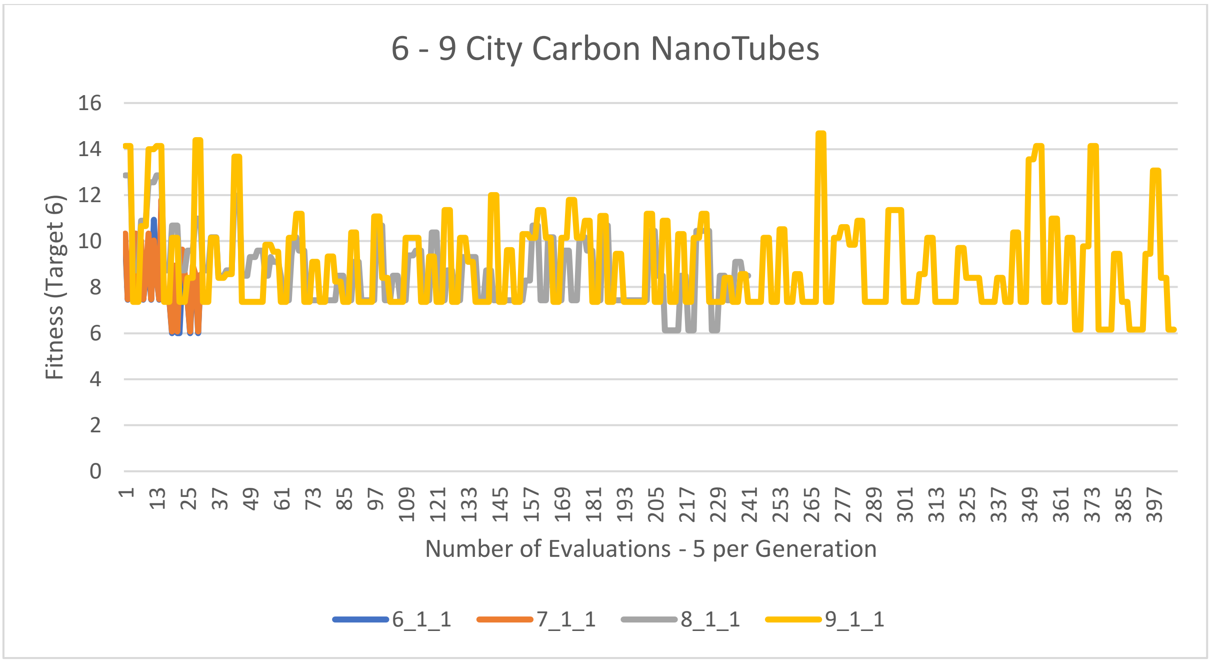


Figure 24 Carbon Nanotube results for 6-9 Cities Evolved until Fitness Target of 6

Figure 25 & Figure 26 illustrate the impact that the random seed has on the progress of evolution. Here there were different runs done with random seeds going up in steps of 1 from 1 to 5.

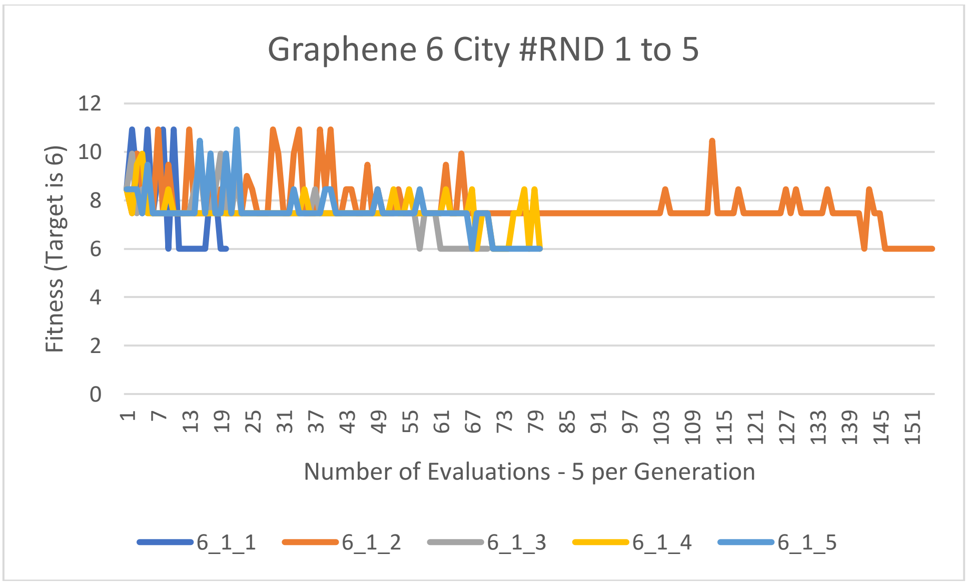


Figure 25 Graphene 6 City Results for Random seeds 1 to 5, Evolved until Fitness Target of 6 Reached

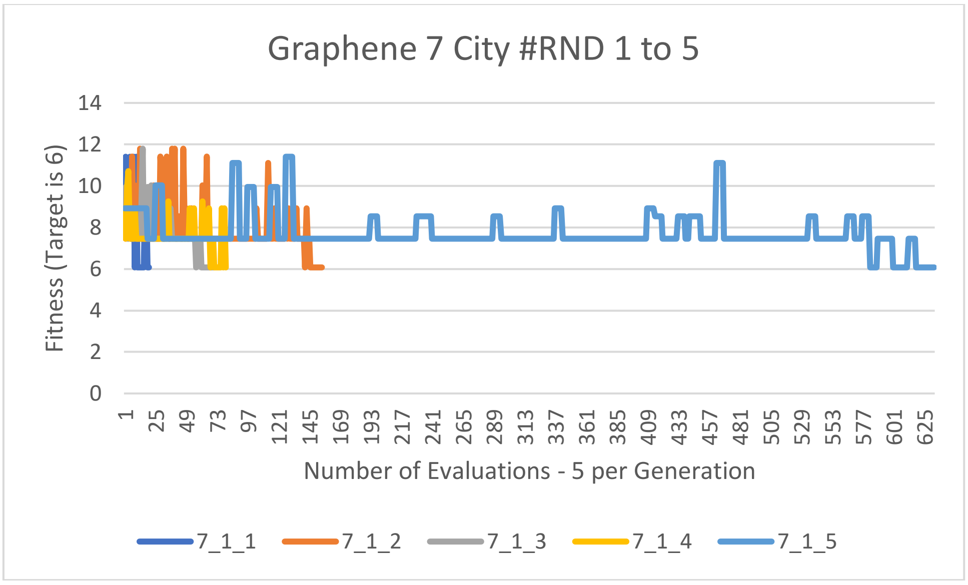


Figure 26 Graphene 7 City Results for Random seeds 1 to 5, Evolved until Fitness Target of 6 Reached

8.2 Summary of High-Cost EiM Results

The results for the graphene, carbon nanotubes and resistors were broadly similar and would suggest that any conductive material with a small resistance could be used for this type of work. The key to success is having a stable material, which does not degrade with use or time and provides repeatable results.

Before discovering the cling film solution, many attempts were made to embed graphene in wood and PVA glues without any success. There is probably an endless supply of other materials, which would make good evolvable materials.

9 Results from the Low-Cost EiM Platform

9.1 Perspective

The results from the Low-Cost platform were very similar to the High-Cost platform, regardless of the fact that the Low-Cost DACs were only 12bit compared to the High-Cost 16bit DACs. Similarly, the Low-Cost ADC, which although 24bit, only had a single ADC which was multiplexed under EA control to its 16 pins and could only sample at 3.5Hz compared to the High-Cost ADC which was 16bit and capable of continuously sampling at 50k samples per second across all channels at the same time.

While working on the Low-Cost platform, it was found that even wrapped in cling film, the graphene eventually degrades after a few months. The run in Figure 27 started well with each City problem size achieving the Fitness Target of 6, but on commencing processing the 10 City problem in Figure 28, the graphene started well but could not progress beyond a Fitness of 7.295 and by evaluation 3596 (generation 719) showed no sign of being able to improve further. This is typical of what happens when the graphene degrades. The symptom is a carried forward elite from the previous generation no longer being able to get the same fitness score as it did in the generation before.

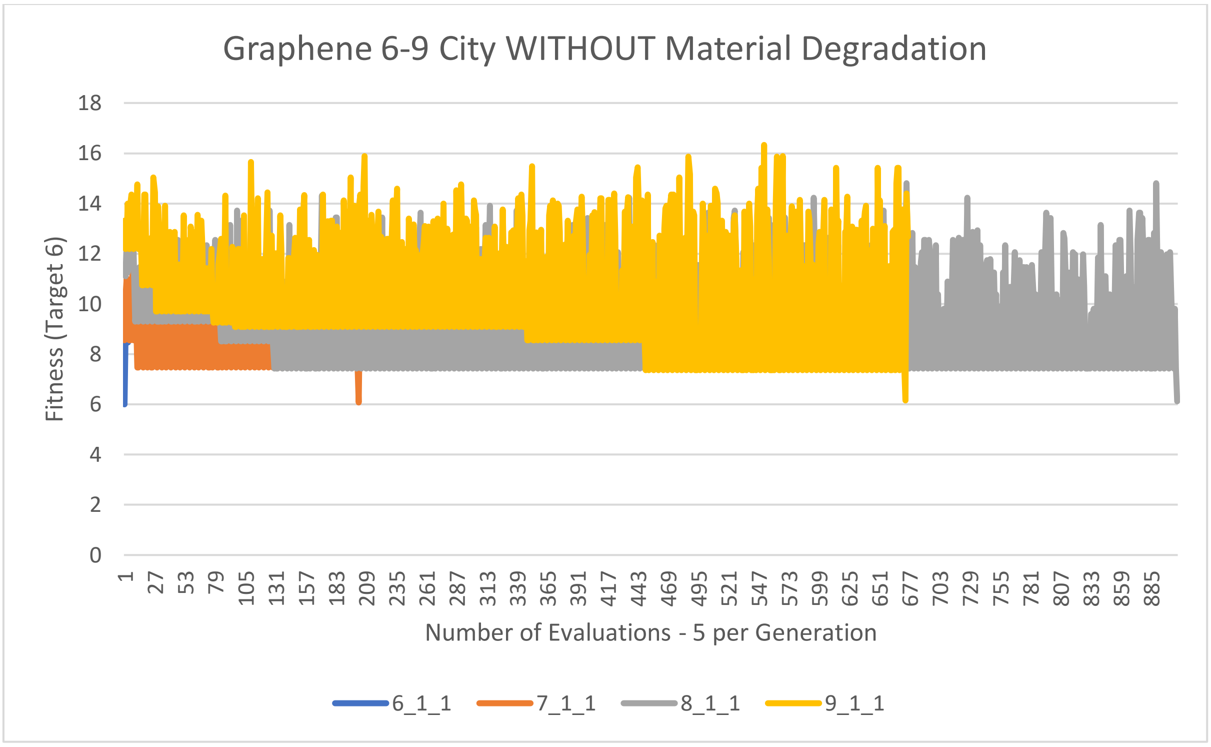


Figure 27 Figure 27 Graphene 6-9 City Without Degradation Evolved until Fitness of 6 Reached

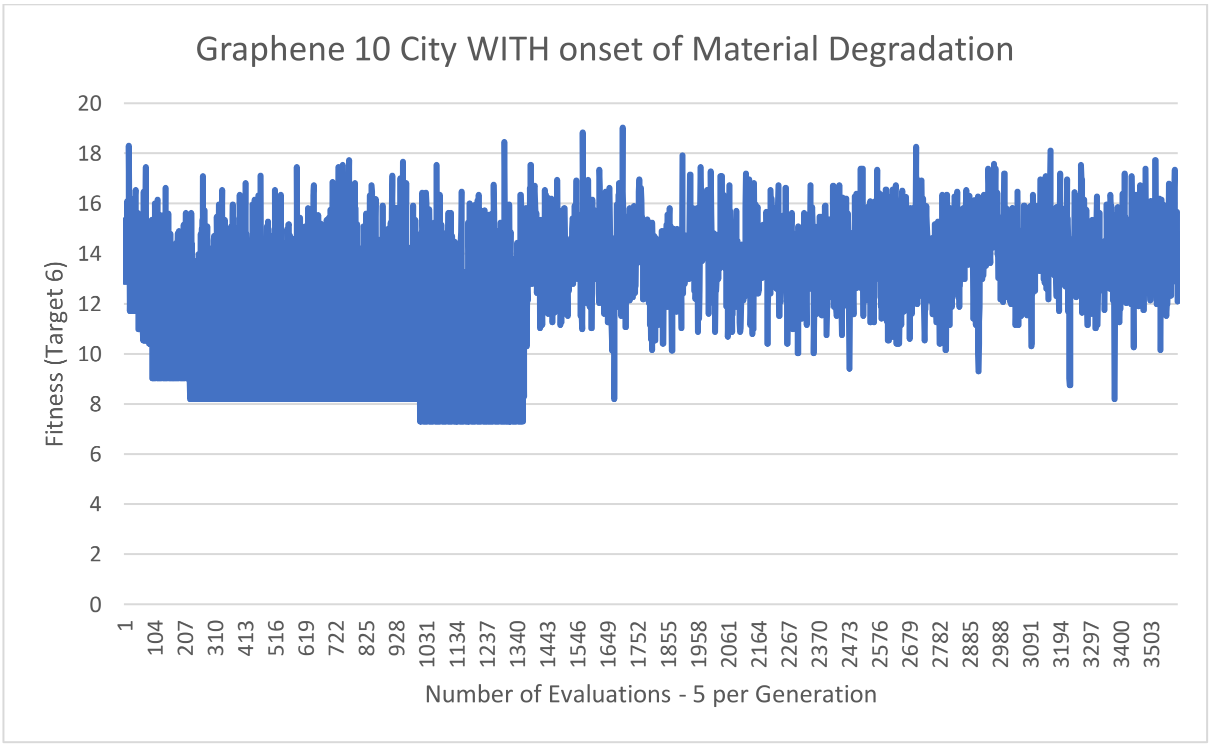


Figure 28 Graphene 10 City With onset of Degradation Fitness Target 6 NOT Reached

Fresh graphene works fine – all the 6 – 10 City TSP problems arrived at the Optimal Fitness of 6, see Figure 29.

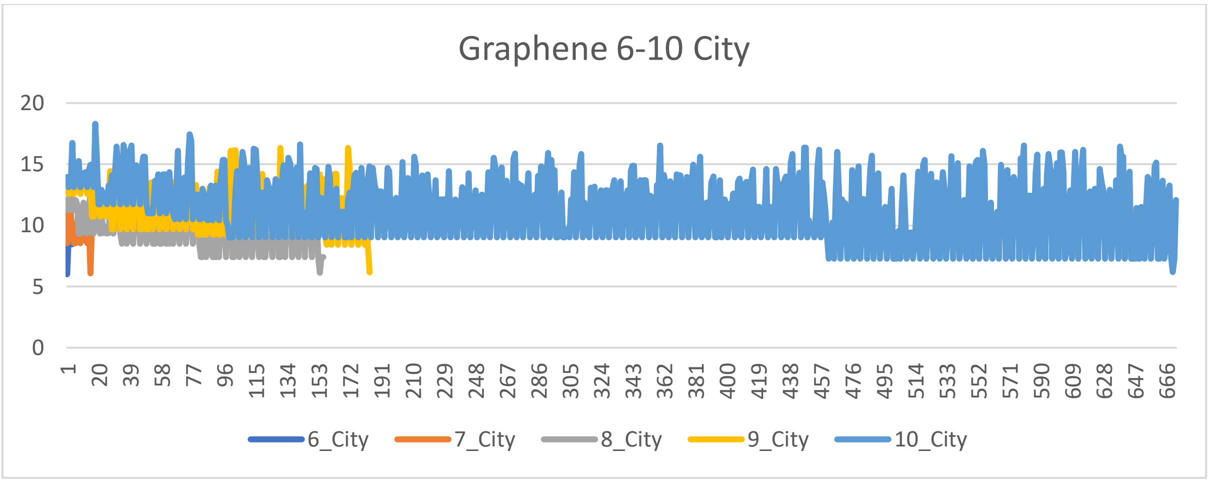


Figure 29 Graphene 6-10 City Good Run Evolved until Target of 6 Reached

Graphene TSP results for 6-9 Cities with a Random Seed of 1 and 2 to give an idea of how the random seed impacts on elapsed time as shown in Figure 30 & Figure 31 and to put these results into perspective, TSP results for Mixed Resistors are shown in Figure 32. Mixed resistors solve the 9 City TSP in 234 steps or about 46 generations whereas Graphene does it in 184 & 374 steps or 30 and 74 generations. Bearing in mind how temperamental Graphene is, even when wrapped in cling film, it degrades after a period, whereas mixed resistors see Figure 12, at the cost of a little soldering, in this case is a much better and more reliable option.

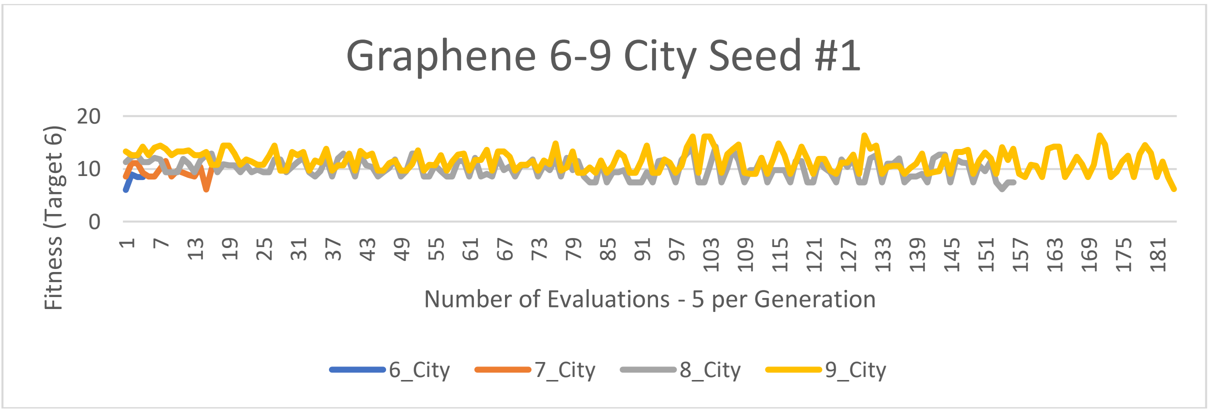


Figure 30 Graphene for 6-9 City with RAND Seed 1 Evolved until Target of 6 Reached

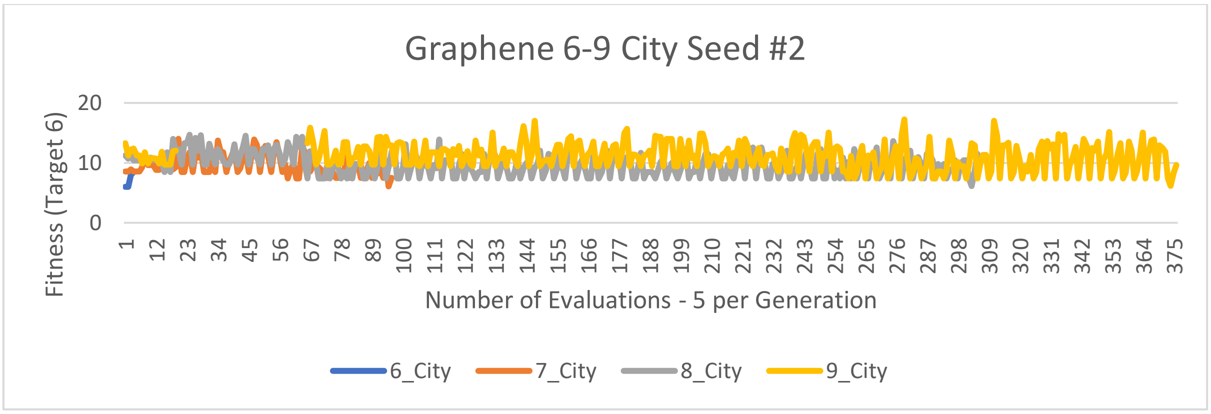


Figure 31 Graphene for 6-9 City with RAND Seed 2 Evolved until Fitness Target of 6 Reached

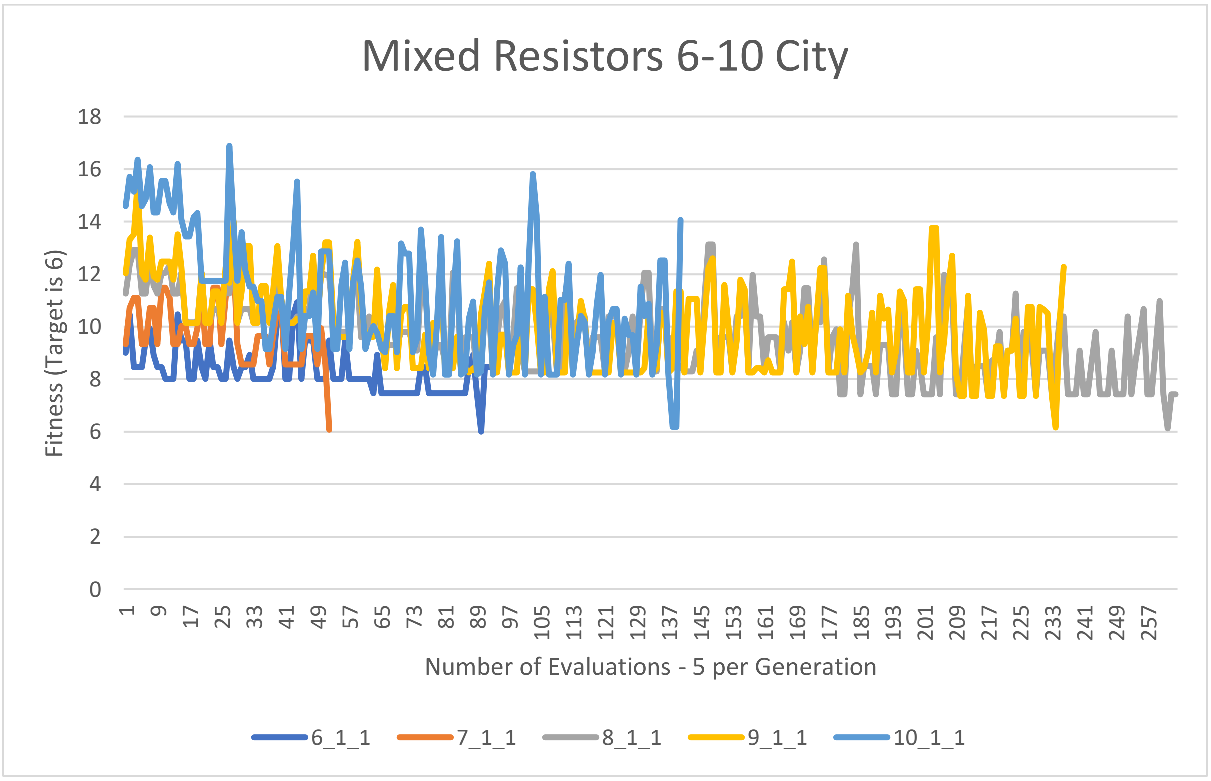


Figure 32 Mixed Resistors 6-10 City Evolved until Target of 6 Reached

9.2 Summary of Low-Cost EiM Results

The results of Low-Cost EiM is very similar to High-Cost EIM. The key message here is that EiM can be done as effectively on a Low-Cost platform and materials as it can on a High-

Cost Platform. There is little to differentiate the results obtained from the two platforms for graphene or mixed resistors.

Also, in this current work there is also little to differentiate graphene, carbon nanotubes and mixed resistors in terms of performance at solving TSPs. To be of long term use and value the degradation problems suffered by graphene either needs to be resolved by finding better ways of preparing the material or by investigating other formulations of graphene, to make it more resilient to use and consistent in performance over time.

10 Pi Further Experiments

10.1 An Evolved Electro-Magnetic Sensor

The author was inspired by Paul Layzell's work with the Evolvable Motherboard (22) and the evolution of novel sensors (23) (24). Investigating sensor evolution using this new Raspberry Pi platform made a lot of sense.

Various unsuccessful attempts were made to evolve sensitivity to a changing magnetic field using graphene in the DIP header as the medium as in Figure 5 How to Make a DIP Graphene Header.

The first attempt involved trying to see if evolution could be sensitive to a pulsing electromagnetic field at around 40Hz (40Hz was chosen because that was about as fast as was possible to continuously switch the Adafruit Featherlight Power Relay on and off). An apparently electro-magnetic sensitive montage quickly and very easily evolved, but then did not respond at all if the voltage and cross-point switch montage was loaded and the varying electromagnetic field was presented manually to the material. It transpired that somehow evolution had learned to detect the on and off signals being sent to the Adafruit Featherlight Power Relay[19]. The solenoid was isolated and moved to be under the control of a second Raspberry Pi, which was sent single ON and OFF controls via GPIO pins – the second Pi then turned the Relay ON and OFF at 40Hz. After isolating the relay to a second Pi and placing the relay in a metal box a few feet away, evolution was not able to find a solution. Note that care needs to be taken to ensure that what is thought to be evolving is the case and that evolution has not exploited or learnt something related but not what is not actually required.

Since a large part of this Thesis involved attempting to solve TSPs it made sense to attempt to use a TSP Solution montage of configuration volts and cross-point switch settings, which produced a successful TSP solution as the start point for evolving a Sensor.

[19] www.adafruit.com/product/3191

The EA was modified such that a TSP solution was evolved and then, that optimal solution was mutated. If the mutation still produced an optimal TSP solution, a varying magnetic field was applied to the material with the successful TSP montage of configuration volts and cross-point switch setting in place to see if the TSP solution would degrade. Note this approach uses the TSP solution as a form of Reservoir (25) "*uses computer based evolutionary algorithms to optimise a set of electrical control signals to induce reservoir properties within the substrate. In the training process, evolution decides the value of analogue control signals (voltages) and the location of inputs and outputs on the substrate that improve the performance of the subsequently trained reservoir readout*". In the current work the montage of electrical control signals and cross-point switch settings are evolved to solve a TSP. Then, that particular montage was evaluated again in the presence of a varying magnetic field, such that the gestalt of the software, hardware, material and montage becomes a reservoir.

Many unsuccessful attempts were made to evolve a magnetic sensor using graphene in the DIP Header.

Then it occurred to the Author to try and use standard resistors - Figure 12 Maplin Resistors Used as Evolvable Material. This proved to work very well and very quickly a magnetic sensor evolved using the set of mixed resistors. It is believed that the varying magnetic field was inducing small currents into the wires and materials in the resistors, which for whatever reason was not possible or perhaps the LTC2657 12bit DACs were not fine grained enough to generate sufficiently small increments of voltage to be able to place the graphene and 47k ohm resistors into a sufficiently unstable state, so that the varying magnetic field would produce an effect on the Reservoir.

Before a magnetic field was applied to the resistors or the material, a solution was evolved to a 6 or 7 City TSP see Figure 33, Figure 34 & Figure 35 below. Once the solution had evolved, to the optimum fitness of 6, the genome was mutated and if the result of the mutation was still an optimal TSP solution with a fitness of 6, then the material was tested with a 40Hz electromagnetic field. If the solution degraded under the electromagnetic field, then that montage of configuration voltage and cross-point switch was reapplied and tested 14 times, with and without the electromagnetic field, to verify that the initial degradation was not a one off and was repeatable.

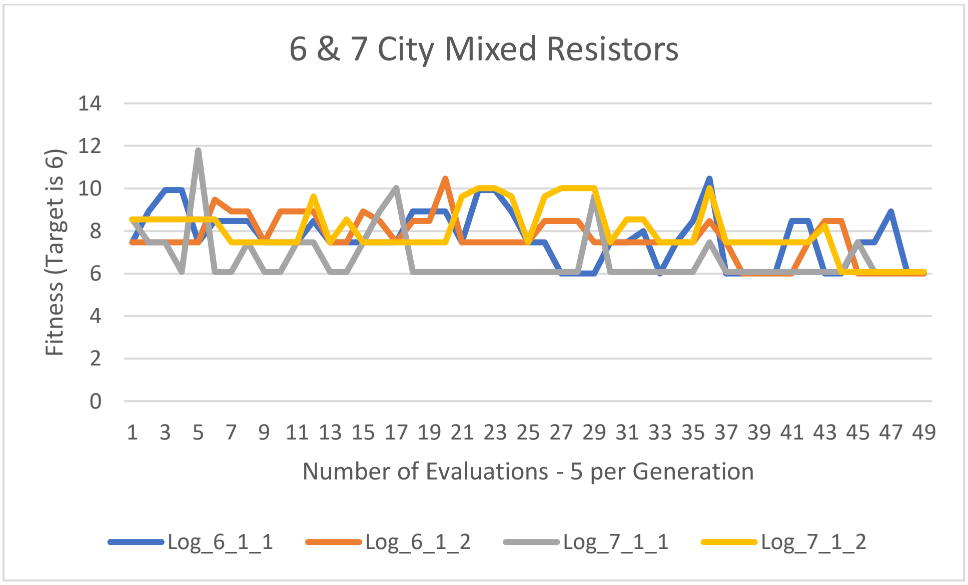


Figure 33 6 & 7 City Mixed Resistors - Evolved for a Fixed 10 Generations with a Fitness Target of 6

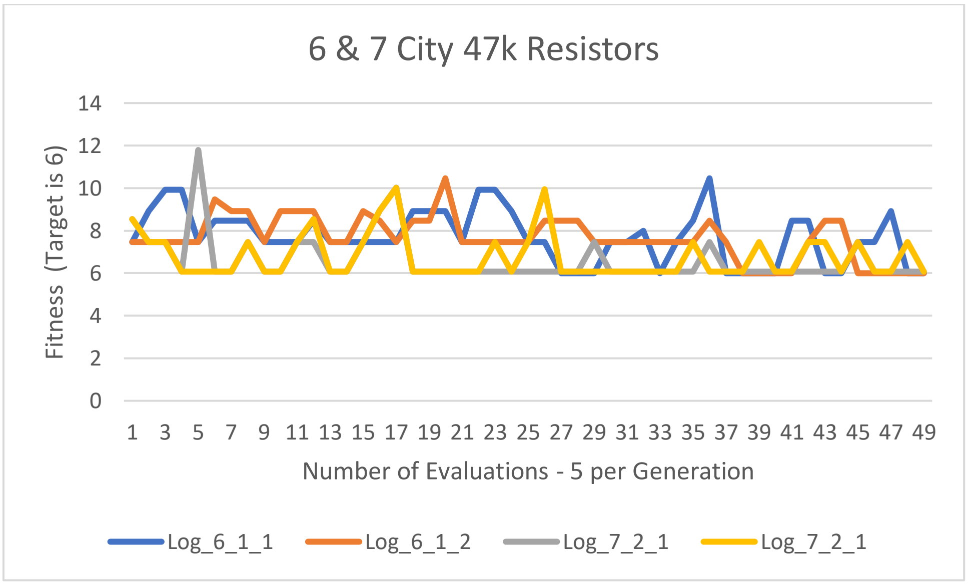


Figure 34 6 & 7 City 47k Ohm Resistors - Evolved for a Fixed 10 Generations with a Fitness Target of 6

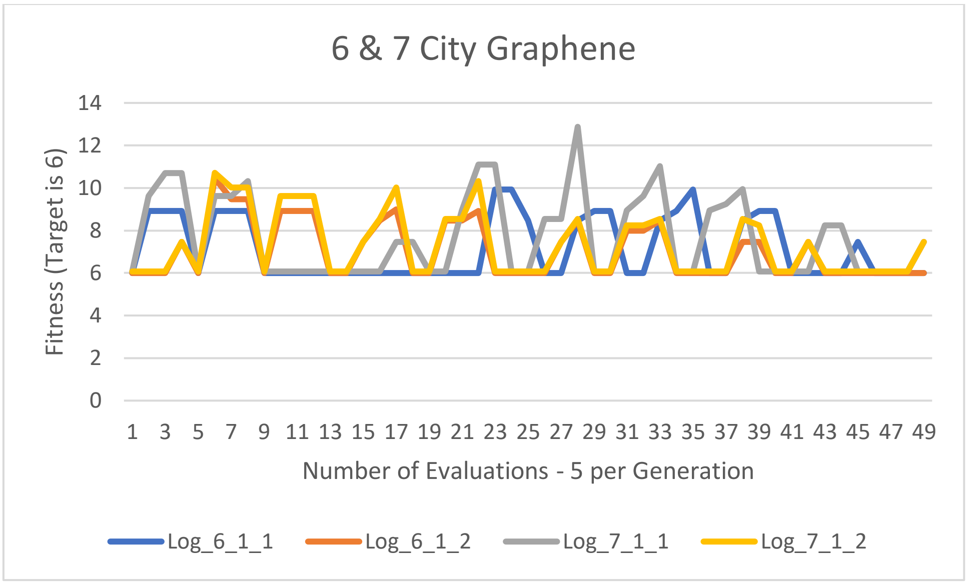


Figure 35 6 & 7 City Graphene - Evolved for a Fixed 10 Generations with a Fitness Target of 6

The only material tested in this work, for which it was possible to evolve a significant electromagnetic response within this environment was the mixed resistors Figure 12 . Figure 36 shows a range of responses from 57% correct to 71.4% correct.

Correct Without Electromagnet	Correct With Electromagnet	Total Tries	% Correct
14	5	28	67.9%
14	2	28	57.1%
14	6	28	71.4%
11	7	28	64.3%
14	3	28	60.7%
13	6	28	67.9%
14	3	28	60.7%

Figure 36 Results for Mixed Resistors With / Without Electromagnetic Stimulation

10.2 Summary of Pi Further Experiments

Lots of attempts were made to evolve sensitivity to a 40Hz electromagnetic field. A seeming initial success proved to be an error, whereby evolution had somehow learnt to detect interactions between GPIO and a GPIO controlled solenoid. When the solenoid was moved under the control of a second Raspberry Pi, no electromagnetic sensitivity could be evolved.

To improve the sensitivity of the environment it was decided to try using a montage of configuration voltages and cross-point switch settings of a successful TSP solution as a way of detecting the 40 Hz magnetic field. This eventually worked using a set of mixed resistors as the material, giving up to 70% detection accuracy of the presence or absence of the electromagnetic field. With further work this sensitivity could almost certainly be improved.

This work shows that with a little ingenuity, it is possible to evolve a sensor using a Low-Cost platform and Low-Cost materials.

11 Review

A High-Cost PC EA platform was built using Matlab, a PCI-6259 and custom printed circuit board, containing a cross-point switch a DIP socket for the material and some external connectors. These were all housed for portability in a toolbox. Graphene was chosen as the material and a way was devised of interfacing it to the printed circuit board. An EA was developed which configured the cross-point switch and controlled and operated the PCI-6259, enabling the material to be fed with configuration voltages and measurements taken from the material because of the supplied configuration voltages. The EA was successfully run many times to solve TSPs but then noise was found to be present in the cost function when re-evaluating elites. This was thought to be due to changing atmospheric conditions affecting the graphene and so the material was sealed in cling film which made the material more stable. Alternative materials were investigated including carbon nanotubes and standard resistors, which both proved to be similarly usable for solving TSP Problems.

A Low-Cost EA platform was built using a Raspberry Pi, Pycharm and a custom development board was designed and built containing the ADC/DACs and two cross-point switches. The EA was developed which configured the cross-point switches and operated the DACs and ADC as part of its EA processing. By its nature this platform took a lot more development effort because there was minimal experience anywhere of doing EiM work on a Pi with these types of devices. Once developed there was little in terms of results to differentiate the High and Low-Cost platforms or materials, apart from the long- term stability of graphene.

Considerable time was spent attempting to evolve sensitivity to an electromagnetic field but with some ingenuity a way was found of using optimal TSP solutions as start point for this work. For some reason only mixed resistors were successful in sensing electro-magnetic fields within the current environment.

Doing statistical analysis of many TSP training runs was considered as a way of detecting electro-magnetism, but because of the amount of noise produced during evolution e.g. Figure 24, Figure 27 and Figure 29, this was not followed up.

The author is delighted to have evolved sensitivity to an electro-magnetic field as there is great potential for such a Low-Cost platform to be able to be used for the development of a range of sensors and computation at the same time.

12 Bibliography

1. **Holland, J.** *Adaptation in Natural and Artificial Systems.* Cambridge Massachusetts : MIT Press, 1992.

2. **Mitchell, Melanie.** *An Introduction to Genetic Algorithms* . Cambridge MA : MIT Press, 1996.

3. **Goldberg, D.** *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, Massachusetts : Addison-Wesley, 1989.

4. **Thompson, Adrian.** *An evolved circuit, intrinsic in silicon, entwined with physics.* : Springer, 1996. pp. 390-405. Vol. Evolvable Systems: From Biology to Hardware.

5. —. *Hardware Evolution.* : Springer, 1998. Phd Thesis University of Sussex.

6. *Evolution in materio: Looking Beyond the Silicon Box.* **Julian F. Miller, Keith Downing.** : IEEE Computer Society , 2002, Vol. Proceeding of NASA/DoD Evolvable Hardware Workshop, pp. 167-176.

7. *Evolution in Materio: Exploiting the Physics of Materials for Computation.* **Simon Harding, Julian Miller, Edward A. Reitman.** York UK : IEEEE, 2008, Journal of Unconventional Computing, pp. 155--194.

8. **Harding, Simon.** *Evolution in Materio.* York UK : 2006. Phd Thesis University of York.

9. *Programmable Matter.* **Simon Harding, Julain F. Miller, Edward A. Reitman.** : IEEE Transactions on Nanotechnology, 2005, pp. 1964-1967.

10. *Evolution in materio: A tone discriminator in liquid crystal.* **Harding, S. & Miller, J. F.** York : 2004. Vol. Proceedings of the 2004 Congress on Evolutionary Computation, pp. 1800–1807.

11. *Travelling Salesman Problem Solved 'in materio' by Evolved Carbon Nanotube Device.* **Dr. Kester Dean Clegg, Dr. Julian Francis Miller.** York UK : Springer, 2014. Vols. Parallel Problem Solving from Nature – PPSN XIII, pp. 692-701. https://link.springer.com/chapter/10.1007/978-3-319-10762-2_68.

12. *Evolution-in-materio: solving computational problems using carbon nanotube-polymer composites.* **Maktuba Mohid, Julian F. Miller, Simon L. Harding, Gunnar Tufte, Mark K.**

Massey, Michael C. Petty. Issue 8, : Springer-Verlag, Soft Computing - A Fusion of Foundations, Methodologies and Applications, Vol 20, pp. 3007-3022 .

13. *Evolution-In-Materio: Solving Machine Learning Classification Problems Using Materials.* **Maktuba Mohid, Julian Francis Miller,Simon L. Harding,Gunnar Tufte, Odd Rune Lykkebø, Mark K. Massey, Michael C. Petty.** [ed.] Branke J., Filipič B., Smith J. Bartz-Beielstein T. York : Springer-Verlag, 2014. Lecture Notes in Computer Science, Vol 8672, pp. 721-730.

14. **E. Vissol-Gaudin, A. Kotsialos, M.K. Massey, C. Groves, C. Pearson, D.A. Zeze, M.C. Petty.** Solving Binnary Classification Problems with Carbon Nanotube / Liquid Crystal Composites and Evolutionary Algorithms. 2017, Vol. 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 1924-1931.

15. **E. Vissol-Gaudin, A Kotsialos, MK Massey, D. A. Zeze, C. Pearson, C. Groves, M. C. Petty.** Data Classification Using Carbon-Nanotubes and Evolutionary Algorithms. [ed.] Emma Hart, Peter R. Lewis Julia Handl. : Springer, 2016, Vols. Parallel Problem Solving from Nature – PPSN XIV, pp. 644-654.

16. *Training a Carbon-Nanotube/Liquid Crystal Data Classifier Using Evolutionary Algorithms.* **E. Vissol-Gaudin, A. Kotsialos, M. K. Massey ,D. A. Zeze,C. Pearson,C. Groves, M. C. Petty.** [ed.] A. Condon M. Amos. : Springer, Cham, Switzerland, 2016. 15th International Conference on Unconventional Computation and Natural Computation. pp. 130-141.

17. *Evolution of Electronic Circuits using Carbon Nanotube Composites.* **M. K. Massey, A. Kotsialos, D. Volpati, E. Vissol-Gaudin, C. Pearson, L. Bowen, B. Obara, D. A. Zeze, C. Groves, M. C. Petty.** Durham : Nature, 2016. Vols. Scientific Reports 6, Article number: 32197 .

18. *Pycharm Community Edition.* Czech Republic **: Jet Brains, 2017.**

19. *AD75019 16 x 16 Crosspoint Switch Array Data Sheet.* : **Analogue Devices.**

20. Linear, Technology. LTC2657 Octal I2C 16-/12-Bit Rail-to-Rail DAC Data Sheet. [Online] http://cds.linear.com/docs/en/datasheet/2657f.pdf**.**

21. —. *LTC2499 24-Bit 8-/16-Channel Delta Sigma ADC Data Sheet.* **: Linear Technology.**

22. *The 'Evolvable Motherboard' A Test Platform for the research of Intrinsic Hardware Evolution.* **Layzell, Paul.** Brighton Sussex : CSRP, 1998. Technical Report CSRP479. COGS University of Sussex**.**

23. *An Evolved Radio and its Implications for Modelling the Evolution of Novel sensors.* **J. Bird, P. Layzell. :** Proceedings of Congress on Evolutionary Computation, 1998. pp. 1836-1841.

24. *A new research tool for intrinsic hardware evolution.* **P, Layzell.** Berlin, Heidelberg : Springer-Verlag, 1998, Proceedings of the Second International Conference on Evolvable Systems, Vol 1478, pp. 47-56.

25. *Reservoir Computing in Materio: An Evaluation of Configuration through Evolution.* **Matthew Nicholas Dale, Susan Stepney, Julian Francis Miller, Martin Albrecht Trefzer.** Athens : 2016. Vol. IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1-8.