Evolution-In-Materio: Solving Machine Learning Classification Problems Using Materials

Maktuba Mohid[1], Julian Francis Miller[1], Simon L. Harding[2], Gunnar Tufte[2], Odd Rune Lykkebø[3], Mark K. Massey[3], and Michael C. Petty

[1] Department of Electronics, University of York, York, UK
{mm1159,julian.miller}@york.ac.uk, slh@evolutioninmaterio.com
[2] Department of Computer and Information Science,
Norwegian University of Science and Technology, 7491 Trondheim, Norway
{gunnart,lykkebo}@idi.ntnu.no
[3] School of Engineering and Computing Sciences and Centre for Molecular and Nanoscale Electronics, Durham University, UK
{m.k.massey,m.c.petty}@durham.ac.uk

Abstract. Evolution-in-materio (EIM) is a method that uses artificial evolution to exploit the properties of physical matter to solve computational problems without requiring a detailed understanding of such properties. EIM has so far been applied to very few computational problems. We show that using a purpose-built hardware platform called Mecobo, it is possible to evolve voltages and signals applied to physical materials to solve machine learning classification problems. This is the first time that EIM has been applied to such problems. We evaluate the approach on two standard datasets: Lenses and Iris. Comparing our technique with a well-known software-based evolutionary method indicates that EIM performs reasonably well. We suggest that EIM offers a promising new direction for evolutionary computation.

Keywords: Evolutionary algorithm, evolution-in-materio, material computation, evolvable hardware, machine learning, classification problem.

1 Introduction

Natural evolution could be viewed as an algorithm which exploits the physical properties of materials. Evolution-in-materio (EIM) aims to mimic the exploitation of physical properties by natural evolution by manipulating physical systems using computer controlled evolution (CCE) [6,10]. In particular, EIM aims to exploit the properties of physical systems for solving computational problems.

Evolution-in-materio was first described by Miller and Downing [10]. The concept was inspired by the work of Adrian Thompson who investigated whether it was possible to evolve working electronic circuits using a silicon chip called a Field Programmable Gate Array (FPGA). He evolved a digital circuit that could discriminate between 1kHz or 10kHz signal [12]. When the evolved circuit was analysed Thompson discovered that artificial evolution had exploited physical

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 721–730, 2014.

properties of the chip. To demonstrate that EIM was possible, Harding and Miller attempted to replicate these findings using a liquid crystal display. They found that computer-controlled evolution could utilize the physical properties of liquid crystal to help solve a number of computational problems [4]:

- Two input logic gates: OR, AND, NOR, NAND, etc. [6].
- Tone Discriminator: A device was evolved which could differentiate different frequencies [4].
- Robot Controller: A controller for a simulated robot with wall avoidance behavior [5].

In this paper, we describe the use of a purpose built platform called Mecobo that facilitates computer controlled evolution of a material (the hardware is described in detail in [8]). The Mecobo platform has been developed within an EU funded research project called NASCENCE [3]. The computational material we have used in this investigation is a mixture of single-walled carbon nanotubes and a polymer. Evolutionary computation has been widely used to solve machine learning classification problems. Here, we show that using the Mecobo platform it is possible to evolve solutions to two classification problems using materials. To form a basic assessment of effectiveness of the technique we have compared our results with a well-known software-based evolutionary computation technique called Cartesian Genetic Programming (CGP) [9] on the same problems.

The organisation of the paper is as follows. In Sect. 2 we give a brief conceptual overview of EIM. We describe the Mecobo EIM hardware platform in Sect. 3. The preparation and composition of the physical computational material is described in Sect. 4. Sect. 5 describes the machine learning classification problem. The way we have used the Mecobo platform for classification problem is described in Sect. 6. We describe our experiments and analysis of results in Sect. 7. Finally we conclude and offer suggestions for further investigation in Sect. 8.

2 Conceptual Overview Of Evolution-In-Materio

EIM is a hybrid system involving both a physical material and a digital computer. In the physical domain there is a material to which physical signals can be applied or measured. These signals are either input signals, output signals or configuration instructions. A computer controls the application of physical inputs applied to the material, the reading of physical signals from the material and the application to the material of other physical inputs known as physical configurations. A genotype of numerical data is held on the computer and is transformed into configuration instructions. The genotypes are subject to an evolutionary algorithm. Physical output signals are read from the material and converted to output data in the computer. A fitness value is obtained from the output data and supplied as a fitness of a genotype to the evolutionary algorithm [11]. Figure 1 shows conceptual overview of EIM.

Miller and Downing noted that only certain materials may be suitable for EIM and they gave some guidelines for choosing materials [10]. The material needs to

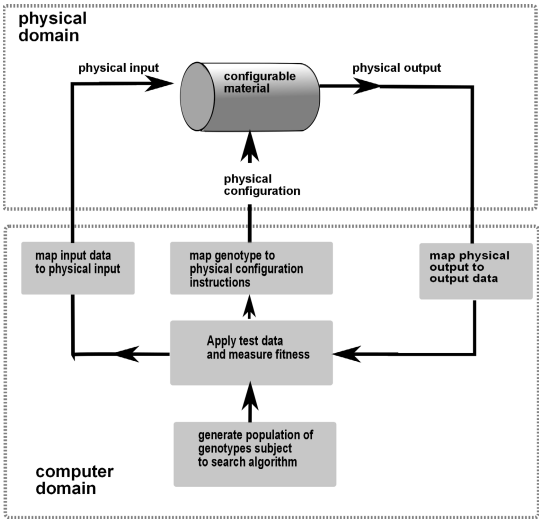

Fig. 1. Concept of evolution-in-materio [11]

be reconfigurable, i.e., it should be able to be evolved over many configurations to get the desired response. It is also important for a physical material to be able to be "reset" in some way before applying new input signals to it, otherwise it might preserve some memory of a past configuration and give fitness scores that are dependent on the past. Preferably, the material should also be able to be physically configured using small voltage and be manipulable at a molecular level.

3 Mecobo Hardware Platform

The hardware system we have used has three main components: a host computer, the Mecobo platform and an electrode array. The Mecobo platform is designed to interface a large variety of materials. The hardware allows the possibility to map inputs, outputs, configurations and signal properties in arbitrary ways. The platform's software components, i.e. the EA and the software stack, are as important as the hardware. Mecobo includes a flexible software platform including hardware drivers, support of multiple programming languages and the possibility to connect to hardware over the internet. This makes Mecobo a highly flexible platform for EIM experimentation [8].

Mecobo is built on PCBs with an FPGA as the main component. The digital and analogue parts of Mecobo are implemented on separate PCBs. All the analogue components are placed on a daughter board; such as crossbar switches and analogue-digital converters. This has the advantage that it allows the redesign of the analogue part of the system without changing the digital part of the motherboard. A micro controller stands as a communication interface between the FPGA and an external USB port.

At present the Mecobo hardware allows only two types of inputs to the material. One is constant voltage (0V or 3.5V) and the other is a square wave signal.

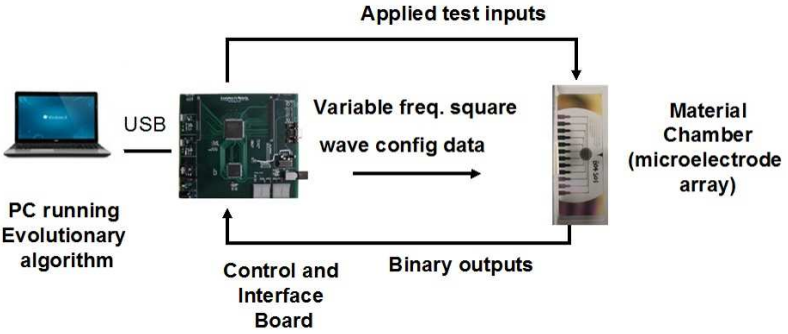

Fig. 2. Mecobo Hardware platform together with gold electrode array and material sample

Different characteristics or input parameters associated with these inputs can be chosen, these control the amplitude of an input, the square wave frequency, cycle time (percentage of period square wave is 1), phase, and the start and stop times of applied input. The start time and end time of each input signal determine how long an input is applied. Mecobo only samples using digital voltage thresholds, hence the material output is interpreted as strictly high or low, (i.e. 0 or 1). In later versions of this hardware, analogue inputs and outputs will be possible.

In the case that an electrode is chosen to be read (see section 4), a user-defined output sampling frequency determines the buffer size of output samples. If the output frequency is F_{out}, start time $Time_{start}$ and end time is $Time_{end}$, then the buffer size is Buf_{size} is given by:

$$Buf_{size} = F_{out}(Time_{end} - Time_{start})/1000 \quad (1)$$

Where, $Time_{start}$ and $Time_{end}$ are measured in milliseconds. However, in practice due to pin latency, the real buffer size is generally smaller.

4 Physical Computational Material

The experimental material consists of single-walled carbon nanotubes mixed with polymethyl methacrylate (PMMA) and dissolved in anisole (methoxybenzene) [1]. The sample is baked causing the anisole to evaporate. This results in material which is mixture of carbon nanotube and PMMA. The concentration of carbon nanotube is 0.71% (weight% fraction of PMMA). Carbon nanotubes are conducting or semi-conducting and role of the PMMA is to introduce insulating regions within the nanotube network, to create non-linear current versus voltage characteristics. Another benefit of the polymer is to help with dispersion of the nanotubes in solution. The preparation of experimental material is given below:

[1] Mark K. Massey and Michael C. Petty prepared the materials used as substrates and the electrode masks for our experiments.

- A M3 sized nylon washer was glued on the electrode array to contain the material whilst drying;
- 20 μL of material were dispensed into the washer;
- This was dried at $\approx 100^o$ C for $\approx$1 h to leave a "thick film".

The experimental material is placed in the middle of a plate of the electrode array. Twelve gold electrodes are connected directly with the experimental material in the plate. The electrode array is connected directly with the Mecobo board via wires. The electrode sample is shown in Fig. 2.

5 Machine Learning: Classification Problems

Classification is an important class of problems in Machine learning. The objective is to correctly classify data instances. In this paper we have evaluated our approach on two classification problems: Lenses and Iris. [2]. Both datasets have four attributes which are classified into one of three classes. The Lenses dataset consists of 24 instances with integer attributes. The attributes are categorical in nature and take values either 1,2 or 1, 2, 3. We used the first 16 instances as training data and the last 8 as testing data. The Iris dataset contains 150 instances with real-valued attributes. The first fifty instances are class 1, the second fifty class two and third set of 50 are class 3. We divided the data set into two groups (training and testing set) of 75 instances each. Each set contained exactly 25 instances of each class.

6 Classifying Data Using Evolution-In-Materio

6.1 Methodology

The experiments were performed with an electrode array having 12 electrodes. For both datasets, four electrodes have been used as inputs (i.e. they are attribute related), 3 electrodes have been used as outputs (i.e. to define the class) and the remaining 5 electrodes have been used for configuration voltages. Each output electrode is associated with an output class. Each chromosome defined which electrodes are either outputs, inputs (receive square waves) or receive the configuration data (square waves or constant voltage). We accumulated sampled output values in a buffer for 128 milliseconds using a 25KHz sampling frequency.

The fitness calculation in the evolutionary algorithm only used training data. Once the evolutionary algorithm finished the configuration of electrodes having the best fitness was subsequently tested with the test data to determine its ability to predict correctly unseen data (the test set).

6.2 Genotype Representation

Each chromosome used $n_e = 12$ electrodes at a time. Associated with each electrode there were six genes which define which electrode was used, or characteristics of the input applied to the electrode: signal type, amplitude, frequency,

Table 1. Description of genotype

Gene Symbol	Signal applied to, or read from i^{th} electrode	Allowed values
p_i	Which electrode is used	0, 1, 2 ... 11
s_i	Type	0 (constant), 1 (square-wave)
a_i	Amplitude	0 , 1
f_i	Frequency	500 ,501 ... 10K
ph_i	Phase	1, 2 ... 10
c_i	Cycle	0, 1, 2 ... 100

phase, cycle (see Sect. 3). This means that each chromosome required a total of 72 genes. Mutational offspring were created from a parent genotype by mutating a single gene (i.e., one gene of 72). The values that genes could take are shown in Table 1, where i takes values 0, 1, ... 11.
The genotype for a chromosome of an individual consists of the 72 genes shown below:

$p_0 s_0 a_0 f_0 ph_0 c_0 \ldots p_{11} s_{11} a_{11} f_{11} ph_{11} c_{11}$

6.3 Input Mapping

The inputs to the electrode array (representing the data instances) were square waves of a particular frequency. The frequency was determined by a linear mapping of attribute data. Denote the i^{th} attribute in a dataset by I_i, where i takes values $1, 2, 3, 4$. Denote the maximum and minimum value taken by this attribute in the whole data set by $I_{i_{max}}$ and $I_{i_{min}}$ respectively. Denote the maximum and minimum allowed frequencies be denoted by F_{max} and F_{min} respectively. Then the linear mapping given in Eqn. 2 allows the i^{th} attribute of an instance I_i to map to a square-wave frequency F_i which was applied to a given electrode. In the experiments we chose $F_{min} = 500$Hz and $F_{max} = 10000$ Hz.

$$F_i = a_i I_i + b_i \tag{2}$$

where the constants a_i and b_i are found by setting I_i and F_i to their respective maximum and minimum and solving for a_i and b_i.

$$a_i = (F_{max} - F_{min})/(I_{i_{max}} - I_{i_{min}}) \tag{3}$$

$$b_i = (F_{min} I_{i_{max}} - F_{max} I_{i_{min}})/(I_{i_{max}} - I_{i_{min}}) \tag{4}$$

6.4 Output Mapping

We determined the class that an instance belonged to, by examining the output buffers which contain samples taken from the output electrodes. The current Mecobo platform can only recognize binary values, so the output buffers contain a binary string. We used the *transitions* from 0 to 1 in the output buffers to define the class that an instance belonged to. For each output buffer, the positions of transitions were recorded and the gaps between consecutive transitions were

measured and an average calculated. A transition based fitness was used as it is frequency related. Since instance data determine the frequencies of applied signals, it seemed natural to use a method of reading output buffer bitstrings that is itself frequency related. An example of average gap calculation for an output electrode has been shown in Fig. 3.

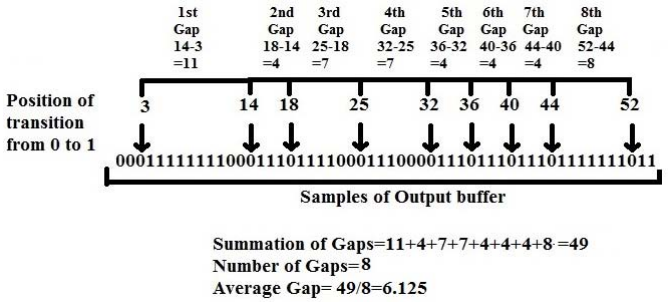

Fig. 3. Example of average transition gap calculation for an output electrode

The output class was determined by the output buffer with largest average transition gap. If two or more buffers had the same average gap then the class was determined by the first such buffer encountered (starting at 1).

6.5 Fitness Score

The fitness calculation required counts to be made of the number of true positives TP, true negatives TN, false positives, FP and false negatives, FN. For an instance having a class c, according to the dataset, and a predicted class p, we can calculate TP, TN, FP, and FN as shown in Eqn. 5.

$$\begin{aligned} &\textbf{if } p = c \textbf{ then } TP = TP + 1;\ TN = TN + 2 \\ &\textbf{if } p \neq c \textbf{ then } FP = FP + 1;\ FN = FN + 1;\ TN = TN + 1 \end{aligned} \tag{5}$$

The explanation of this is as follows. If the predicted p is correct, then it is a true positive so TP should be incremented. It is also a true negative for the other two classes, hence TN should be increased by two. If the predicted class is incorrect, then it is a false positive for the class predicted, so FP should be incremented. It is also a false negative for the actual class of the instance, so FN should be incremented. Finally, the remaining class is a true negative, so TN should be incremented. Once all instances have been classified we calculated the fitness of a genotype using Eqn. 6 [1].

$$fitness = \frac{TP.TN}{(TP + FP)(TN + FN)} \tag{6}$$

So if all instances are correctly predicted, the fitness is 1, since in this case $FP = 0$ and $FN = 0$. In the case that all instances are incorrectly predicted, then $TP = 0$ and $TN = 0$, so the fitness is zero.

7 Experiments

For each of the datasets a $1+\lambda-ES$ evolutionary algorithm with $\lambda = 4$ was used [9] and run for 500 generations. This evolutionary algorithm has a population size of $1 + \lambda$ and selects the genotype with the best fitness to be the parent of the new population. If there is no offspring better than the parent but at least one with a fitness equal to the parent, then an offspring is chosen to be the new parent. The remaining members of the population are formed by mutating the parent. Thirty and twenty independent runs were carried out for the Lenses dataset and Iris dataset respectively. The smaller number of runs for the latter was due to the large time required for each experiment. It took more than 12 hours to run 500 generations on the Iris training set. This time also precluded using leave-one-out cross validation methods.

7.1 Using CGP For Classification

To evaluate the effectiveness of the EIM method for solving classification problems we compared results with those obtainable using CGP using the same $1+4$ evolutionary algorithm over the same number of generations using the same fitness function. It should be noted that CGP has previously been shown to perform well on classification problems (e.g. with Mars terrain images [7] and mammograms [13]). CGP is a graph-based form of genetic programming [9]. The genotypes encode directed acyclic graphs and the genes are integers that represent where nodes get their data, what operations nodes perform on the data, and where the output data required by the user is to be obtained. In classification problems the number of outputs, n_O is chosen to be equal to the number of classes in the dataset. The class of a data instance is defined as the class indicated by the maximum numerical output. The function set chosen for this study was defined over the real-valued interval [0.0, 1.0] and consisted of the following primitive functions of their inputs. The functions were assumed to have three inputs, z_0, z_1, z_2 (but some are ignored):

$(z_0 + z_1)/2$; $(z_0 - z_1)/2$; $z_0 z_1$;
if $|z_1| < 10^{-10}$ **then** 1 **else if** $|z_1| > |z_0|$ **then** z_0/z_1 **else** z_1/z_0;
if $z_0 > z_1$ **then** $z_2/2$ **else** $1 - z_2/2$.

We used *three* mutation parameters. A percentage for mutating connections, μ_c and functions, μ_f. Mutation of outputs μ_o, is done probabilistically. In all experiments $\mu_c = 3\%$, $\mu_f = 3\%$, and $\mu_o = 0.5$. The output mutation probability was set as 0.5 because there are only as many outputs as there are classes. We chose a linear CGP geometry by setting the number of rows, $n_r = 1$ and the number of columns, $n_c = 100$ with nodes being allowed to connect to any previous node.

Table 2. Experimental results comparing results of experimental material with CGP on machine learning classification problem using two datasets: Lenses and Iris. Accuracy is the percentage of the training or test set correctly predicted.

Dataset	Average Training Accuracy of Experimental Material	Average Test Accuracy of Experimental Material	Best Accuracy of Experimental Material	Average Training Accuracy of CGP	Average Test Accuracy of CGP	Best Accuracy of CGP
Lenses	92.7%	65.8%	95.8%	93.8%	68.3%	95.8%
Iris	84.7%	77.1%	96.7%	97.7%	93.6%	98.0%

7.2 Results and Discussion

It can be seen from Table 2 that in the case of the Lenses dataset the training and testing of experimental material are very close to the corresponding accuracies of CGP and best accuracy of experimental material is same as that of CGP. In the case of the Iris dataset, although the results with training and test for the experimental material are not as good as CGP, the best accuracy is quite close.

8 Conclusions and Future Outlook

We have shown how using a purpose-built evolutionary platform called Mecobo, we can evolve configurations of a physical system to perform classification. The material we have used is a mixture of single-walled carbon nanotubes and a polymer. The aim of the paper is not to show that the experimental results of solving machine learning classification problems using EIM is competitive with state-of-the-art machine learning classification algorithms, but rather to start a new beginning in the world of computation. To our knowledge, this is the first time that classification problems have been attempted by the manipulation of a physical material. There were many decisions that were made in this investigation that require more detailed experiments before the ideal experimental conditions can be ascertained. This implies that it is likely that much better results could be obtained in the future. Increasing the number of electrodes could allow us to consider more instances or instances with more attributes, this could make the system faster and scale up to larger problems. Circuitry could be potentially built that allows the electrode array and material sample to act as a standalone classifier (i.e. no PC, or Mecobo board). There remain many questions for the future. The Mecobo platform is currently under development and the next version will be able to allow the utilization of analogue voltages.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 317662.

References

1. Akbarzadeh, V., Sadeghian, A., dos Santos, M.: Derivation of relational fuzzy classification rules using evolutionary computation. In: IEEE Int. Conf. on Fuzzy Systems, pp. 1689–1693 (2008)
2. Bache, K., Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
3. Broersma, H., Gomez, F., Miller, J.F., Petty, M., Tufte, G.: Nascence project: Nanoscale engineering for novel computation using evolution. International Journal of Unconventional Computing 8(4), 313–317 (2012)
4. Harding, S., Miller, J.F.: Evolution in materio: A tone discriminator in liquid crystal. In: Proc. Congress on Evolutionary Computation 2004, vol. 2, pp. 1800–1807 (2004)
5. Harding, S., Miller, J.F.: Evolution in materio: A real time robot controller in liquid crystal. In: Proc. NASA/DoD Conference on Evolvable Hardware, pp. 229–238 (2005)
6. Harding, S.L., Miller, J.F.: Evolution in materio: Evolving logic gates in liquid crystal. Int. J. of Unconventional Computing 3(4), 243–257 (2007)
7. Leitner, J., Harding, S., Forster, A., Schmidhuber, J.: Mars terrain image classification using cartesian genetic programming. In: 11th International Symposium on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS (2012)
8. Lykkebø, O.R., Harding, S., Tufte, G., Miller, J.F.: Mecobo: A Hardware and Software Platform for In Materio Evolution. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 267–279. Springer, Heidelberg (2014), http://dx.doi.org/10.1007/978-3-319-08123-6_22
9. Miller, J.F. (ed.): Cartesian Genetic Programming. Springer (2011)
10. Miller, J.F., Downing, K.: Evolution in materio: Looking beyond the silicon box. In: Stoica, A., Lohn, J., Katz, R., Keymeulen, D., Zebulum, R.S. (eds.) The 2002 NASA/DoD Conference on Evolvable Hardware, vol. 7, pp. 167–176. IEEE Computer Society (2002)
11. Miller, J.F., Harding, S.L., Tufte, G.: Evolution-in-materio: evolving computation in materials. Evolutionary Intelligence 7, 49–67 (2014)
12. Thompson, A.: Hardware Evolution - Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution. Springer (1998)
13. Völk, K., Miller, J.F., Smith, S.L.: Multiple network CGP for the classification of mammograms. In: Giacobini, M., et al. (eds.) EvoWorkshops 2009. LNCS, vol. 5484, pp. 405–413. Springer, Heidelberg (2009)