Evolution-In-Materio: Solving Bin Packing Problems Using Materials

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

[1]Department of Electronics, University of York, York, UK. Emails: [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. Emails: [gunnart, lykkebo]@idi.ntnu.no

[3]School of Engineering and Computing Sciences and Centre for Molecular and Nanoscale Electronics, Durham University, UK. Emails: [m.k.massey, m.c.petty]@durham.ac.uk

***Abstract*—Evolution-in-materio (EIM) is a form of intrinsic evolution in which evolutionary algorithms are allowed to manipulate physical variables that are applied to materials. This method aims to configure materials so that they solve computational problems without requiring a detailed understanding of the properties of the materials. The concept gained attention through the work of Adrian Thompson who in 1996 showed that evolution could be used to design circuits in FPGAS that exploited the physical properties of the underlying silicon [21]. In this paper, we show that using a purpose-built hardware platform called Mecobo, we can solve computational problems by evolving voltages, signals and the way they are applied to a microelectrode array with a chamber containing single-walled carbon nanotubes and a polymer. Here we demonstrate for the *first time* that this methodology can be applied to the well-known computational problem of bin packing. Results on benchmark problems show that the technique can obtain results reasonably close to the known global optima. This suggests that EIM is a promising method for configuring materials to carry out useful computation.**

I. Introduction

Natural evolution can be looked at as blind algorithm that exploits the physical properties of materials (i.e. proteins). This is Dawkin's so-called "blind watchmaker" [5]. There is no designer who constructs living systems from well-understood components, or who carefully manipulates physical materials through an understanding of the resident physics. This method of bottom-up construction is markedly different from the way human beings design useful artefacts. Human designers refine useful building blocks and combine these in highly constrained ways to arrive at solutions to problems. This process requires a deep understanding of the physical properties of the essential components.

Evolution-in-materio (EIM) [14] attempts to mimic the blind bottom-up process of natural evolution by exploiting the properties of physical systems through their manipulation by computer controlled evolution (CCE) [8], [9], [10]. Although EIM has been used in a number of ways to produce configurations of materials for various purposes most published work aims to manipulate physical systems for solving computational problems [15].

EIM was inspired by the work of Adrian Thompson who investigated whether it was possible for unconstrained evolution to evolve electronic circuits using a Field Programmable Gate Array (FPGA). He attempted to evolve a digital circuit that could discriminate between 1kHz or 10kHz signal [22]. However, when the evolved circuit was analysed, he discovered that artificial evolution had managed to exploit the physical properties of the chip. In other words it was working at an analogue level. Harding and Miller followed up this work by showing that EIM could also exploit the electrical properties of liquid crystal to solve computational problems. They found it was possible to evolve configurations of a liquid crystal display to solve a number of computational problems [6]:

- Two input logic gates: OR, AND, NOR, NAND, etc. [9].
- Tone Discriminator: A device was evolved which could differentiate different frequencies [6].
- Robot Controller: A controller for a simulated robot with wall avoidance behavior [7].

In this paper, we use a platform called Mecobo that was designed and constructed to facilitate computer controlled evolution of a material [12] for solving computational problems. The Mecobo platform has been developed within an EU funded research project called NASCENCE [2]. The computational material we have used in this investigation is a mixture of single-walled carbon nanotubes and a polymer.

In particular, we show that using the Mecobo platform it is possible to evolve solutions to instances of the well-known NP-hard combinatorial problem known as bin packing. This is the first time EIM has been used to solve bin packing problem. It is not our intention here to demonstrate that EIM using carbon nanotubes can yet compete with established methods for solving bin-packing problems. We are simply showing that EIM can be applied to such a benchmark problems and to evaluate various aspects of EIM using the Mecobo platform. For instance, what type of signals are appropriate,

what materials give the best results. Using materials in the genotype-phenotype map has, at present, some drawbacks as at present the actions of generating input signals and recording outputs is fairly slow (see later), this means that we can only feasibly evaluate relatively few potential solutions. However, it is an entirely new approach to the solution of computational problems and in time it could may offer advantages over conventional computational methods [15].

The organization of the paper is as follows. In Sect. II we give a short conceptual overview of EIM. The Mecobo EIM hardware platform in described in Sect. III. The preparation and composition of the physical computational material is described in Sect. IV. Sect. V describes the bin packing problem and benchmark datasets which are used in the experiment. The way we have used the Mecobo platform for bin packing is described in Sect. VI. We describe our experiments and analysis of results in Sect. VII. Finally we conclude and offer suggestions for further investigation in Sect. VIII.

II. Conceptual Overview Of Evolution-In-Materio

EIM uses a hybrid analogue-digital 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. Such signals are either input signals, output signals or configuration instructions. A computer is used to control 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. This is then subjected to an evolutionary algorithm. Physical output signals are read from the material and via a suitable mapping 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 [15]. The conceptual overview of EIM has been shown in figure 1.

EIM uses a very indirect genotype-phenotype mapping which may utilise aspects of the physical world that may not even be understood. The essential idea behind this is that it may increase the evolvability of evolutionary algorithms by giving them more freedom to exploit physical effects and use them to solve computational problems. Software-only genotype-phenotype mappings are necessarily constrained which contrasts strongly with natural evolution which operates in a physical world and directly exploits the physical properties of materials (mainly proteins) [1]. Despite this, there have been very few attempts to date to include materials in the evolutionary process.

One of the important questions in EIM concerns what properties material should have to make them most suitable. Miller and Downing suggested some guidelines for choosing such materials: The material needs to be reconfigurable, i.e., it can be evolved over many configurations to get desired response. It is important for a physical material to be able to be "reset" in some way before applying new input signals on it, otherwise it might preserve some memory and might

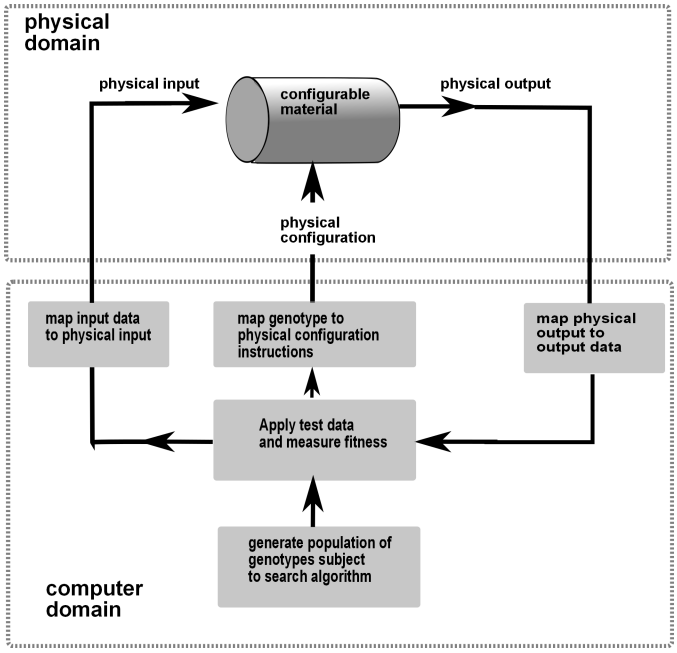


Fig. 1: Concept of evolution-in-materio [15].

give fitness scores that are dependent on the past behaviour. Preferably the material should be physically configured using small voltage and be manipulable at a molecular level [14], [15]. The NASCENCE project follows up some of these suggestions by using microelectrode arrays on which tiny chambers are mounted containing a variety of materials [2].

III. Mecobo: an Evolution-In-Materio Hardware Platform

The Mecobo hardware platform is a EIM system that has been designed and built within an EU-funded research project called NASCENCE [2]. It allows the possibility to map input, output and configuration terminals, signal properties and output monitoring capabilities in many ways. The platform's software component, i.e. EA and software stack, is 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 it over the internet [12].

In EIM the computational substrate is piece of material and appropriate physical variables that allow it to be manipulated by evolution are likely to be poorly understood (see Fig 1). This means that the selection of signal types, i.e. inputs, outputs and configuration data, assignment to I/O ports could easily not relate to material specific properties. Thus interactions with the materials should be as unconstrained as possible. This means that any I/O port should be allowed by the hardware to accept any signal type. In addition, the signal properties, e.g. voltage/current levels, AC, DC, pulse or frequency, should be allowed to be chosen during evolution. The Mecobo hardware interface is designed to handle all these

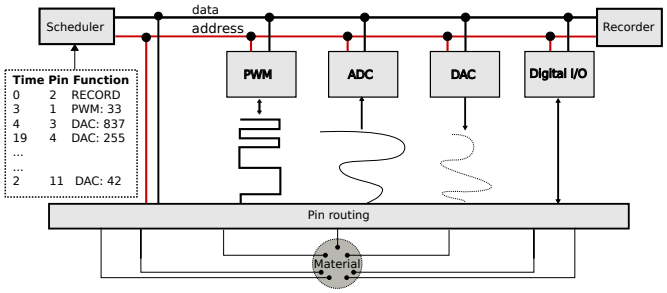


Fig. 2: Overview of the complete system.

features. Many computational problems require input data so Mecobo has been designed to allow user-defined external input data signals.

Figure 2 gives overview of the hardware interface. The figure shows an example set up in the dotted box. A genome defines pin 2 to be the output terminal, pin 1 to be the data input and pin 3 - 12 to be configuration signals. The architecture is controlled using a scheduler which controls a number of modules. Digital I/O can output digital signals and sample responses. Analogue output signals can be produced by the DAC module. The DAC can be configured to output static voltages or any arbitrary time dependent waveform. Sampling of analogue waveforms from the material is performed by the ADC. Pulse Width Modulated (PWM) signals are produced by the PWM module.

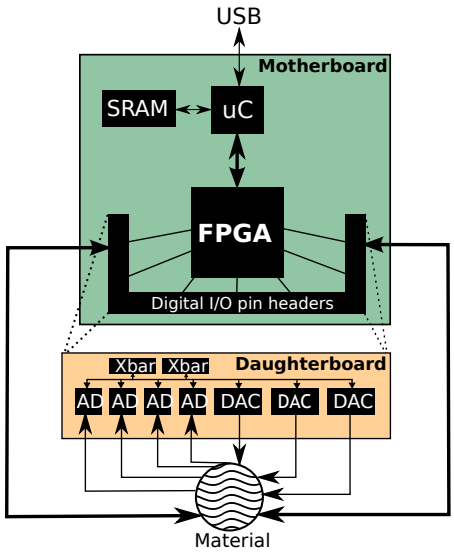
The system's scheduler can set up the system to apply and sample signals statically or produce time scheduled configurations of stimuli/response. The recorder stores samples, digital discrete values, time dependent bit strings, sampled analogue discrete values or time dependent analogue waveforms. Note that the recorder can include any combination of these signals.

In the interface all signals pass through a crossbar switch, i.e. pin routing. Pin routing is placed between the signal generator modules and the sampling buffer (PWM, ADC, DAC, Digital I/O and Recorder) making it possible to configure any terminal of a material to be input, output or receive configuration signals.

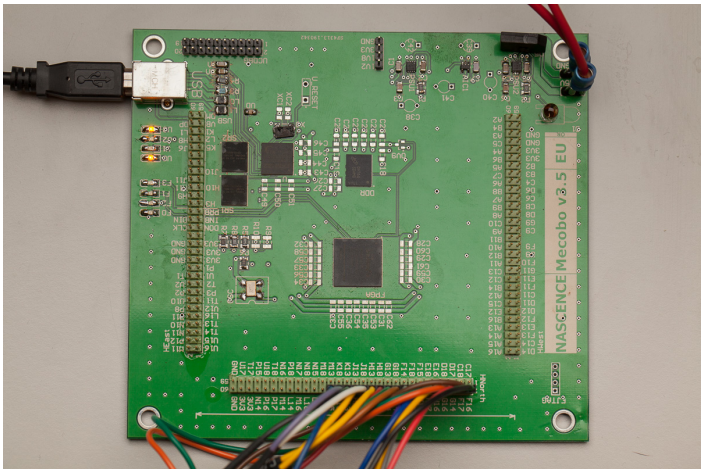
The material signal interface presented in Figure 2 is designed to be very flexible. It not only allows the possibility to evolve the I/O terminal placement but also a large variety of configuration signals are available to support materials with different sensitivity, from static signals to time dependent digital functions. At present, the response from materials can be sampled as purely static digital signals, digital pulse trains. The next version of Mecobo will allow the direct input and output of analogue signals. Further the scheduler can schedule time slots for different stimuli when time dependent functions are targeted or to compensate for configuration delay, i.e. when materials need time to settle before a reliable computation can be observed.

A. *Hardware implementation*

A block diagram of the hardware implementation of the interface is seen in figure 3(a). The Mecobo PCB has an FPGA as a key component. The system shown in Figure 2 is part of the FPGA design together with communication modules interfacing a micro controller and shared memory. The digital and analogue designs have been deliberately split into two (see Figure 3(a)). All analogue components such as crossbar switches and analogue-digital converters are placed on a daughter board. This allows the analogue part to be re-designed without changing the digital part of the motherboard. Figure 3(a) shows an example of the current system. The micro controller serves as a communication interface between the FPGA and the external USB port.



(a) Mecobo block diagram.



(b) Picture of Mecobo.

Fig. 3: Hardware interface implementation overview.

Figure 3(b) shows the motherboard with the Xilinx LX45 FPGA, Silicon Labs ARM based EFM32GG990 micro controller connected to a 12 terminal material sample.

Currently, the Mecobo hardware allows only two types of inputs to the material. It must be either a constant voltage (0V or 3.5V) or a square wave signal. However, different characteristics or input parameters associated with these inputs can be selected. These input parameters are described in Table I.

TABLE I: Adjustable Mecobo input parameters.

Parameter Name	**Description**	**Note**
Amplitude	0 or 1 corresponding to 0V or 3.5V	wave signal amplitude must be 1
Frequency	Frequency of square wave signal	Irrelevant if fixed voltage input
Cycle Time	Percentage of period for which square wave signal is 1	Irrelevant if fixed voltage input
Phase	Phase of square wave signal	Irrelevant if fixed voltage input
Start time	Start time of applying voltage to electrodes	Measured in milliseconds.
End time	End time of applying voltage to electrodes	Measured in milliseconds.

The user can choose the duration of input signals by varying the start time and end time. Mecobo only samples using digital voltage thresholds, hence the material output is strictly high or low, (i.e. 0 or 1). Later versions will allow analogue inputs and outputs.

When reading from an electrode the user must choose an output sampling frequency which determines the size of a buffer containing output samples. If the output frequency is F_{out}, and the start time $Time_{start}$ and end time is $Time_{end}$, are defined in milliseconds then the buffer size is Buf_{size} is given by:

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

In practice however, due to pin latency, real buffer sizes are usually smaller.

IV. Description Of Physical Computational Material

For our experiments we used a material consisting of single-walled carbon nanotubes (SWCNTs) mixed with poly(methyl methacrylate) (PMMA), dissolved in Anisole (methoxybenzene) [1]. This mixture is mixed using an ultrasonic homogenizer and then a small volume (20 μL) is spread on a gold microelectrode array. The sample is baked to evaporate the solvent. This leaves behind a film of SWCNT and PMMA. The concentration of SWCNT is 0.71 % (expressed as a weight % fraction of the PMMA).

Fig. 4: Electrode array with sample.

Carbon nanotubes, as grown, contain a mixture of semiconducting and metallic species. The role of the PMMA in these composites is to introduce insulating regions within the SWCNT network, creating non-linear current-voltage characteristics. It was thought that this might show some interesting computational behaviour. Another benefit of the polymer is to help with dispersion of the nanotubes in solution. The process of preparing experimental material is given below:

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

The electrode array has 12 electrode tracks, connected to the SWCNT/PMMA material and is connected directly with the Mecobo board via wires. The electrode sample is shown in Fig. 4.

V. Bin Packing

Bin-packing is a well-studied NP hard problem [4]. In the bin packing problem, each item, n_i with weight w_i from a total number of items, n_o, have to be placed in bins. Each bin however has a maximum weight capacity c_j. The objective is to place all the items in the least number of bins such that no bin has its weight limit exceeded [11].

Scholl and Klein have collected bin-packing benchmarks [19]. The datasets are in three classes, according to difficulty. In our experiments we have chosen 4 instances from each difficulty class. The best result for each dataset has been obtained by Scholl et al [20] using an algorithm called BISON which combines a successful heuristic meta-strategy tabu search and a branch and bound procedure.

VI. Solving Bin Packing Problems Using Evolution-In-Materio

A. Methodology

The experiments were performed with an electrode array having twelve electrodes. Bin-packing problems require no

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

inputs. The total number of outputs must equal the number of items that need to be packed into bins. Since bin-packing problems typically have 50 or more items multiple chromosomes must be used. Each chromosome defines a number of configuration signals to the electrode array and the remaining outputs supply recorded values in output buffers. The number of chromosomes required is given by the number of items to be packed into bins divided by the number of outputs chosen. For instance for a problem with 50 items, if two outputs are chosen the genotype requires 25 chromosomes. Thus in this case, there will be 10 configuration signals applied for each chromosome processed. We investigated various numbers of outputs and configuration signals using one of the hard bin-packing benchmarks called HARD0 [19].

We read a series of output values (0 or 1) from a buffer of samples taken from output electrode(s). The output values read from electrode(s) were linearly mapped between values -1.0 to 1.0. These values were then used to define the index of the bin (bin_i) in which the object i of the bin packing problem will be placed. So, the number of total number of outputs must be equal to the total number of objects (n_o). If a bin packing problem has more items than the number of output electrodes of the electrode array, then more chromosomes are needed to supply the outputs needed. We refer to this as a split genotype technique.

Using the Mecobo platform we can control the time that a signal is applied to the material (see Sect. III). Here, we accumulated output values in a buffer for 128 milliseconds using 25KHz sampling frequency. This gives a buffer size of 3,200 bits.

B. Genotype Representation

Each chromosome used $n_e = 12$ electrodes at a time. Associated with each electrode there were six genes which either define which electrode(s) was used as an output(s), or characteristics of the input applied to the electrode: signal type, amplitude, frequency, phase, cycle (see Sect. III). So each chromosome requires a total of 72 genes and a genotype of 50 chromosomes having electrode combination of 2 electrodes as outputs and 10 as configuration signals requires a total of 1800 (72x25) genes. Different numbers of mutations (m_n) were used with HARD0 benchmark to identify a number of mutations that gave good results. We investigated two values for m_n: $m_n = 1$ and $m_n = 2$.

The values that genes could take are shown in Table II. The chromosome index, i takes values $0, 1, \ldots d-1$, where d is the number of chromosomes and the electrode index, j takes values $0, 1, \ldots n_e - 1$.
The i^{th} chromosome, C_i is defined by:

$$C_i = p_{i,0}s_{i,0}a_{i,0}f_{i,0}ph_{i,0}c_{i,0} \ldots p_{i,11}s_{i,11}a_{i,11}f_{i,11}ph_{i,11}c_{i,11}$$

The genotype for a problem having d chromosomes is given by: $C_0C_1 \ldots C_{d-1}$

Since the electrode array we have been using has only twelve electrodes, we need a strategy for obtaining the required number of outputs from the device, so that we can handle the number of required values defined by the computational problem we are trying to solve. We do this by evolving a genotype consisting of many chromosomes. Each chromosome defines the configuration signals for the device and how many outputs will be used. In bin-packing there can be many items that have to be packed into bins. For instance, assuming we have 500 items and we use two outputs per chromosome, then we require 250 chromosomes. Since each chromosome has 72 genes, this means the total number of genes in this case is 250*72=18,000 genes!

TABLE II: Description of genotype.

Gene Symbol	**Signal applied to, or read from i^{th} chromosome and j^{th} electrode**	**Allowed values**
$p_{i,j}$	Which electrode is used	0, 1, 2 ... 11
$s_{i,j}$	Type	0 (constant) or 1(square-wave)
$a_{i,j}$	Amplitude	0 , 1
$f_{i,j}$	Frequency	500 ,501 ... 10K
$ph_{i,j}$	Phase	1, 2 ... 10
$c_{i,j}$	Cycle	0, 1, ... 100

For solution with 1 output and 11 configuration voltages, the last 6 gene values of i^{th} chromosome are related to the output. These are: $p_{i,11}s_{i,11}a_{i,11}f_{i,11}ph_{i,11}c_{i,11}$

In these output genes, only the first $p_{i,11}$ has any effect, the remainder are redundant. The gene $p_{i,11}$ decides which electrode will be used for the output of the device. Thus, mutations in this gene can choose a different electrode to be used as an output. The user can choose how many outputs and configuration signals will be used. For instance, when two outputs are used there are ten configuration signals and last 12 gene values are used for output.

C. Output Mapping

To determine a real-valued output from a collection of ones in an output buffer it was decided to use the fraction of ones. We chose this purely for simplicity and it is possible that other mappings of bits in the buffer to a real number could have worked better. However, initial findings revealed that the output buffer never contained more than 40% ones. As a result, before the bin packing experiment, an initial evolutionary investigation was performed to discover the typical contents of an output buffer under various conditions. The fraction of number of ones in the output buffer was calculated to obtain the values of outputs required to solve bin packing problems. However, because the buffer contained a maximum of 40% ones, the fraction of ones was multiplied by 2.5 so that a real-valued output would take values between 0 and 1. We denote this value for the buffer, i by q_i.

The values q_i were linearly mapped to values, x_i in the interval [-1.0, 1.0] using Eqn. 2.

$$x_i = -1.0 + 2.0q_i \tag{2}$$

The linearly mapped output values x_i corresponding to each chromosome were used to decide the bin index, bin_i which denotes which bin item, i will be placed in.

$$bin_i = \left\lfloor n_o \frac{(x_i + 1.0)}{2 + \epsilon} \right\rfloor \quad (3)$$

Assuming the number of items is n_o, the bin index is given by Eqn. 3. The floor function $\lfloor z \rfloor$ returns the nearest integer less than or equal to its argument, z. $epsilon$ is a very small positive quantity. Essentially, Eqn 3 divides the interval [-1, 1] into n_o equal intervals corresponding to bins, so that the mapped output values decide which bin an item will be placed in.

For instance, assuming the number of items, $n_o = 50$ if x_i is -1.0, bin_i is 0, and if x_i is 1.0, bin_i is 49.

D. Fitness Calculation

Fitness for each individual of population is calculated as follows.

If for an individual there is at least one bin filled above its capacity the fitness is calculated using following equation:

$$fitness = \sum_{j=1}^{j=n_{ob}} (c_j - b_j) \quad (4)$$

Where c_j is bin capacity of each bin, b_j is total weight of overflowing bin, i.e., the summation of weights of all objects placed in that bin, n_{ob} is the number of bins that have exceeded their capacity. Note that the fitness is always negative when there are any bins that are filled beyond their capacity.

However, if there are no bins overflowing:

$$fitness = n_{ub} \quad (5)$$

Where n_{ub} denotes the number of unused bins.

Thus a fitness less than zero indicates at least one bin is over capacity. If all bins are used and none overflow then the fitness is zero. If no bins overflow and some bins are unused then the fitness is a positive value equal to the number of unused bins. Thus maximization of fitness drives genotypes towards representing the smallest number of non-overflowing bins.

VII. Experiments

A $1 + \lambda - ES$, evolutionary algorithm with $\lambda = 4$ was used [13] and run for 5000 generations. The $1 + \lambda - ES$ 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. The remaining members of the population are formed by mutating the parent. We define mutation, m_n to be the number of discrete mutations made in the entire collection of chromosomes. The experiment was performed over 20 independent runs in case of each experiment. Note if there is no offspring that has a higher fitness than the parent, but there is at least one has a fitness equal to the parent, then an offspring is chosen to be the new parent. This algorithm was used partly because of its simplicity and partly because in other studies comparisons have been made between an evolutionary software approach using this evolutionary strategy and evolution-in-materio [3], [18], [17].

TABLE III: Comparative results of different electrode combination of chromosome. The experiment has been performed using the bin-packing benchmark HARD0 [19]. In the first column $p_{y,z}$ denotes electrode combination of chromosome, where y is the number of electrodes used as outputs and z is the number of electrodes used as configuration voltages. The second column indicates the average result of 20 runs. The third column indicates the best result of all 20 runs. 'Overflow' is used where at least one bin was filled beyond its capacity.

Pin Configuration	Average minimum number used bins	Best minimum number used bins
$p_{1,11}$	Overflow	69
$p_{2,10}$	70.75	68
$p_{4,8}$	Overflow	72
$p_{5,7}$	Overflow	73
$p_{10,2}$	Overflow	71

TABLE IV: Comparative results of two different numbers of mutations applied in experiments. The experiment has been performed using the bin-packing benchmark HARD0 [19]. The first column shows the number of mutations used. The second column shows the average result of 20 runs. The third column indicates the best result of all 20 runs. The electrode combination used for all the experiments mentioned in this table is 2 electrodes as outputs and 10 as configuration voltages as this electrode combination gave best results according to table III.

Mutation number	Average Minimum number used bins	Best minimum number used bins
1	70.75	68
2	73.3	71

Twelve benchmarks of bin-packing problems were used in the experiments (four from each difficulty class). We used the benchmark HARD0 (the first benchmark in the hardest category) to determine how many output to use in each chromosome and the best number of mutations to use in the later experiments. We performed five different experiments with HARD0 to determine the best number of outputs per chromosome. For all of these experiments we chose a single mutation ($m_n = 1$). The results are shown in table III.

We also investigated using HARD0 two different values of m_n to see which gave the best results. The number of mutations used was either one or two. The results are shown in table IV. We found that best number of mutations is one and best electrode combination is to use two electrodes as outputs and ten as configuration voltages. We used these parameter settings in all other experiments. The results are shown in Table V. It should be noted that 20 evolutionary runs (5000 generations each) on each benchmark problem took more than 2 days.

A. Analysis of Results

The experiments show that on 3/12 benchmarks, the average results of experimental material are close to optimum and in case of 4/12 benchmarks, the best results of experimental material are close to optimum. In two cases the results of all runs were unable to find a solution in which all bins were within capacity. In one set of evolutionary runs, some of the runs could not find solutions in which all bins were within capacity but other did.

In further experiments, one of the benchmark problem (N4W2B3R5) which could not find solutions all within the bin capacity (in 5000 generations) was investigated over longer evolutionary runs. Ten runs of 25000 generations were performed with the N4W2B3R5 benchmark. This dramatically changed the results for the better. The average result of 10 runs was 128 and best result in 10 runs was 125, where optimum is 101. This indicates that if evolutionary runs are performed for more number of generations, far better results can be found (i.e. the algorithm does not get stuck in a local optimum).

We examined the final gene values of configuration data for one single bin packing problem and could find no pattern. This is not surprising as the number of gene values is very large. For example, for 50 item bin packing problem with 10 configuration voltages and 2 outputs, requires 25 chromosomes each having 25*72=1800 genes.

In addition we examined whether the material was an essential part of the experiment by attempting to evolve solutions using an electrode array containing no material. One of the bin packing benchmarks (dataset N1C1W1_A) was chosen. For this experiment we used two electrodes as outputs and ten configuration voltages. We carried out an single evolutionary run of 1000 generations using a single mutation to create each offspring. We found that no evolution happened at all in this case. The fitness value in first generation is same as fitness value of 1000th generation. The fitness value indicates that all items went into the same bin. We can conclude that the material is essential for solving computational problems, however we can not, at this stage, rule out that the Mecobo board itself is not involved in the computation in some way. In future work we intend to construct a standalone system in which the appropriate final evolved configuration signal are provided in a circuit without the Mecobo board. Then an investigation can be carried out to see if the electrode array still solves the chosen benchmark problem.

VIII. Conclusions And Future Outlook

We have shown that computer-controlled evolution of signals applied to an electrode array can be used to solve the well known difficult computational problem of bin-packing. This is the first time that this problem has been tackled in this way. In some cases, we found that we could find solutions that were close to the known global optimum. Despite this, at this stage, we do not claim that the experimental results of solving bin packing problem using EIM are particularly competitive with state-of-the-art bin packing algorithms, Our aim is rather to show that EIM can solve standard computational problems and to demonstrate that exploiting the physical properties of materials for computational purposes is both feasible and promising.

In other work using Mecobo, it has been shown that digital logic functions can be implemented [12]. We have also obtaining encouraging results on function optimization [17], traveling salesman [3] and machine learning classification [18] problems. Although in this paper, we have looked at whether we can evolve solutions to a hard computational problem, it is also possible to try to evolve functions that may have utility in electronics applications. For instance, in a sister paper we showed that we can evolve frequency classifiers using the Mecobo system [16]. It is interesting to note that in principle, a classifier can be implemented using an electrode array and a material sample on a microscope slide and some interfacing electronics. Such a system could act as a standalone device. This could have utility in a number of application areas.

Naturally, there remain many questions for the future. How does evolutionary computation in materio scale on larger problem instances. What other classes of computational problems can be solved using this technique? What are the most suitable materials and signal types for evolution-in-materio? In future different substrates and different mixtures of SWCNT and polymers will be used to solve same computational problems. This will allow us to give comparative results. At present, the electrode arrays with their materials are purpose built. Currently a number of other electrode designs are in production together with different materials, it remains to be seen which will have most utility for problem solving. In other work by colleagues in the NASCENCE project [2] experiments are being conducted using extremely small electrode arrays to manipulate the movement of electrons in random collections of gold nanoparticles. Currently the electrode array has to be kept at temperatures in the milli-Kelvin, however in principle using different sized nanoparticles may allow the system to operate at room temperature.

Finally, a new version of the Mecobo platform has now been constructed which will able to allow the utilization of analogue voltages. This may make some types of computational problems more readily solved.

IX. 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] Banzhaf, W., Beslon, G., Christensen, S., Foster, J., Képès, F., Lefort, V., Miller, J., Radman, M., J., R.: Guidelines: From artificial evolution to computational evolution: a research agenda. Nature Reviews Genetics **7**, 729–735 (2006)

[2] 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)

TABLE V: Experimental results on twelve bin packing benchmark problems. The best and average results for experimental material are computed from 20 independent evolutionary runs. All results are for 5000 generations. Two electrodes have been used as outputs and ten are supplied with configuration signals or voltages. A single mutation is used to generate offspring in the evolutionary algorithm ($m_n = 1$). The third column indicates the dataset to which the instance belongs to (according to [19]).The fifth column indicates the average result of all runs. The sixth column indicates the best result of all runs. 'Overflow' is used where at least one bin overflowed. The "Result" column shows whether the result of experimental material is equal to or close to optimum or not. '✓' indicates the result is equal to or close to optimum and 'X' indicates the result is not close to optimum. The first results of this column are for average results and second results are for best results. Note 5000 generations of a $1 + 4$ evolutionary strategy amounts to very few bin-packing evaluations.

Instance name	Optimum result	Dataset	Total number of items	Minimum number of used bins (Average)	Minimum number of used bins (Best)	Result
N1C1W1_A	25	1	50	27.4	27	✓ ✓
N2C2W2_B	56	1	100	63	60	X ✓
N3C3W4_R	87	1	200	Overflow	103	X X
N4C2W1_M	217	1	500	Overflow	Overflow	X X
N1W1B1R0	18	2	50	20.2	20	✓ ✓
N3W1B2R3	65	2	200	81.7	78	X X
N4W2B3R5	101	2	500	Overflow	Overflow	X X
N2W3B1R9	15	2	100	20.45	19	✓ ✓
HARD0	56	3	200	70.75	68	X X
HARD3	55	3	200	71.15	68	X X
HARD4	57	3	200	70.05	69	X X
HARD9	56	3	200	72.4	70	X X

[3] Clegg, K.D., Miller, J.F., Massey, K., Petty, M.: Travelling salesman problem solved 'in materio' by evolved carbon nanotube device. In: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (eds.) Parallel Problem Solving from Nature PPSN XIII, *Lecture Notes in Computer Science*, vol. 8672, pp. 692–701. Springer International Publishing (2014)

[4] Coffman Jr., E.G., Garey, M.R., Johnson, D.S.: Approximation algorithms for bin packing: A survey. In: D.S. Hochbaum (ed.) Approximation Algorithms for NP-hard Problems, pp. 46–93. PWS Publishing Co., Boston, MA, USA (1997)

[5] Dawkins, R.: The Blind Watchmaker. Penguin Books Limited (1988)

[6] Harding, S., Miller, J.F.: Evolution in materio: A tone discriminator in liquid crystal. In: In Proceedings of the Congress on Evolutionary Computation 2004 (CEC'2004), vol. 2, pp. 1800–1807 (2004)

[7] Harding, S., Miller, J.F.: Evolution in materio : A real time robot controller in liquid crystal. In: Proceedings of NASA/DoD Conference on Evolvable Hardware, pp. 229–238 (2005)

[8] Harding, S., Miller, J.F.: Evolution in materio. In: R.A. Meyers (ed.) Encyclopedia of Complexity and Systems Science, pp. 3220–3233. Springer (2009)

[9] Harding, S.L., Miller, J.F.: Evolution in materio: Evolving logic gates in liquid crystal. International Journal of Unconventional Computing **3**(4), 243–257 (2007)

[10] Harding, S.L., Miller, J.F., Rietman, E.A.: Evolution in materio: Exploiting the physics of materials for computation. International Journal of Unconventional Computing **4**(2), 155–194 (2008)

[11] Horowitz, E., Sahni, S., Rajasekaran, S. (eds.): Computer Algorithms. Silicon Press (2007)

[12] Lykkebø, O.R., Harding, S., Tufte, G., Miller, J.F.: Mecobo: A hardware and software platform for in materio evolution. In: O.H. Ibarra, L. Kari, S. Kopecki (eds.) Unconventional Computation and Natural Computation, LNCS, pp. 267–279. Springer International Publishing (2014)

[13] Miller, J.F. (ed.): Cartesian Genetic Programming. Springer (2011)

[14] Miller, J.F., Downing, K.: Evolution in materio: Looking beyond the silicon box. In: A. Stoica, J. Lohn, R. Katz, D. Keymeulen, R.S. Zebulum (eds.) The 2002 NASA/DoD Conference on Evolvable Hardware, vol. 7, pp. 167 – 176. IEEE Computer Society (2002)

[15] Miller, J.F., Harding, S.L., Tufte, G.: Evolution-in-materio: evolving computation in materials. Evolutionary Intelligence **7**, 49–67 (2014)

[16] Mohid, M., Miller, J.F., Harding, S.L., Lykkebø, O.R., Tufte, G., Massey, M.K., Petty, M.C.: Evolution-in-materio: A frequency classifier using materials. In: Proceedings of the Conference on Evolvable Systems: From Biology to Hardware. IEEE Symposium Series on Computational Intelligence, pp. XX–XX. IEEE Computational Intelligence Society (2014). In Press

[17] Mohid, M., Miller, J.F., Harding, S.L., Tufte, G., Lykkebø, O.R., Massey, M.K., Petty, M.C.: Evolution-in-materio: Solving function optimization problems using materials. In: Computational Intelligence (UKCI), 2014 14th UK Workshop on, pp. xx–xx (2014). In press

[18] Mohid, M., Miller, J.F., Harding, S.L., Tufte, G., Lykkebø, O.R., Massey, M.K., Petty, M.C.: Evolution-in-materio: Solving machine learning classification problems using materials. In: T. Bartz-Beielstein, J. Branke, B. Filipič, J. Smith (eds.) Parallel Problem Solving from Nature PPSN XIII, *Lecture Notes in Computer Science*, vol. 8672, pp. 721–730. Springer International Publishing (2014)

[19] Scholl, A., Klein, R.: Bin packing. URL http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm

[20] Scholl, A., Klein, R., Jürgens, C.: BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers Operations Research **24**(7), 627 – 645 (1997)

[21] Thompson, A.: An evolved circuit, intrinsic in silicon, entwined with physics. In: T. Higuchi, M. Iwata, L. Weixin (eds.) Proc. 1st Int. Conf. on Evolvable Systems (ICES'96), *LNCS*, vol. 1259, pp. 390–405. Springer-Verlag (1997)

[22] Thompson, A.: Hardware Evolution - Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution. Springer (1998)