The alchemy of computation: designing with the unknown

Julian Francis Miller¹

Published online: 24 April 2019
© The Author(s) 2019

Abstract

Modern computers allow a *methodical* search of possibly billions of experiments and the exploitation of interactions that are not known in advance. This enables a *bottom-up* process of design by assembling or configuring systems and testing the degree to which they fulfill the desired goal. We give two detailed examples of this process. One is referred to as Cartesian genetic programming and the other evolution-in-materio. In the former, evolutionary algorithms are used to exploit the interactions of software components representing mathematical, logical, or computational elements. In the latter, evolutionary algorithms are used to manipulate physical systems particularly at the electrical or electronic level. We compare and contrast both approaches and discuss possible new research directions by borrowing ideas from one and using them in the other.

Keywords Evolutionary algorithms · Cartesian genetic programming · Evolution-in-materio · Physical computation

1 Introduction

Despite having no designer, natural evolution has caused the construction of “machines” of incredible sophistication and complexity. In the construction of biological organisms it has utilised countless physical properties of materials, many of which are still not fully understood. It is now possible to emulate natural evolution’s ability to utilise and exploit unknown physical and software interactions to create useful artefacts. We refer to this as computational alchemy. This process is akin to alchemy in that there is a search for a desired goal which is similar to the process of alchemy where poorly understood processes are utilised in a goal-driven, bottom-up and informal process to achieve a desired result.

A simplified and algorithmic form of natural evolution is widely used to build solutions to many computational problems. The field has two main subdivisions: genetic programming (GP) and evolutionary algorithms (EAs). The general form of an EA is shown in algorithm 1:

Algorithm 1 Evolutionary Algorithm

```
1: Randomly generate a population of individuals
2: Generation = 1
3: while a solution is not found or the generation limit is not reached do
4:   Evaluate the fitness of the members of the population (A)
5:   Make new population (B) by selecting fitter members of A
6:   Recombine some members of B
7:   Randomly alter some members of B
8:   Replace A with B
9:   Increment generation counter
10: end while
```

In evolutionary algorithms, fixed length encodings of potential solutions are evolved to find optima. To do this one identifies all the variables that are required to build a solution to a problem and then a vector or string of these variables is constructed. This string is usually referred to as chromosome or genotype.

GP uses EAs to evolve variable sized computational structures (Koza 1992). In this paper, we describe a form of genetic programming known as *Cartesian genetic programming* (CGP) (Miller 2011; Miller and Thomson 2000). CGP represents computational structures as graphs. It uses a fixed length list of genes describing the function and connections of computational nodes. It can have multiple outputs. CGP has been used to evolve electronic circuits, mathematical expressions, artificial neural networks, differential equations, data classifiers, control systems and solutions to many other types of problems. CGP exploits

✉ Julian Francis Miller
julian.miller@york.ac.uk

¹ Department of Electronic Engineering, The University of York, York YO10 5DD, UK

unknown or poorly understood interactions between *known* computational primitives (nodes) and often finds unusual or innovative solutions to problems.

In evolution-in-materio (EIM), EAs are used to exploit unknown interactions between *unknown* or poorly understood physical components to find solutions (Miller and Downing 2002; Miller et al. 2014). The main idea of EIM is that the application of some physical signals to a material (configuration variables) can cause it to alter how it responds to an applied physical input signal and how it generates a measurable physical output (see Fig. 1). Physical outputs from the material are converted to output data and a numerical fitness score is assigned depending on how close the output data (or a function of it) is to that desired. This fitness is used in an EA to attempt to find configurations that cause a material to produce a desired behaviour. In EIM, the variables optimised by evolutionary algorithms are usually the magnitudes of voltages and the positions of the electrodes to which the voltages will be applied.

The purpose of this paper is to compare and contrast these two approaches and identify new insights that might

be useful to improve either technique. The underlying assumption of EIM is that utilising potentially unknown physical effects and interactions can improve evolvability. Materials that possess such properties can be thought of as physically rich. They can be more evolvable precisely because evolution has more dimensions that can be adjusted. Can such a concept be emulated in some way, in software? Miller and Hartmann suggested including a certain degree of randomness in models of logic gates and showed that this conferred a number of advantages such as fault tolerance and small circuit size (Miller and Hartmann 2001). Another possible route to enriching the basic components used in circuit evolution might be to represent the circuits at the transistor level (Greenwood and Tyrrell 2007; Trefzer and Tyrrell 2015). This would allow aspects of physics in the simulation model to be exploited. Enriching software models of components however has the drawback that it increases simulation time. On the other hand in EIM one could build arrays of evolved molecular devices, by evolving the basic components first and then allowing artificial evolution to combine them to solve computational problems.

The plan of the paper is as follows. Firstly, we explain CGP then in Sect. 3 we discuss three examples where unknown or poorly understood interactions between software components are exploited to create innovative solutions to various problems. In Sects. 4–6 we discuss how computer controlled evolution can utilise unknown or poorly understood physical interactions to solve computational problems. In Sect. 7 we compare and contrast the two methods and discuss future possibilities.

2 Cartesian genetic programming

In the original form of CGP graphs were indexed by their Cartesian coordinates, hence the term “Cartesian” (Miller and Thomson 2000). It differs from GP, which represents programs in the form of trees, in a number of ways. It uses graphs to represent solutions to computational problems which means that the outputs of the nodes in the graph can be reused multiple times rather than once as in trees. Secondly, it allows programs to have multiple-outputs which widens its applicability. It uses a simple address-based program genotype encoding which can ignore nodes if they are not referenced. The presence of explicitly inactive genes assists evolution by allowing a process of genetic drift (Miller and Smith 2006; Turner and Miller 2015a; Vassilev and Miller 2000; Yu and Miller 2001, 2006). As the upper bound on the maximum genotype size is increased an increasing proportion of genes are unexpressed this assists neutral exploration as many mutated genotypes have the same phenotype. As a result

Fig. 1 Concept of evolution-in-materio. There are two domains: physical and 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 EA. 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 EA. The EA starts with a randomly generated initial population. It ends when either the used-defined number of generations has elapsed or the fitness value is satisfactory

there is a strong bias in CGP for small phenotypes (Goldman and Punch 2013). This can often mean that evolved solutions are small and can be interpreted. The maximum allowed size of the genotype is highly influential (Turner and Miller 2015a) on search efficiency and often the ideal genotype length is quite large (hundreds or thousands of nodes).

Originally, CGP encoded acyclic graphs only, but it has also been extended to represent cyclic (Ryser-Welch et al. 2016) and recurrent graphs (Turner and Miller 2014b; Walker et al. 2009).

Each CGP chromosome consists of function genes (F_i), connection genes (C_{ij}) and output genes (O_i); where i indexes each node and j indexes each nodes inputs. The function genes represent indexes in a function look-up-table and describe the functionality of each node. The connection genes describe from where each node gathers its inputs. For regular acyclic CGP, connection genes may connect a given node to any previous node in the program, or any of the program inputs. The output genes address any program input or internal node and define which are used as program outputs.

Originally CGP programs were organized with nodes arranged in rows (nodes per layer) and columns (layers); with each node indexed by its row and a column. However, here we show a generic (one row) CGP chromosome (see Equation 1), where α is the arity of each node, n is the number of nodes and m is the number of program outputs.

$$F_0 C_{0,0} \dots C_{0,\alpha} F_1 C_{1,0} \dots C_{1,\alpha} \dots F_n C_{n,0} \dots C_{n,\alpha} O_0 \dots O_m \quad (1)$$

An example CGP program is given in Fig. 2 along with its corresponding chromosome. As can be seen, all nodes are

Fig. 2 Example CGP program. The program has three inputs, x , y , z and three outputs O_0 , O_1 , O_2 . The genotype is shown at the top of the figure. Underlined genes indicate node function genes. In the example there are four possible node functions, add, subtract, multiply and divide (with genes 0, 1, 2, 3). The lower schematic shows how the genotype is decoded into a graph and thence into equations. Note that program input, z is not used. The function of node 3 is add its inputs, x and y so it produces $x + y$. The output of node 4 is not used as neither nodes or program outputs refer to it. Node 5 multiplies its inputs so it produces $(x + y) * y$. Node 6 also multiplies its inputs (which are y and the output of node 5) so it produces $(x + y)^2 y$. Thus the graph represents the three equations: $O_0 = x$, $O_1 = (x + y) * y$ and $O_2 = (x + y)^2 y$

connected to previous nodes or program inputs. Not all program inputs have to be used, enabling evolution to decide which inputs are significant. CGP typically uses a simple evolutionary strategy (ES), sometimes referred to as a probabilistic hill-climber, called a $(1 + \lambda)$ -ES, where $\lambda = 4$. This is shown in Algorithm 2.

The integer genes in CGP are constrained to make sure all links and node functions in the graph are valid.

Algorithm 2 The $(1 + 4)$ evolutionary strategy

```

1: for all  $i$  such that  $0 \leq i < 5$  do
2:   Randomly generate individual  $i$ 
3: end for
4: Select the fittest individual, which is promoted as the parent
5: while a solution is not found or the generation limit is not reached do
6:   for all  $i$  such that  $0 \leq i < 4$  do
7:     Mutate the parent to generate offspring  $i$ 
8:   end for
9:   Generate the fittest individual using the following rules:
10:  if an offspring genotype has a better or equal fitness than the parent then
11:    Offspring genotype is chosen as fittest
12:  else
13:    The parent chromosome remains the fittest
14:  end if
15: end while

```

Algorithm 2 allows only one genotype to be selected for the next generation (called the parent), then four offspring are generated from the parent by copying the parent and then making some random alterations (mutation). Since many genes are inactive, it is often the case that the parent and offspring only genetically differ from the parent in the inactive genes. This means that the phenotype of such offspring will be identical to the parent. In situations where no offspring has a fitness greater than the parent but at least one has an equal fitness, then the the offspring is chosen as the new parent. This allows a process of genetic drift to occur.

3 Exploitation of unknown interactions in CGP

3.1 Digital circuits

Algorithms that optimise the number of gates used to build Boolean functions have been an active area of research for many decades (Brayton et al. 1984; Devadas 1994; Sentovich et al. 1992). However, in order to make general purpose logic minimisation algorithms they assume rather limited choices of atomic logic gates such as AND, OR, NOT or AND, EXOR, NOT and this can lead to sub-optimum circuits (Miller et al. 2000). This is a general problem that applies to many fields. To create an automated design system very restrictive assumptions have to be made about how such systems can be represented, this is where computational alchemy can be useful. One can use representations of solutions that are much less restrictive,

in the logic case this could mean allowing circuits to be built from a large collection of primitive gates e.g. all 16 two-input Boolean functions, multiplexers, universal logic modules. It is important to realise that there are *no* formal algorithmic methods for building circuits that use such collections of components. We give a simple example of the strangeness of solutions designed using computational alchemy in Fig. 3. The use of the two input Boolean function which is an AND gate with one input inverted is very peculiar and would not be chosen by a human designer. Bitwise multiplication follows the conventional paradigm of long multiplication. This produces the circuit shown in Fig. 3a. Allowing the use of all possible two-input Boolean functions allows bizarre circuits to arise that combine the elementary products in ways which a human designer would not consider since they do not operate according to conventional principles (Fig. 3b). It should be noted that although the total number of elementary gates used in both designs is the same, the number of transistors required may be less in the evolved design as it uses only one XOR gate (which is relatively transistor expensive).

However, with this representational freedom comes the drawback of an enormous and intractable search space. Directly trying to evolve circuits with a reasonable number of inputs is intractable as the number of conditions is an exponential function of the number of inputs. This problem was elegantly solved in Vašíček and Sekanina (2011) where CGP is used to synthesise digital circuits with hundreds of inputs and thousands of gates (Vašíček 2015). The intractability problem was solved by using conventional algorithms to synthesise the smallest circuit and then using this circuit to provide a seed for CGP to optimise it further. However, to check whether an evolved circuit correctly implemented the chosen circuit function a SAT solver was used, this could check the equivalence of logical functions in usually polynomial time. Evolved circuits that did not correctly implement the targeted circuit were given

a fitness of zero, whereas the ones that were correct were given higher fitness scores for being small. This illustrates that computational alchemy can operate successfully from human designed starting points.

3.2 Object recognition

Harding et al. (2013) and Leitner et al. (2012) used CGP to evolve very sophisticated object recognizers using conventional components. The CGP nodes were chosen from a large collection of elementary functions. The data type flowing through the CGP graph was an image. The node functions included arithmetic, mathematical, statistical and many image operations available in a well-known image processing library (OpenCV) (Bradski 2000). Harding used a set of images including the objects of interest and the aim was to produce a pixel classifier which identified whether a pixel in an image belonged to a given object or not. The training set of images included the object in different positions and lighting conditions and with the presence of other objects (not of interest). The training images included images containing the object passed through various filters such as: hue, saturation, brightness, red, green, blue, and grayscale. An example of an evolved CGP graph showing transformations of input images after various node operations is shown in Fig. 4.

The evolved filter program corresponding to the data flow graph is shown in Fig. 5. The first thing to note is how small the program is. This illustrates the tendency of CGP to produce compact solutions to problems. As with evolved digital circuits, evolved object recognition filters combine well-understood components in very unusual ways and there is no formal procedure that can design such programs. Although this particular image filter is hard to understand it is possible that some evolved filters could be understandable which could provide some insights in object recognition in general.

Fig. 3 Two-bit multipliers: conventional (a), evolved (b). Circles on inputs to gates indicate inversion. Both designs use the same number of elementary gates. The least significant input (product) bits are A0 and B0 (P0). The conventional circuit computes the output products in

order: P0, P1, P2, P3. The evolved computes the outputs in a very peculiar way, P3 and P1 are produced together and P0 and P2 are produced together in two separate sub-circuits (Miller et al. 2000)

Fig. 4 CGP image data flow graph (Harding et al. 2013) that can identify pixels belonging to a particular packet of tea. On the left two images of a scene containing the object of interest are used. Various image processing functions are applied resulting in a binary classification image (far right) showing where pixels are that belong to the packet of tea

3.3 CGP encoded artificial neural networks (CGPANNs)

CGP can encoded artificial neural networks (ANNs) (Khan et al. 2010a, b, 2013). Encoding and evolving ANNs in CGP gives immediate benefits. Turner et al. demonstrated that using heterogeneous activation functions gave improved results compared with the traditional single activation function ANNs (Turner and Miller 2014a). In addition, it has been shown that evolving ANNs with CGP produced networks whose performance was much less sensitive to topology choices than fixed topology evolved ANNs (Turner and Miller 2013). The topology of CGPANNs is unusual when compared with traditional

ANNs and often cannot be described in terms of layers and nodes per layer.

Figure 6 gives an example of the type of ANN which can be created using CGP. Neuron inputs are highly unconstrained; they can connect to any previous neuron in the network including input neurons. The arity of each neuron can vary. Additionally any neuron can be used as an output; including the input neurons. Neurons can have different and arbitrary activation functions, possibly even non-differentiable functions. There can also be multiple connections between the same pair of neurons (e.g. the two connections between the logistic neuron and the rectilinear neuron). CGP is capable of discovering topologies which would be unlikely to be considered by a human designer.

```
public class MyEvolvedFilter : GpImageFilterRunner
{
    public override GpImage RunFilter() {
        GpImage node0 = InputImages[0];
        GpImage node1 = InputImages[1];
        GpImage node2 = node0.erode(1);
        GpImage node3 = node2.ShiftDown();
        GpImage node4 = node0.absdiff(node2);
        GpImage node5 = node0.avg(node2);
        GpImage node7 = node5; //NOP
        GpImage node8 = node1.erode(3);
        GpImage node9 = node3.sub(node8);
        GpImage node12 = node4.add(node4);
        GpImage node13 = node7.min(node12);
        GpImage node16 = node13.absdiff(node9);
        GpImage node24 = node16.sub(node9);
        GpImage node50 = node24.gauss(15);
        GpImage node78 = node50.gauss(15);
        GpImage node89 = node78.threshold(64);
        GpImage node99 = node89.gauss(13);
        return node99;
    }
    public override void SetUsedInputs() {
        this.UsedInputs.Add(1);
        this.UsedInputs.Add(0);
    }
}
```

Fig. 5 Evolved object recognition program (Harding et al. 2013) that can identify pixels belonging to a packet of tea. The program can be executed in real-time on live video. This allows an object to be tracked while undergoing rotation, being partly hidden, with changed lighting and other environmental changes

4 Evolution-in-materio

The concept of EIM was inspired by the work of Adrian Thompson et al. (1999) and Thompson (2001) in a sub-field of evolutionary computation called evolvable hardware (Haddow and Tyrrell 2011; Tyrrell and Trefzer 2015b). Thompson demonstrated that unconstrained evolution on a silicon chip called a Field Programmable Gate Array (FPGA) could utilize the physical properties of the chip to solve computational problems. Inspired by this, Harding and Miller showed that unconstrained evolution on a liquid crystal display could also solve a number of computational problems (Harding et al. 2004, 2015; Harding and Miller 2007; Harding et al. 2008). Indeed, it seemed to be easier to evolve computational functions with liquid crystal than it was with silicon (Harding et al. 2004). Interestingly, similar ideas, albeit without computers, were conceived in the late 1950s (particularly by Gordon Pask Cariani 1993; Pask 1958). Pask's material was an acidified solution of Ferrous Sulphate and adjusting voltages applied

Fig. 6 Unusual ANNs can be evolved using CGP

to a solution grew iron threads whose form was related to the magnitude of current flow.

4.1 Travelling salesman problem

To illustrate EIM we choose a classic problem in computer science. Namely the travelling salesman problem (TSP). The aim of which, is to discover the smallest length tour of a number of cities. Small instances of TSP problems were solved using the EIM approach (Clegg et al. 2014). Single-walled carbon nanotubes (SWCNT) in PMMA polymer (Poly-methyl-methacralate) were placed on an electrode array. The material is configured by applying static voltages to electrodes. The voltages on the remaining electrodes are used to define a permutation of cities.

Figure 7 shows results using a 3×4 gold electrode array for a nine-city TSP problem. The particular TSP instances were generated by placing cities on a circle so that they were equidistant from one another. The genotype defined a number of real-values voltages and to which electrodes these configuration voltages would be applied. The latter was accomplished by using digitally configurable analogue cross point switches. A data acquisition card (DAQ) digitally configures the switch connections and then generates analogue configuration voltages which are applied to the electrode array. Finally, it records the voltages on the non-configuration electrodes. The number of configuration voltages deployed depends on the problem being tackled and the availability of spare electrodes on the array. The configuration voltages and electrodes to which they connect were decided by a 1+4-ES (see Sect. 2). The range of voltages values was restricted to $\pm 3V$ and all connections are one to one (i.e. one configuration voltage can only go to one electrode). Configuration voltages were applied for one second and a mean value of sampled voltages from the output electrodes was calculated from the last 0.2 seconds of sampled values. This was done to exclude any “settling periods” within the material. The time required to configure the analogue switch and set up

channels on the DAQ card means that testing a configuration takes several seconds. Actually further investigations revealed that signals from the SWCNT-PMMA materials have negligible noise levels after the initial 50ms so that sampling times could be substantially reduced.

The method of obtaining a tour of cities (i.e. a permutation) is as follows. A vector of voltage values with as many elements as cities is read from the electrode array. The i th element represents city i . The vector is sorted and the city indexes form a permutation, thus defining a tour (see Clegg et al. 2014 for details). The graphic at the top right of Fig. 7 shows a set of voltages read from a 3×4 array. Choosing the lowest voltages (y-axis) in sequence and observing which electrode corresponds to that, one obtains the permutation: 7 6 5 4 3 2 1 9 8. The results are shown in Table 1.

It remains to be seen how quickly larger solutions of the TSP can be evolved on larger electrode arrays. Also it would be interesting to investigate whether EIM could be used to solve general TSP problems by using an evolved device to choose which city to visit next on a given problem. In principle this could be done by inputting real-valued Cartesian coordinates (normalised) of cities and trying to use evolution to identify an output that goes high say when the next city is a good one to choose for a minimal tour.

4.2 Other problems attempted with CNTs

Within a European project called NASCENCE (Broersma et al. 2012) it was shown that many computational problems could be solved using EIM working with carbon nanotubes: classification (Mohid et al. 2014d; Vissol-Gaudin et al. 2016), bin-packing (Mohid et al. 2014b), function optimization (Mohid et al. 2014c), frequency classification (Mohid et al. 2014a), Boolean logic functions (Kotsialos et al. 2014; Massey et al. 2015; Mohid and Miller 2015b), robot control (Mohid and Miller 2015a) and graph colouring (Lykkebø and Tufte 2014). CNTs were

Fig. 7 Top left: Array of electrodes (3×4) with non-uniform covering of SWCNTs (a). Note the mask fault on the third electrode. This did not appear to affect the evolutionary search when finding the shortest tour of the TSP. The fitness of the best performing genotype for each generation in an evolutionary run is shown (d). The

configuration voltages defined by the best genotype are applied to the circled electrodes and the remaining electrodes provide the floating point voltage values used to determine the tour of cities (b). Recorded voltages which when sorted determine the order to visit cities (b). Optimum tour solution of the TSP (c)

Table 1 9 or 10 city TSP results for different numbers of configuration voltages

Size of TSP	Number of configuration voltages	Average (median) number of generations for successful runs
9	2	158.6 (104.5)
9	3	57.36 (42.5)
9	4	118.4 (61.5)
10	2	157.95 (155.0)
10	3	79.76 (63.5)
10	4	68.03 (46.5)

Averages and medians are calculated over 30 runs of a 1+4-ES for 1500 generations

also used in EIM as a reservoir computing device (See Sect. 6 for more details) on time series prediction, memory

capacity, signal transformation and classification (Dale et al. 2016a, b, c).

5 EIM with gold nanoparticles

EIM was used to configure disordered networks of gold nanoparticles (NPs) to act as two-input Boolean gates (Bose et al. 2015). Fig. 8a, shows an atomic force micrograph (AFM) of a disordered network of gold nanoparticles (NPs) interconnected by insulating molecules (1-octanethiols) (b) over a 200nm diameter circular space surrounded by 8 radial electrodes (Ti/Au). The electrodes sit on top of a highly doped Si/SiO₂ substrate, which functions as another electrode (called the back gate). At low temperatures, the charging energy of the nanoparticle is given by, $EC = e^2/C$ (with e the electron charge and C

the nanoparticle's total capacitance) is larger than the thermal energy, and each NP exhibits Coulomb blockade and acts as a single electron transistor. One electron at a time can tunnel when sufficient energy is available (ON state), either by applying a voltage across the SET or by electrostatically shifting its potential. Otherwise, the transport is blocked because of the Coulomb blockade (OFF state). Two electrodes are chosen as inputs (V_{IN1} , V_{IN2}) and there is one output (I_{OUT}). The remaining 5 electrodes and back gate have fixed voltages applied to them. An EA was used to arrive at suitable configuration voltages to make the device operate as a logic gate.

Time dependent signals in the order of a hundred mV are applied to the input electrodes and a time dependent current in the order of a hundred pA is read from the output electrode (See Fig. 9). It was found that the EA was able to find the values of the voltages to apply to the electrodes that programmed the *same device* to act as any of the 16 possible two-input Boolean functions. All gates showed great stability and reproducibility. Indeed, it was found that the gates still functioned correctly when perturbations were applied to the configuration voltages. Also after raising temperatures above the operating range and then cooling again to the same temperatures at which they were evolved the gates recovered reliably their original evolved function.

Although the EIM work using gold nanoparticles is very promising the technique needs to be evaluated on many more computational problems. Even though it is possible to solve specific problems with EIM evolved devices, it would be more useful if general purpose computational devices could be obtained. In the next section we will see how this could potentially be accomplished.

Fig. 8 Circular eight electrode array. The material is a disordered network of 20 nm gold nanoparticles interconnected by insulating molecules (1-octanethiols). The nanoparticles are trapped in a circular region (200 nm in diameter) between radial metal (Ti/Au) electrodes on top of a highly doped Si/SiO₂ substrate. The device operates at temperatures below 1°K (Bose et al. 2015)

6 Reservoir computing with carbon nanotubes

Reservoir computing (RC) is a computational paradigm which utilises a dynamical system (the reservoir) and a trainable readout mechanism (Jaeger 2001; Maass 2011; Schrauwen et al. 2007). Often reservoirs are implemented as randomly wired recurrent neural networks (Jaeger 2001). External input signals are fed into a reservoir and the internal dynamics of the reservoir maps the input into a higher dimensional space. RC uses a simple readout mechanism (weighted outputs) that is trained to read the state of the reservoir and map it to the desired output. The main benefit is that training is performed only at the readout stage and the reservoir is fixed. A reservoir functions much like a temporal kernel (Lukoševičius et al. 2012) and can represent any excitable non-linear medium provided two requirements are met (1) that it can create a high-dimensional projection of the input into observable reservoir states; and (2) that it possesses a fading memory of previous inputs and internal states. With these properties it has been shown that a reservoir can realise any non-linear filter with bounded memory, and with the aid of a trained readout approximate any function.

Dale et al. used EIM to create physical reservoir computing devices (Dale et al. 2016c, 2017). The reservoirs

Fig. 9 Digital input signals and two evolved responses. The objective is to produce an logical AND function of the two inputs P and Q . One figure (with red cross) shows a response with low fitness. The other shows a correct response which would have high fitness (green cross). (Color figure online)

were electrode arrays covered with a layer of carbon nanotubes in a polymer. The RC in-materio approach provided a competitive new approach to unconventional computing. The system is depicted in Fig. 10. The advantage of using EIM to create reservoir computing devices is that evolved reservoirs could act as general purpose trainable computational devices rather than a computing device for a single application. Ultimately, implementing reservoir computing devices physically (rather than as neural networks) may offer devices that are faster and consume less power.

It would also be interesting to replace the neural networks in reservoir computers with CGP evolved graphs or even CGPANNs. Conceptually, allowing evolution to manipulate the inner workings of the reservoir computing device would appear to offer an alternative way of arriving at reservoirs with good properties. In addition, using CGP would allow reservoir computing devices to be constructed that handle more complex data (i.e. whole images or mixed data type) so that for instance general purpose trainable image filters could be attempted.

7 Comparing and contrasting CGP with EIM

In EIM one usually has little understanding of *useful* physical processes in materials that could be exploited in higher-level problem solving. As a consequence one applies configuration signals to various locations in the device to cause some physical effect that affects how applied inputs are transformed into useful outputs. In CGP, the internal components are known in advance and one is trying to exploit the *interactions* between these components for problem solving. Comparing the two methodologies suggests that CGP would be more like EIM if the internal connectivity of the graph was not directly affected by evolution, instead one could evolve configuration signals in addition to inputs to a fixed graph (presumably randomly generated). This is quite similar to the concept of

reservoir computing in which the reservoir (an analogue of a physical device) is fixed and only the external inputs and weights of output signals are adjusted. However, in CGP not allowing internal connections between primitive functions would be likely to hamper evolution as being able to manipulate internal interactions gives evolution a high degree of fine control of internal components. It would nevertheless be interesting to investigate the proposed alternative form of CGP to see how it performs in comparison with standard CGP.

Another way in which the benefits of ‘richness’ of materials that are exploited in EIM could inspire new variants of genetic programming could be to replace the primitive function set with random trees or graphs (possibly with pseudo-random constants). This would free one from deciding which primitive functions to use. Of course, the understandability of the evolved solutions would become more difficult, however the additional freedom for evolution to combine and exploit unusual interactions between components could be enhanced and it may lead to greater evolvability.

The central idea of EIM is that evolution may be able to exploit the richness of the material world for computation. The drawback is that it is not clear what kinds of materials are most evolvable and exploitable. However, recently, -Dale et al. (2018) have developed a method for characterising the quality of any substrate for reservoir computing. This may lead to a better understanding of what physical systems might be most suitable. Furthermore, in the future, it might be better to create an array of different materials each of which can be configured and with evolvable connections between the different materials. Then evolution could choose which materials to use and how to connect them. This would be like an FPGA but with materials rather than engineered silicon. Indeed such a proposed device has been referred to as a Field Programmable Matter Array (FPMA) (Miller and Downing 2002). Nanoparticle based EIM appears to offer great potential particularly since, as we have seen, evolved devices have high reliability and stability. There are many variants that could be investigated in the future. Different materials could be used to keep the nanoparticles apart and also one could also use insulating as well as conducting nanoparticles. Work is also underway in attempting to build devices with smaller nanoparticles that can operate at higher temperatures. An important aspect in the evolvability and scalability of EIM is the number of electrodes. With much higher electrode densities one can potentially have much finer control of small regions of material, this ought to mean that exploitation of physical effects could be enhanced, which would lead to higher evolvability.

A key question for EIM is how to obtain general purpose computing devices rather than solutions for particular

Fig. 10 Reservoir computing with carbon nanotube physical reservoir configured using an EA

problems. We have discussed reservoir computing as one potential methodology for general problem solving. In reservoir computing the reservoir is usually a recurrent neural network which has certain statistical properties. It isn't evolved. However, recurrent neural or recurrent CGP encoded ANNs or even recurrent non-neural CGP graphs could be used as the reservoir and evolution could be used to arrive at general purpose reservoirs.

Another step in this direction of general purpose computing from EIM devices could be to use a material in a similar way to compositional pattern producing networks (CPPNs) (Stanley 2007). CPNNs read the coordinates of pairs of artificial neurons and outputs a weight for that connection. In principle evolved material configurations could do the same. This would allow evolution to configure *materials* that could build arbitrary sized software ANNs.

8 Conclusion

We have discussed two forms of computational alchemy where computer-controlled search can exploit physical, mathematical, logical and software interactions to solve computational problems in unconventional ways. In CGP computational solutions are built from known components but the combinations and interactions between these components are not formally tractable while in evolution-in-materio the basic physical components are not even known in advance. Despite this, search algorithms are able to utilise these effects in problem solving. This mirrors in a small way, perhaps, the extraordinary exploitation of physics and chemistry made by natural evolution. The two techniques have not been discussed in the same article before and it is clear that comparing and contrasting the two techniques offers new insights into possible new research directions for both GP, EIM and reservoir computing.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Bose SK, Lawrence CP, Liu Z, Makarenko KS, van Damme RMJ, Broersma HJ, van der Wiel WG (2015) Evolution of a designless nanoparticle network into reconfigurable boolean logic. *Nat Nanotechnol*. <https://doi.org/10.1038/NNANO.2015.207>
- Bradski G (2000) The opencv library. *Dr. Dobb's Journal of Software Tools*
- Brayton RK, Sangiovanni-Vincentelli AL, McMullen CT, Hachtel GD (1984) Logic minimization algorithms for VLSI synthesis. Kluwer Academic Publishers, Berlin
- Broersma H, Gomez F, Miller JF, Petty M, Tufte G (2012) Nascence project: nanoscale engineering for novel computation using evolution. *Int J Unconv Comput* 8(4):313–317
- Cariani P (1993) To evolve an ear: epistemological implications of Gordon Pask's electrochemical devices. *Syst Res* 3:19–33
- Clegg KD, Miller JF, Massey K, Petty M (2014) Travelling salesman problem solved 'in materio' by evolved carbon nanotube device. In: *Parallel problem solving from nature—PPSN XIII*. Springer, pp 692–701
- Dale M, Miller JF, Stepney S, Trefzer MA (2016a) Evolving carbon nanotube reservoir computers. In: *UCNC, Lecture notes in computer science*, vol 9726. Springer, pp 49–61
- Dale M, Miller JF, Stepney S, Trefzer MA (2016b) Reservoir computing: evolution in Materio's missing link. In: *Proceedings of the 9th York doctoral symposium on computer science and electronics*, pp 57–67
- Dale M, Stepney S, Miller JF, Trefzer M (2016c) Reservoir computing in Materio: an evaluation of configuration through evolution. In: *Proceedings of the IEEE international conference on evolvable systems (ICES): from biology to hardware*. IEEE, pp 1–8
- Dale M, Stepney S, Miller JF, Trefzer M (2017) Reservoir computing in materio: a computational framework for in materio computing. In: *International joint conference on neural networks, IJCNN*, pp 2178–2185
- Dale M, Stepney S, Miller JF, Trefzer M (2018) A Substrate-independent framework to characterise reservoir computers. [arXiv:1810.07135](https://arxiv.org/abs/1810.07135)
- Devadas S, Ghosh A, Keutzer K (1994) Logic synthesis. McGraw-Hill Inc., New York
- Goldman BW, Punch WF (2013) Length bias and search limitations in Cartesian genetic programming. In: *Proceeding of the fifteenth annual conference on genetic and evolutionary computation conference*. ACM, pp 933–940
- Greenwood GW, Tyrrell AM (2007) Introduction to evolvable hardware - a practical guide for designing self-adaptive systems. Wiley, New York
- Haddow PC, Tyrrell AM (2011) Challenges of evolvable hardware: past, present and the path to a promising future. *Genet Program Evol Mach* 12(3):183–215
- Harding S, Miller JF (2004) Evolution in materio: a tone discriminator in liquid crystal. In: *Proceedings of the congress on evolutionary computation 2004 (CEC'2004)*, vol 2, pp 1800–1807
- Harding S, Miller JF (2005) Evolution in materio : a real time robot controller in liquid crystal. In: *Proceedings of NASA/DoD conference on evolvable hardware*, pp 229–238
- Harding SL, Miller JF (2007) Evolution in materio: evolving logic gates in liquid crystal. *Int J Unconv Comput* 3(4):243–257
- Harding SL, Miller JF, Rietman EA (2008) Evolution in materio: exploiting the physics of materials for computation. *Int J Unconv Comput* 4(2):155–194
- Harding S, Leitner J, Schmidhuber J (2013) Cartesian genetic programming for image processing. In: *Genetic programming theory and practice*, vol X. Springer, pp 31–44
- Jaeger H (2001) The 'echo state' approach to analyzing and training recurrent neural networks. Tech. Rep. GMD 148, German National Research Center for Information Technology
- Khan MM, Khan GM, Miller JF (2010a) Evolution of neural networks using Cartesian genetic programming. In: *Proceedings of the IEEE congress on evolutionary computation, CEC*, pp 1–8

- Khan MM, Khan GM, Miller JF (2010b) Evolution of optimal ANNs for non-linear control problems using Cartesian genetic programming. In: Proceedings of the 2010 international conference on artificial intelligence, pp 339–346
- Khan MM, Ahmad AM, Khan GM, Miller JF (2013) Fast learning neural networks using Cartesian genetic programming. *Neuro-computing* 121:274–289
- Kotsialos A, Massey MK, Qaiser F, Zeze DA, Pearson C, Petty MC (2014) Logic gate and circuit training on randomly dispersed carbon nanotubes. *Int J Unconv Comput* 10:473–497
- Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
- Leitner J, Harding S, Förster A, Schmidhuber J (2012) Mars terrain image classification using Cartesian genetic programming. In: 11th International symposium on artificial intelligence, robotics and automation in space (i-SAIRAS)
- Lukoševičius M, Jaeger H, Schrauwen B (2012) Reservoir computing trends. *KI-Künstliche Intelligenz* 26(4):365–371
- Lykkebø O, Tufte G (2014) Comparison and evaluation of signal representations for a carbon nanotube computational device. In: 2014 IEEE international conference on evolvable systems (ICES), pp 54–60
- Maass W (2011) Liquid state machines: motivation, theory, and applications. In: Cooper SB, Sorbi A (eds) Computability in context. Computation and logic in the real world. Imperial College Press, London, pp 275–296
- Massey MK, Kotsialos A, Qaiser F, Zeze DA, Pearson C, Volpati D, Bowen L, Petty MC (2015) Computing with carbon nanotubes: optimization of threshold logic gates using disordered nanotube/polymer composites. *J Appl Phys* 117(13):134903
- Miller JF (ed) (2011) Cartesian genetic programming. Springer, New York
- Miller JF, Downing K (2002) Evolution in materio: looking beyond the silicon box. In: Proceedings of NASA/DoD evolvable hardware workshop, pp 167–176
- Miller JF, Hartmann M (2001) Untidy evolution: evolving messy gates for fault tolerance. In: Evolvable systems: from biology to hardware, LNCS, vol 2210. Springer, pp 14–25
- Miller JF, Thomson P (2000) Cartesian genetic programming. In: Proceedings of the European conference on genetic programming, vol 1820. Springer, pp 121–132
- Miller JF, Smith S (2006) Redundancy and computational efficiency in Cartesian genetic programming. *IEEE Trans Evolut Comput* 10(2):167–174
- Miller JF, Job D, Vassilev VK (2000) Principles in the evolutionary design of digital circuits—part I. *Genet Program Evol Mach* 1(1–2):7–35
- Miller JF, Harding SL, Tufte G (2014) Evolution-in-materio: evolving computation in materials. *Evolut Intell* 7:49–67
- Mohid M, Miller J (2015) Evolving robot controllers using carbon nanotubes. In: Proceedings of the 13th European conference on artificial life (ECAL2015). MIT Press, pp 106–113
- Mohid M, Miller J (2015) Solving even parity problems using carbon nanotubes. In: Proceedings 15th workshop on computational intelligence (UKCI). IEEE Press
- Mohid M, Miller JF, Harding SL, Tufte G, Lykkebø OR, Massey MK, Petty MC (2014a) Evolution-in-materio: solving machine learning classification problems using materials. In: Parallel problem solving from nature - PPSN XIII—13th international conference, proceedings, LNCS, vol 8672. Springer, pp 721–730
- Mohid M, Miller J, Harding S, Tufte G, Lykkebø O, Massey M, Petty M (2014b) Evolution-in-materio: a frequency classifier using materials. In: Proceedings of the 2014 IEEE international conference on evolvable systems (ICES): from biology to hardware. IEEE Press, pp 46–53
- Mohid M, Miller J, Harding S, Tufte G, Lykkebø O, Massey M, Petty M (2014c) Evolution-in-materio: solving bin packing problems using materials. In: Proceedings of the 2014 IEEE international conference on evolvable systems (ICES): from biology to hardware. IEEE Press, pp 38–45
- Mohid M, Miller J, Harding S, Tufte G, Lykkebø O, Massey M, Petty M (2014d) Evolution-in-materio: solving function optimization problems using materials. In: 2014 14th UK workshop on computational intelligence (UKCI). IEEE Press, pp 1–8
- Pask G (1958) Physical analogues to the growth of a concept. Mechanisation of thought processes, no. 10 in national physical laboratory symposium. Her Majesty's Stationery Office, London, pp 877–922
- Ryser-Welch P, Miller JF, Swan J, Trefzer MA (2016) Iterative Cartesian genetic programming: creating general algorithms for solving travelling salesman problems. *Proc Eur Conf Genet Program LNCS* 9594:294–310
- Schrauwen B, Verstraeten D, Van Campenhout J (2007) An overview of reservoir computing: theory, applications and implementations. In: Proceedings of the 15th European symposium on artificial neural networks, pp 471–482
- Sentovich EM, Singh KJ, Lavagno L, Moon C, Murgai R, Saldanha A, Savoj H, Stephan PR, Brayton RK, Sangiovanni-Vincentelli A (1992) Sis: a system for sequential circuit synthesis. University of California, Berkeley Tech. rep
- Stanley KO (2007) Compositional pattern producing networks: a novel abstraction of development. *Genet Program Evol Mach* 8(2):131–162
- Thompson A (2001) Hardware evolution: automatic design of electronic circuits in reconfigurable hardware by artificial evolution. Springer, New York
- Thompson A, Layzell P, Zebulum RS (1999) Explorations in design space: unconventional electronics design through artificial evolution. *IEEE Trans Evolut Comput* 3(3):167–196
- Trefzer MA, Tyrrell AM (eds) (2015) Evolvable hardware—from practice to application. Natural computing series. Springer, New York
- Turner AJ, Miller JF (2013) The importance of topology evolution in neuroevolution: a case study using Cartesian genetic programming of artificial neural networks. In: Bramer M, Petridis M (eds) Research and development in intelligent systems, vol XXX. Springer, New York, pp 213–226
- Turner AJ, Miller JF (2014a) NeuroEvolution: the importance of transfer function evolution and heterogeneous networks. In: Proceedings of the 50th anniversary convention of the AISB, pp 158–165
- Turner AJ, Miller JF (2014b) Recurrent Cartesian genetic programming. In: 13th International conference on parallel problem solving from nature (PPSN 2014), LNCS, vol 8672, pp 476–486
- Turner AJ, Miller JF (2015a) Neutral genetic drift: an investigation using Cartesian genetic programming. *Genet Program Evol Mach* 16(4):531–558
- Tyrrell AM, Trefzer MA (2015b) Evolvable hardware: from practice to application. Springer, Berlin
- Vašíček Z (2015) Cartesian GP in optimization of combinational circuits with hundreds of inputs and thousands of gates. *Proc Eur Conf Genet Program LNCS* 9025:139–150
- Vašíček Z, Sekanina L (2011) Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genet Program Evol Mach* 12(3):305–327
- Vassilev VK, Miller JF (2000) The advantages of landscape neutrality in digital circuit evolution. In: Proceedings of the international conference on evolvable systems, LNCS, vol 1801. Springer, pp 252–263
- Vissol-Gaudin E, Kotsialos A, Massey MK, Zeze DA, Pearson C, Groves C, Petty MC (2016) Data classification using carbon-

- nanotubes and evolutionary algorithms. In: Parallel problem solving from nature—PPSN XIV—14th international conference, Edinburgh, UK, September 17–21, 2016, Proceedings, Lecture notes in computer science, vol 9921. Springer, pp 644–654
- Walker JA, Hilder JA, Tyrrell AM (2009) Towards evolving industry-feasible intrinsic variability tolerant CMOS designs. In: 2009 IEEE congress on evolutionary computation, pp 1591–1598
- Yu T, Miller J (2001) Neutrality and the evolvability of boolean function landscape. Genetic programming. Lecture notes in computer science, vol 2038. Springer, Berlin, pp 204–217
- Yu T, Miller JF (2006) Through the interaction of neutral and adaptive mutations, evolutionary search finds a way. *Artif Life* 12(4):525–551

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.