
Genetic Programming and Evolvable Machines, 1, 259–288, 2000

Principles in the Evolutionary Design of Digital Circuits—Part II

JULIAN F. MILLER j.miller@cs.bham.ac.uk
School of Computer Science, The University of Birmingham, Birmingham, B15 2TT, England

DOMINIC JOB d.job@dcs.napier.ac.uk
School of Computing, Napier University, Edinburgh, EH14 1DJ, Scotland

VESSELIN K. VASSILEV v.vassilev@dcs.napier.ac.uk
School of Computing, Napier University, Edinburgh, EH14 1DJ, Scotland

Received August 6, 1999; Revised September 2, 1999

Abstract. In a previous work it was argued that by studying evolved designs of gradually increasing scale, one might be able to discern new, efficient, and *generalisable* principles of design. These ideas are tested in the context of designing digital circuits, particularly arithmetic circuits. This process of discovery is seen as a *principle extraction loop* in which the evolved data is analysed both phenotypically and genotypically by processes of data mining and landscape analysis. The information extracted is then fed back into the evolutionary algorithm to enhance its search capabilities and hence increase the likelihood of identifying new principles which explain how to build systems which are too large to evolve.

Keywords: evolutionary algorithms, digital circuits, fitness landscapes, case based reasoning, principle extraction

1. Introduction

This paper is a companion to an earlier one, Part I [17]. In that work it was argued that an evolutionary algorithm combined with the concept of *assemble-and-test* enables an exploration of a much larger space of possible designs than that allowed by conventional rule-based methods. These ideas have been largely adopted in the field of Evolvable Hardware [24, 30].

Part I also discussed the particular characteristics of an effective evolutionary algorithm which can be used to produce radically new designs for certain types of digital circuits, namely arithmetic circuits. These are interesting subjects for evolutionary design because they are conventionally thought of as being modular in construction and thus can be easily used as building blocks to create arbitrarily large systems. It was precisely for this reason that they may afford an answer to the fundamental question (TFQ) posed in Part I:

> "Can we by evolving a series of sub-systems of increasing size, extract the general principle and hence discover new principles?"

Although novel and efficient designs for circuits as large as the three-bit multiplier were obtained in Part I (some were 20% more efficient than the best conventional), the evolutionary process of design was very time consuming. Thus evolving efficient larger circuits appears to be almost insurmountable. However, it is precisely digital circuits with larger numbers of inputs that are especially interesting to attempt to evolve. The reasons are:

1. The efficiency of the evolved designs may grow with its size.
2. Larger and more efficient building blocks are more likely to appear in evolved circuits with greater size.
3. Evolving increasingly larger circuits gives a better chance of finding whether TFQ can be answered in the affirmative.

An additional problem with designing larger digital circuits using evolutionary techniques is that as the number of inputs grows the time taken for fitness evaluation increases exponentially.

In this paper two investigations into different aspects of this problem were undertaken. The first investigation examines the nature of the fitness landscapes and attempts to understand the structure of these landscapes in terms of their smoothness, ruggedness and neutrality. It is reasonable to assume that the interplay between these landscape characteristics should lead to the development of more efficient evolutionary search. The second investigation examines the nature of the phenotypes themselves and attempts to discover useful sub-structures and methods by which they can be reused to create larger circuits. This also facilitates understanding of the novel and efficient designs.

These two investigations are an important part of the cycle of evolutionary discovery shown in Figure 1 and together with the findings in Part I [17] complete the *principle extraction loop*.

In Section 2 a summary is given of the way in which digital circuits are encoded into genotypes and some details of the evolutionary algorithm used. The techniques of landscape analysis developed in [33, 35] are discussed in Section 3. The analysis is used in finding principles that should lead to a better understanding of the nature of the problem of evolving digital circuits, and hence, effective evolutionary search. The process of discerning design rules and principles from the

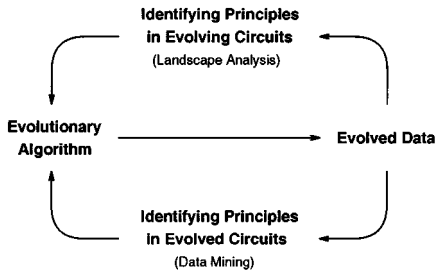

Figure 1. The principle extraction loop.

evolved data can be seen as a form of data mining [9]. This can make recommendations about useful components and sub-structures which feedback into the evolutionary algorithm and hence improve the evolvability of the circuits. How this in turn enhances our ability to understand the nature of new designs is examined in Section 4. Finally, the paper ends with conclusions and suggestions for future work.

2. Digital circuit evolution and evolved data

The encoding of a digital combinational circuit into a genotype was discussed in detail in Part I. The digital logic circuit is considered as a graph and is a particular case of a more general graph based computational model called *Cartesian Genetic Programming* (CGP) [16]. The genotype is a linear string of integers representing the connections and functions of a rectangular array of gates. It is characterised by three parameters: the *number of columns*, the *number of rows* and *levels-back*. The first two are merely the dimensions of the rectangular array and the latter is a parameter that controls the internal connectivity. It determines how many columns of cells to the left of a particular cell may have their outputs connected to the inputs of that cell. The genotype and the mapping process of genotype to phenotype are illustrated in Figure 2.

The n_I primary circuit inputs $X_1, X_2, \ldots, X_{n_I}$ are allowed to be connected to the input of any cell or any of the n_O primary circuit outputs $Y_1, Y_2, \ldots, Y_{n_O}$. The cells c_{ij} may implement any of the binary functions listed in Table 1. The overbar

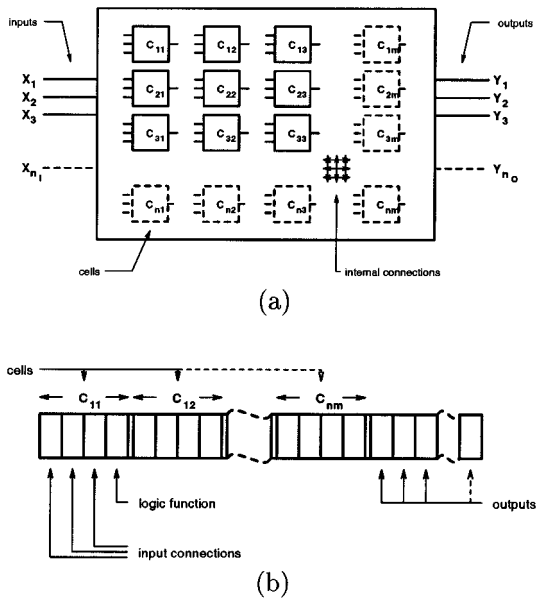

Figure 2. The genotype-phenotype mapping: (a) a $n \times m$ geometry of logic cells with n_I inputs and n_O outputs, and (b) the genotype structure of the array.

Table 1. Allowed cell functions

Letter	Function	Letter	Function
0	0	10	$a \oplus b$
1	1	11	$a \oplus \bar{b}$
2	a	12	$a + b$
3	b	13	$a + \bar{b}$
4	$\bar{a}$	14	$\bar{a} + b$
5	$\bar{b}$	15	$\bar{a} + \bar{b}$
6	$a \cdot b$	16	$a \cdot \bar{c} + b \cdot c$
7	$a \cdot \bar{b}$	17	$a \cdot \bar{c} + \bar{b} \cdot c$
8	$\bar{a} \cdot b$	18	$\bar{a} \cdot \bar{c} + b \cdot c$
9	$\bar{a} \cdot \bar{b}$	19	$\bar{a} \cdot \bar{c} + \bar{b} \cdot c$

represents inversion, and the symbols "$\cdot$", "$\oplus$", and "$+$", represent the operations AND, XOR, and OR, respectively. Functions 16 to 19 are all binary multiplexers with various inputs inverted. The multiplexer (MUX) implements a simple IF-THEN statement (i.e. IF $c = 0$ THEN a ELSE b).

Figure 3 shows the genotype and phenotype for a small gate array consisting of four logic cells. The logic cells in this case have functions XOR, AND, or MUX. **A**, **B**, and $\mathbf{C_{in}}$ represent the primary inputs. $\mathbf{C_{out}}$ and **S** (Sum) are the output bits of the adder. For example the upper right cell (output 5) below has input connections 3, 2, 1. This means that the first input is connected to the output of the cell with output label 3 (upper left), the second input is connected to the primary input $\mathbf{C_{in}}$, and the third input is connected to primary input **B**. The function of each cell is expressed as the fourth gene associated with each cell (shown in bold typeface).

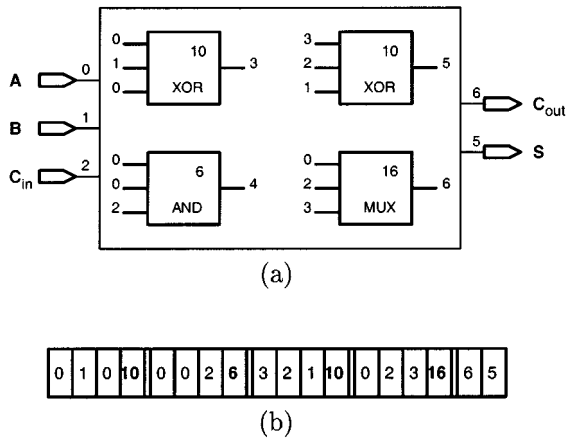

Figure 3. The genotype-phenotype mapping of evolved one-bit adder with carry: (a) gate array representation, and (b) genotype representation.

The primary outputs of the gate array are also expressed as connections. For example, $\mathbf{C_{out}}$ is connected to the output of the cell with output label 6. It is important to emphasise that cell outputs may be re-used and when a program is used to evolve the genotypes the amount of re-use of sub-calculations can be automatically determined.

All functions are specified by a truth table. The fitness of a genotype is the number of correct output bits. Thus for the one-bit adder with carry seen in Figure 3 there are 8 input cases and 2 outputs, this gives 16 output bits. A fully correct circuit would have fitness 16.

The evolved circuit designs are produced by Cartesian Genetic Programming with truncation selection and mutation. The latter is defined as a percentage of the genes in a single genotype which are to be randomly mutated. The population consists of $1 + \lambda$ genotypes where λ is usually about 4. Initially the elements of the population are chosen at random. To update the population, the operator for mutation is applied to the fittest genotype, and thus, an offspring is generated. The offspring together with the parent constitute the new population. This mechanism of population update has some similarities with that employed in other evolutionary techniques such as $(1 + \lambda)$ Evolution Strategy [1, 23] and the Breeder Genetic Algorithm [21].

This algorithm has allowed the automatic discovery of highly efficient circuits that are very unusual in construction. The one-bit adder that was used as an example above (Figure 3) was actually evolved and it required two gates less than the conventional design. The MUX gate occurs in an unfamiliar configuration and it implied that these gates are very useful building blocks for the construction of adder circuits. It was surprising that this one-bit adder automatically *emerged* as a building block in an evolved two-bit adder. This suggested that it would be worthwhile attempting to evolve larger and more complex circuits, such as the two-bit and three-bit multipliers. Indeed it was found that some of the evolved three-bit multipliers were 20% more efficient than the most efficient conventional design. The efficiency with which the evolutionary algorithm was able to solve these difficult problems suggested that this algorithm might also be very effective in evolving parity functions. It is well known that parity functions are particularly difficult to evolve under certain conditions. For further details see Part I [17].

Figure 4 shows the symbols used to represent logic gates in circuit diagrams. Note that small circles may appear on some of the inputs and outputs of these devices; this indicates inversion.

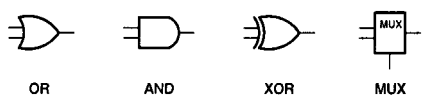

Figure 4. Binary circuit symbols.

3. Identifying principles in evolving circuits

The evolutionary design of digital circuits can be considered as a search on a fitness landscape. It has been shown that the structure of fitness landscapes affects the ability of the evolutionary algorithms to search [11, 15, 20]. The evolutionary search is easier when the landscapes are smoother. However, the search becomes more difficult on more rugged landscapes since the population can be trapped in local optima. Recently, the importance of neutrality in landscapes has been suggested [22] but it is still not clear what its role is in evolutionary search.

The circuit evolution landscapes are quite different from many recently studied landscapes [6, 11, 36]. The difference originates in the structure of the genotypes which are strings defined over two completely different alphabets that are responsible for the functionality and connectivity of the array of logic cells (see Section 2). This gives rise to a complicated relationship between the genes within the genotype which makes the study of the landscapes much more convoluted.

In Vassilev *et al*. [34, 35] a model for studying the structure of circuit evolution landscapes has been introduced. It is based on the idea that a landscape might be decomposed to isotropic subspaces [6]. Thus it became possible to investigate the ruggedness of these landscapes in general, by measuring their autocorrelations and the corresponding amplitude spectra derived from the landscapes' Fourier transformations. Here the model is employed to investigate the structure of circuit evolution landscapes in terms of the *interplay* between smoothness, ruggedness and neutrality. The smoothness and ruggedness are related to the fitness differences between neighbouring points whereas the neutrality refers to the flat landscape areas [22, 27]. The study of the characteristics of these landscapes is an important concern in digital circuit evolution both for their scalability and in the importance of chosing appropriate sets of logic functions used in the assembly of the digital circuits.

The section concentrates on landscapes associated with five digital circuits—a two-bit multiplier (Figure 18b), two three-bit multipliers (Figures 19 and 20), and two four-bit parity functions (Figure 21)—which are evolved by evolutionary algorithms. The landscapes are generated by a onepoint mutation operator, since this is the only evolutionary operator in the optimisation schemes employed. The interplay of the landscape smoothness, ruggedness and neutrality is studied by an information analysis based on that given by Vassilev [32]. It is shown that the digital circuit evolution landscapes are characterised by vast and sharply differentiated landscape plateaus. It is also shown that the continuity of these landscapes depends on the scale and the set of logic functions used in the assembly of digital circuits.

The next subsection explains the notion of a fitness landscape in evolutionary computation. Section 3.2 gives the information analysis for fitness landscapes. The model of circuit evolution landscapes is briefly described in Section 3.3. Section 3.4 shows the results of applying the analysis to these landscapes. Finally, in Section 3.5 a summary of the main conclusions is given.

3.1. Fitness landscapes

The notion of a *fitness landscape*, introduced by Wright [37], has become an important concept in evolutionary computation. The metaphor is taken from biology and it expresses the idea that evolution can be considered as a population flow on a surface in which the altitude of a point qualifies how well the corresponding organism is adapted to an environment. In evolutionary computation the fitness landscapes are simply search spaces defined over elements called *phenotypes* which are represented by their *genotype*. The genotype is a sequence of elements taken from a finite alphabet. A *fitness value* is assigned to each genotype and the evolutionary algorithm refers to these values when deciding which phenotypes should survive and reproduce. The fitness value of a genotype is evaluated by a *fitness function*, f, which measures how *good* the encoded phenotype is. The population of genotypes flows on a landscape guided by an evolutionary operator. Hence, the connections between the landscape points are determined by an operator that is employed to search in the landscape.

As suggested by Stadler [26], Stadler and Wagner [29] a fitness landscape, $\mathscr{L}$, can be defined on a graph, $\mathscr{G}_f = (V, E)$, whose vertices are genotypes labeled with fitness values, and the connections are defined by the evolutionary operator which agrees with the concept "one operator, one landscape" [10]. The genotype representation, the neighbourhood relation, and the fitness function define the structure of a fitness landscape. The structure can be specified in terms of three characteristics of fitness landscapes. These are the landscape smoothness, ruggedness and neutrality.

The landscape structure can be studied by quantifying the time series that is obtained by sampling values along a random walk on the landscape [26, 36]. Weinberger [36] has pointed out that the theory of stationary random processes can be applied to the study of random walks on landscapes, when the landscapes are *statistically isotropic*. A landscape is statistically isotropic when the sequence of fitness values, obtained by a random walk on the landscape, forms a stationary random process for the assumed joint distribution of fitness values.

3.2. Information analysis of landscapes

Various techniques for statistical analysis of the structure of fitness landscapes have been proposed. Weinberger [36] has investigated how the autocorrelation function of the fitness values of points along the steps of a random walk relates to the ruggedness of the examined landscape. The autocorrelation function as a measure of landscapes has been also studied by Stadler [26] who suggested that the autocorrelations can be used to estimate the amplitude spectra derived from the Fourier transforms of the landscapes [7]. Another landscape analysis, based on the Box and Jenkins [2] approach, has been proposed by Hordijk [5]. The basic idea, initially suggested by Weinberger [36], has been to explore the landscape structure

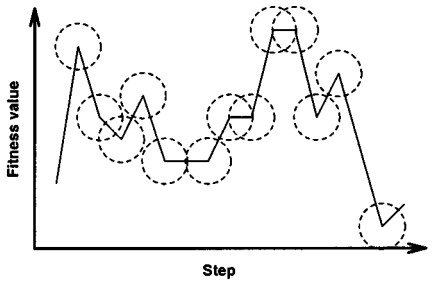

Figure 5. A sequence of fitness values as an ensemble of objects.

by studying the corresponding autoregressive moving average (ARMA) model which in fact characterises the landscape ruggedness much more precisely.

An important feature of the aforementioned techniques is that they study the structure of the landscapes in terms of their ruggedness. Here an information analysis is employed to investigate the interplay of smoothness, ruggedness and neutrality of the fitness landscapes. The underlying idea is to consider a fitness landscape as an ensemble of objects as is illustrated in Figure 5. Each object consists of a landscape point and its nearest neighbours. The figure shows how a landscape path obtained by a walk on the landscape can be presented as an ensemble of objects. To study how the ruggedness and smoothness of a landscape are related to the landscape neutrality, two subsets of the original ensemble are specified as demonstrated in Figure 6. For each subset an *information function* is constructed, which reveals how the estimate of the entropy of this sub-ensemble changes when the neutrality of the landscape is increased.

Consider a sequence of fitness values $\{f_t\}_{t=0}^{n}$, real numbers from the interval $\mathscr{I}$, that are obtained by a *walk* on a landscape, $\mathscr{L}$. The sequence is a time series that represents a path in $\mathscr{L}$, and contains information about the structure of the landscape. The aim is to extract this information, by representing the time series as

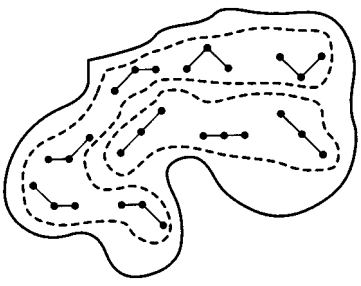

Figure 6. A fitness landscape as an ensemble of objects in two categories.

an ensemble of objects. The ensemble can be defined as a string, $S(\varepsilon) = s_1 s_2 s_3 \ldots s_n$, of symbols $s_i \in \{\bar{1}, 0, 1\}$, and they are obtained by function

$$\Psi_{f_t}(i, \varepsilon) = \begin{cases} \bar{1}, & \text{if } f_i - f_{i-1} < -\varepsilon \\ 0, & \text{if } |f_i - f_{i-1}| \leq \varepsilon \\ 1, & \text{if } f_i - f_{i-1} > \varepsilon \end{cases} \tag{1}$$

in the following way

$$s_i = \Psi_{f_t}(i, \varepsilon) \tag{2}$$

for any fixed ε. The parameter ε is a real number from the interval $[0, l_{\mathcal{F}}]$, where $l_{\mathcal{F}}$ is the length of the interval $\mathcal{F}$. Note that the parameter ε determines the accuracy of calculation of the string $S(\varepsilon)$. If $\varepsilon = 0$, the function Ψ_{f_t} will be very sensitive to the differences between the fitness values and $S(\varepsilon)$ will be determined as precisely as it is possible. When the parameter ε is $l_{\mathcal{F}}$, $S(\varepsilon)$ will be a string of 0s.

The string $S(\varepsilon)$ contains information about the structure of the landscape. Note that the function Ψ_{f_t} associates each edge of the path with an element from the set $\{\bar{1}, 0, 1\}$. Consequently, each object of the path is represented by a string, $s_i s_{i+1}$, which is a sub-block of length two of the string $S(\varepsilon)$. One can even think of $S(\varepsilon)$ as a sequence of elements (a sample) of the *incidence* matrix of the landscape underlying graph. The incidence matrix of a landscape is related to the landscape's graph *Laplacian* matrix whose eigenvectors are the orthogonal basis of eigenfunctions of the Fourier transform of the landscape [26].

3.2.1. Information characteristics. Two entropic measures of the ensemble of the sub-blocks of length two of $S(\varepsilon)$ can be devised. These are

$$H(\varepsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]}, \tag{3}$$

and

$$h(\varepsilon) = -\sum_{p = q} P_{[pq]} \log_3 P_{[pq]}, \tag{4}$$

referred to as the first (FEM) and second (SEM) entropic measures, respectively. The FEM is an estimate of the ruggedness of a landscape with respect to the landscape neutrality, while the SEM relates to the interplay of smoothness and the neutrality of the landscape. The probabilities $P_{[pq]}$ are frequencies of the possible blocks pq of elements from set $\{\bar{1}, 0, 1\}$. They are defined as

$$P_{[pq]} = \frac{n_{[pq]}}{n} \tag{5}$$

where $n_{[pq]}$ is the number of sub-blocks pq in the string $S(\varepsilon)$. The logarithms from equations 3 and 4 are taken with bases six and three, respectively, since these are the numbers of blocks of length two composed by two different and respectively two equal symbols taken from $\{\bar{1}, 0, 1\}$.

3.2.2. Information functions. The FEM and the SEM characterise the time series $\{f_t\}_{t=0}^{n}$ with a certain accuracy. The accuracy of the estimations can be varied by the parameter ε that in turn defines the entropic measures as functions of the accuracy. One can think of the parameter ε as a magnifying glass through which the landscape can be observed. For small values of ε, the function Ψ_{f_t} from equation 1 will be very sensitive to the difference between the fitness values, i.e. the glass will make each element of the landscape visible. If ε is zero then the accuracy of the estimations, obtained by $H(\varepsilon)$ and $h(\varepsilon)$, is high. In contrast, for $\varepsilon = l_{\mathcal{F}}$, the FEM and the SEM of $S(\varepsilon)$ are 0, i.e. for such ε the landscape path will be determined as relatively flat. The role of the parameter ε is illustrated in Figure 7 which shows a landscape profile for different values of ε.

3.2.3. Information Analysis. The analysis starts by performing a random walk on the landscape, using the evolutionary operator by which the neighbourhood relationship of the landscape points is defined. For each step, the fitness value of the current point is recorded. Thus, for a certain number of steps a time series will be obtained. Then the information functions $H(\varepsilon)$ and $h(\varepsilon)$ are calculated.

To quantify the entropic measure of a landscape, represented as an ensemble of objects, does not necessarily imply measuring the ruggedness of the landscape. At this point, the information analysis differs significantly from many other statistical approaches. For instance, consider the autocorrelation function, which reveals how the correlation is "spread" over the landscape. The lower the correlations are, the more rugged is the landscape. The information analysis is different. It gives us a notion of what is the relation between the landscape smoothness, ruggedness and

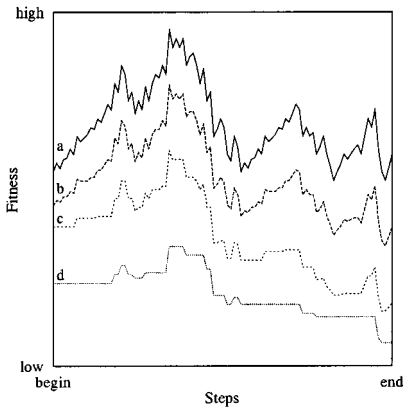

Figure 7. The profile of a landscape path for different values of parameter ε: (a) 0, (b) 0.02, (c) 0.05, and (d) 0.1.

neutrality. For a better understanding of the analysis some interesting properties of functions $H(\varepsilon)$ and $h(\varepsilon)$ are given below, some of which are described in Vassilev [31, 32].

Let $\varepsilon^\star$ be the lowest value of ε for which $S(\varepsilon)$ is a string of 0's. Firstly, the FEM and the SEM of a time series generated by a *regular* walk on the landscape are positive constants for each $\varepsilon \in [0, \varepsilon^\star)$. The term *regularity* is defined as follows: a time series, $\{f_t\}_{t=0}^n$, is generated by a *regular* walk (it has nothing to do with the notion of regular graphs) on a landscape when the time series obeys

$$f_{t+1} = f_t \pm \kappa c, \tag{6}$$

where c is a constant and κ is a variable which can be 0, 1, or -1. If equation 6 is not fulfilled, the landscape path is generated by an *irregular* walk.

In practice regular walks on a landscape are a rare occurrence. Consider equation 6 in the form $f_{t+1} = f_t + \kappa \sum_i c_i$ where c_i are different constants. The degree of regularity of a landscape is the number of different κc_i, that is to say the number of all possible differences of fitness values. For instance, the degree of regularity of *Nk* landscapes generated by onepoint mutation is low, and it decreases as k increases from 0 to $N-1$ since the number of possible fitness values increases with a higher than linear rate (Figure 8).

Secondly, there exists a sequence of fitness values, and parameters ε_1 and ε_2, such that $H(\varepsilon_1) < H(\varepsilon_2)$, where $0 \le \varepsilon_1 < \varepsilon_2 \le l_{\mathscr{F}}$. The landscapes associated with this class are characterised by relatively low neutrality. For such landscapes, $H(\varepsilon)$ is an increasing function for small values of ε, since the landscape as an ensemble consists mainly of two types of objects. An example of such landscapes is given in Figure 8, which shows the functions $H(\varepsilon)$ of *Nk* landscapes for different values of k.

Thirdly, if $H(\varepsilon) = \log_6 2$ and $h(\varepsilon) = 0$ for $\varepsilon = 0$, then the explored time series is maximally multimodal or an increasing/decreasing step function.

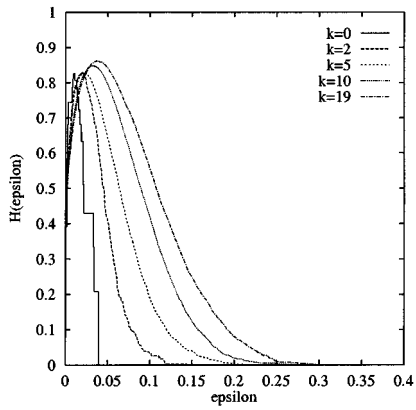

Figure 8. The information function $H(\varepsilon)$ of *Nk* landscapes with random neighbourhoods for $N = 20$ and different values of k.

Lastly, the neutrality prevails over the ruggedness in a time series, if $H(\varepsilon)$ is a decreasing function.

3.3. *Circuit evolution landscapes*

Consider the genotype representation of an idealised field-programmable gate array (n rows, m columns, n_I inputs, n_O outputs) described in Section 2. The genotype is a composition of three different parts which are responsible for first, the gates functionality, second, the array internal connectivity, and third, the array outputs. The reason for splitting the genotype into parts is the difference in the purposes of the parts. For convenience, the term *chromosome* will be used to mean a genotype part. The chromosomes have different lengths and they are defined over two different alphabets. The "gate functionality" chromosomes are strings over alphabet α with length the number of gates. The alphabet size l_α is the number of allowed logic functions used in the circuit design. The "internal connectivity" and "array outputs" chromosomes are defined over alphabet β, and they are strings with length the number of gates and the number of array outputs, respectively. The alphabet β is related to the size of the neighbourhood of the cells and array outputs, which is dependent upon the levels-back parameter. The alphabet is a set of integers, which are reference numbers of the elements of a neighbourhood. Hence, the size of β is

$$l_\beta = \begin{cases} nL, & \text{if } L \le m \\ nm + n_I, & \text{otherwise} \end{cases}, \tag{7}$$

where n is the number of rows, m is the number of columns, n_I is the number of inputs of the gate array, and L is the levels-back.

A major difficulty in studying the structure of circuit evolution landscapes stems from the complex relations between the genes within the genotypes. It determines a complicated *mixture* of epistatic characteristics of the genotype which is an almost insurmountable impediment if the genotype is considered as an inseparable chain. The landscapes are highly *non-isotropic* which was also revealed by measured autocorrelations of random walks with length 100,000. It was observed that the autocorrelation functions differed from each other in a significantly wide range. To avoid the "vagueness" in the definition of circuit evolution landscapes, the genotype space is split into three partitions as was done in Stadler and Grünter [28].

Since each genotype consists of three chromosomes it is assumed that the original landscape for a given evolutionary operator is a superposition of three configuration spaces defined over alphabets α and β. Let the hypercubes $\mathscr{G}_{l_\alpha}^{nm}$, $\mathscr{G}_{l_\beta}^{3nm}$, and $\mathscr{G}_{l_\beta}^{n_O}$ represent the configuration spaces of the chromosomes responsible for functionality, connectivity, and output connections, respectively. If ϕ denote an evolutionary operator, then for each hypercube a family of landscapes that repre-

sents the genotype space with respect to the hypercube can be defined. The landscapes are

$$\begin{aligned}
&(1)\quad \left\{\mathcal{G}_{f_i}, f_i, \phi\right\}_{i=0}^{l_\beta^{3nm+n_O}-1}\\
&(2)\quad \left\{\mathcal{G}_{g_i}, g_i, \phi\right\}_{i=0}^{l_\alpha^{nm} l_\beta^{n_O}-1}\\
&(3)\quad \left\{\mathcal{G}_{h_i}, h_i, \phi\right\}_{i=0}^{l_\alpha^{nm} l_\beta^{3nm}-1}.
\end{aligned} \tag{8}$$

The graphs $\mathcal{G}_{f_i}$, $\mathcal{G}_{g_i}$, and $\mathcal{G}_{h_i}$ are obtained by assigning each vertex from $\mathcal{G}_{l_\alpha}^{nm}$, $\mathcal{G}_{l_\beta}^{3nm}$, and $\mathcal{G}_{l_\beta}^{n_O}$, respectively, with a fitness value. The fitness values are provided by fitness functions $\{f_i\}_{\forall i}$, $\{g_i\}_{\forall i}$ and $\{h_i\}_{\forall i}$ which are defined as follows

$$\begin{aligned}
&(1)\quad \forall i\left(\mathbf{c}_i \in \mathcal{G}_{l_\beta}^{3nm} \times \mathcal{G}_{l_\beta}^{n_O}\right), \forall \mathbf{x} \in \mathcal{G}_{l_\alpha}^{nm} : f_i(\mathbf{x}) = F(\mathbf{x} \circ \mathbf{c}_i)\\
&(2)\quad \forall i\left(\mathbf{c}_i \in \mathcal{G}_{l_\alpha}^{nm} \times \mathcal{G}_{l_\beta}^{n_O}\right), \forall \mathbf{x} \in \mathcal{G}_{l_\beta}^{3nm} : g_i(\mathbf{x}) = F(\mathbf{x} \circ \mathbf{c}_i)\\
&(3)\quad \forall i\left(\mathbf{c}_i \in \mathcal{G}_{l_\alpha}^{nm} \times \mathcal{G}_{l_\beta}^{3nm}\right), \forall \mathbf{x} \in \mathcal{G}_{l_\beta}^{n_O} : h_i(\mathbf{x}) = F(\mathbf{x} \circ \mathbf{c}_i).
\end{aligned} \tag{9}$$

The function F is defined over the genotype space (the operator "$\circ$" is considered to merge the strings in a special way so that the genotype structure as given in Figure 2 is maintained) and it evaluates the number of correct output bits in the circuit. Note that for each family of landscapes a group of fitness functions is specified, and each fitness function estimates only a part of the genotype. Hence, its index is determined by the constant string $\mathbf{c}_i$ which is the remainder of the genotype.

3.4. Results

The analysis is applied to time series obtained by random walks on the three subspaces of the landscapes associated with evolved arithmetic and parity functions, described in Section 2. The random walks are performed by onepoint mutation applied only to the studied subspace. The length of each random walk is 100,000 steps. The onepoint mutation is an operator which changes the allele of one gene at each time step.

Figure 9 shows the information functions (a) $H(\varepsilon)$ and (b) $h(\varepsilon)$ of (1) functionality, (2) internal connectivity, and (3) output connectivity subspaces associated with the landscape of the two-bit multiplier depicted in Figure 18b. The subspaces are characterised with vast neutrality since the FEM $H(0)$ and the SEM $h(0)$ of each subspace are significantly higher than $\log_6 2$ and 0, respectively (see Section 3.2). The plots in Figure 9a also imply subspaces with significantly different profiles. For instance, the information functions $H(\varepsilon)$ of the functionality and output connectivity subspaces increase as ε increases from 0 to approximately 0.0157, while $H(\varepsilon)$

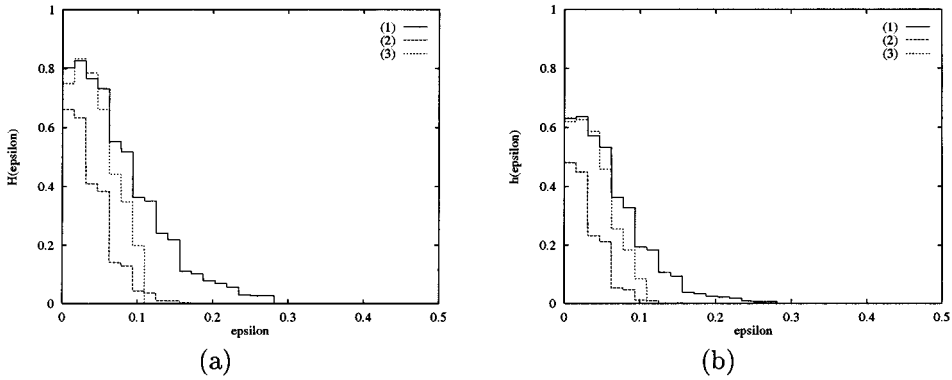

Figure 9. The information functions (a) $H(\varepsilon)$ and (b) $h(\varepsilon)$ of (1) functionality, (2) internal connectivity, and (3) output connectivity subspaces of the two-bit multiplier landscape, generated by onepoint mutation. The landscape originates from the two-bit multiplier depicted in Figure 18b.

of the internal connectivity subspace decreases as ε increases. Consequently, the neutrality in the internal connectivity subspace prevails over its ruggedness and smoothness, the latter is revealed by the plots in Figure 9b. This is not valid for the functionality and output connectivity subspaces, according to the properties of $H(\varepsilon)$ described in Section 3.2. However, the neutrality in the functionality subspace is higher since the information content of this subspace is not as close to $\log_6 2$ as $H(0)$ obtained for the output connectivity subspace.

The plots in Figure 9a also reveal that the degree of regularity in the two-bit multiplier subspaces is high, which together with the landscape neutrality implies sharply differentiated landscape plateaus. It suggests that $(1 + \lambda)$-ES (see Part I [17]) might be much more effective to search in these landscapes which agrees with the findings in [16].

The structure of the three-bit multiplier landscapes is similar to that of the two-bit multiplier. This is revealed by the information functions depicted in Figure 10. The figure represents the functions $H(\varepsilon)$ of (1) functionality, (2) internal connectivity, and (3) output connectivity subspaces of the landscapes associated with the three-bit multipliers of (a) 24 and (b) 21 gates, depicted in Figures 19 and 20, respectively. The constraints in evolving these arithmetic circuits are different which is revealed in the plots of the information functions. Figure 10 also shows that by increasing the scale, the regularity of multiplier landscapes decreases, which is to be expected since the truth table of these arithmetic functions increases with a rate higher than linearly. Consequently, by increasing the scale, the corresponding landscape becomes continuous and perhaps easier for evolutionary search.

The degree of regularity of digital circuit evolution landscapes is also related to the set of logic functions of the gates. Consider for instance the four-bit parity function. The circuit can be easily designed by a simple random search, when only XNOR gates are allowed (Figure 11a). However, the evolutionary design of this digital circuit, allowing only XNOR gates might be a difficult task. The reason is the

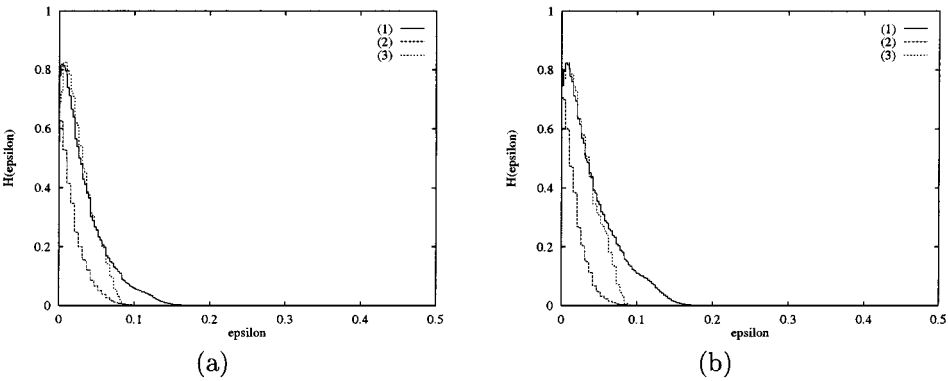

Figure 10. The information functions $H(\varepsilon)$ of (1) functionality, (2) internal connectivity, and (3) output connectivity subspaces of two three-bit multiplier landscapes, generated by onepoint mutation. The landscapes originates from the three-bit multipliers of (a) 24 and (b) 21 gates depicted in Figures 19 and 20, respectively.

regularity which together with the vast neutrality of the four-bit parity landscapes is an almost insurmountable impediment for evolutionary search. Figure 11 shows the information functions of four-bit parity landscapes using (a) only gate XNOR, and (b) gates {6, 9, 12, 15} (Table 1). It can be seen that the subspaces in Figure 11a are regular, which is not true for those characterised in Figure 11b, although the two digital circuits have identical functions (Figure 21).

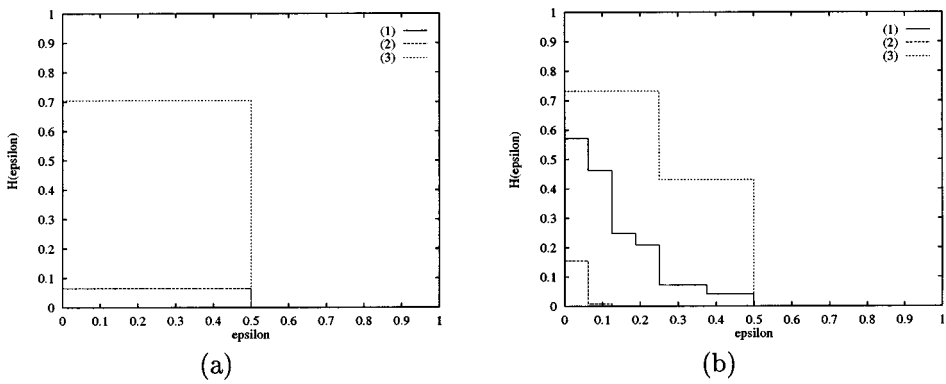

Figure 11. The information functions $H(\varepsilon)$ of (1) functionality, (2) internal connectivity, and (3) output connectivity subspaces of two four-bit parity landscapes, generated by onepoint mutation. The landscapes originates from four-bit parity functions designed using (a) only gate XNOR and (b) gates {6, 9, 12, 15} as given in Table 1.

3.5. Summary

The landscapes associated with evolving digital circuits, and particularly, arithmetic and parity functions, have been defined and investigated. It was shown that these landscapes are more properly considered as a superposition of three spaces—functionality, internal and output connectivity subspaces, each with different characteristics. The interplay of smoothness, ruggedness and neutrality for these subspaces was studied, and it was found that the circuit evolution landscapes consist of subspaces with a different structure. They are landscapes with vast neutrality and sharply differentiated plateaus. It was also found that the continuity of these landscapes is related to the scale of the evolved digital circuits and the choice of elementary gates. These findings should help to define more effective search strategies particularly as the scale of the problem is increased.

4. Identifying principles in evolved circuits

4.1. A problem of scale and a possible solution

In the design process it has long been accepted that the best way to solve a problem is to decompose the problem into several simpler sub-problems and solve these sub-problems. One problem with evolving digital circuit programs is that it is computationally expensive, particularly for larger programs. Since more complex functions and larger numbers of inputs require exponentially larger circuits to produce a solution there is a limit to the size and complexity of circuit program that can be evolved. This is referred to as the scaling problem.

Section 4 describes efforts to overcome the scaling problem. The approach attempts to decompose the solution programs produced by the evolutionary algorithm. This involves extracting meaningful sub-programs, or design principles, from the evolved solutions, and using them to try to solve the scaling problem and also to help in understanding the way the evolved solutions work.

Principle extraction and reuse is achieved by integration of evolutionary and Case-Based Reasoning (CBR) techniques. This section discusses the creation of a Case-Base that will allow for adaptation and reuse of evolved Binary Cartesian Genetic (BCG) programs [17], and the sub-programs within these programs, to create larger programs at a reasonable computational expense.

It was seen in Part I [17] that arithmetic adder and multiplier circuits are modular in construction and so are useful functions to study and refine techniques of principle extraction. Modularity by definition allows very large systems to be constructed by connecting modules together. This is clear that as multiplication is a process of repeated addition, multiplication circuits can be built by using AND gates to perform elementary one-bit multiplication and then binary full-adders connected in an arrangement called a cellular array. When biologically inspired algorithms such as evolutionary algorithms are allowed to design the building blocks and assemble the parts it gives rise to an amazing number of potentially new designs. This brings us to the fundamental question (TFQ) stated in Section 1. In

one instance this was positively answered in [9, 19] where it was shown that the principle of the ripple-carry adder could be inferred by studying evolved designs for one-bit and two-bit adders. This process of *data-mining* from evolved solutions potentially allows us to "close the loop" of principle extraction (Figure 1). The extracted principles by making recommendations as to useful components and sub-structures may feed back into the evolutionary algorithm.

These essential sub-structures when collected together and subjected to analysis might lead to the discovery of a new principle. This section discusses a "finger printing" technique applicable to the genotype discussed previously that reveals the type and frequency of embedded sub-structures. Initial examination of the principle extraction problem showed that by using this finger printing technique it was possible to find known human principles, and additionally find hitherto unknown principles.

Section 4.2 describes how CBR is a suitable technique for the automatic identification of principles. The methods that are used to process the evolved programs to create a Case-Base are described in Section 4.3. In Section 4.4 some results of the experiments and their analysis are presented. Section 4.5 discusses the achievements and limitations of the work and suggests scope for future work.

4.2. CBR as a principle identification technology

One potentially suitable solution to the scaling problem is to find a way to reuse evolved BCG programs using CBR. CBR is an artificial intelligence technique that is designed to reuse past experiences to solve new problems. It can provide answers to problems in poorly understood complex domains; it does not require a domain model or rules; and it can provide an explanation of its own reasoning. The reuse of old solutions to solve new problems is also the problem facing software reuse, as many old solutions need to be adapted to suit new problems.

CBR can provide selection, retrieval and adaptation of old software solutions to solve new problems and it has been successfully used as a reuse system for retrieving and adapting software artifacts [13, 26]. Case-Based reasoning has already been successfully applied to the understanding of evolutionary produced designs [8, 14]. This suggests that CBR could be used for understanding, retrieving and adapting evolutionary designs, to solve new problems. CBR also provides a scalable approach, and can be used to create designs larger than the designs that make up its source material (its Case-Base). CBR provides data mining, indexing, matching, retrieval and adaptation, and these techniques should assist the process of principle extraction and application.

CBR can partly address the problem of scaling up evolved digital circuit programs. The scaling problem might be overcome by effective reuse of principles contained in the evolved programs. Identifying these principles is however a very complex task. CBR relies upon Cases that have known structure, e.g. attribute value pairs. Evolved programs lack any "understanding" incorporated in their structure. All knowledge beyond their functionality must be identified before a

useful Case-Base can be made. These principles might be able to be recombined and adapted to create new designs for new scaled-up problems.

Evolutionary algorithms have been successfully used as a "knowledge lean" method to generate knowledge for a Case-Base in earlier research [8]. This paper differs significantly from previous work as the phenotypes (programs in this work), used in previous evolutionary algorithms, have clearly defined components that allow a Case-Base to be simply generated. In this research an evolutionary algorithm is the only general method for producing efficient solutions [17].

This approach raises several questions:

1. What knowledge exists within the evolved programs that may be of use?
2. How can this knowledge be automatically identified and utilised?
3. How can this knowledge be reused?

To answer these questions the following approach was taken: In human designs small programs are designed, and then linked together to make larger programs. For this reason collections of evolved programs were examined to see if such *principle sub-programs* could be identified. This facilitates understanding of how the evolved programs work. These principles may consist of small sub-programs that have been extensively used throughout a large number of programs of different functionality, and the methods for assembling them into larger programs. An example of an identified sub-program that is used in a larger program is shown in Figure 12.

Each principle contains knowledge pertaining to a particular sub-program, and collections of principles form the Cases in the Case-Base. Case-Based retrieval is then used to retrieve appropriate principles based on specified requirements. Suitable adaptation techniques such as those developed by Hanney [4], Giraud-Carrier [3] could be applied to build larger and more complex programs that are too computationally expensive to be evolved. Since the required functionality of programs can be specified as a truth table, the sub-programs that make up the complete design obtained by Case-Based Reasoning can be tested automatically.

Repair of faulty solutions can be achieved by Case-Based substitution. The parts of the solution program where error(s) have occurred are identified and replaced

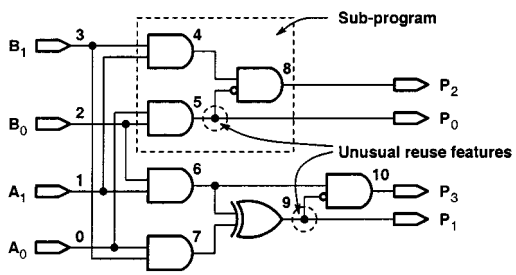

Figure 12. A novel sub-program and two unusual features of reuse in the evolved two-bit multiplier. The labels from 0 to 10 refer to the connection points in the corresponding BCG program.

with error-free substitutes from other Cases that do not display the errors. In a similar manner CBR may be used to optimise the evolved programs produced for specific purposes, e.g. routing, speed, size.

4.3. Automatic creation of a case-base for reuse

In this research, BCG programs were evolved from randomised starting populations and then processed to create a Case-Base. It should be noted that these experiments do not examine the evolutionary technique itself, but only use the data produced by the technique. The processing is achieved in four steps.

The first step involves the removal of redundant information and duplicate programs left over from the evolutionary process. This is followed by compression and normalisation of the remaining programs to facilitate the CBR functions of matching, retrieval and adaptation. Compression involves removal of the spaces left in the genotype after redundant information has been removed. Normalisation reorders differently ordered cells with the same function into a standard form. These steps are taken to simplify comparison of genotypes.

The second step involves splitting the normalised programs into sub-programs and calculation of their structure, behaviour and functionality. The behaviour of a BCG program is represented by the binary outputs of every cell for the given function. In Table 2 the structure (genotype) can be seen for the example in Figure 12, and the function and behaviour of this circuit can be seen in Table 3.

The third step is the separation of the sub-programs into perfect and imperfect solution elements for the given requirements.

The fourth step concerns the indexing of the programs according to their function, behaviour, structure, and sub-programs. The indexing mechanism used is a Case-Based index [12]. Each program is a Case in the Case-Base that stores its own information on its similarity to all other Cases. This information relates to the

Table 2. Structural knowledge in BCG program case

Connection Points	BCG Program—Allele(Gene)			
4	$1_{(0)}$	$3_{(1)}$	†$_{(2)}$	$6_{(3)}$
5	$0_{(4)}$	$2_{(5)}$	†$_{(6)}$	$6_{(7)}$
6	$1_{(8)}$	$2_{(9)}$	†$_{(10)}$	$6_{(11)}$
7	$0_{(12)}$	$3_{(13)}$	†$_{(14)}$	$6_{(15)}$
8	$4_{(16)}$	$5_{(17)}$	†$_{(18)}$	$7_{(19)}$
9	$6_{(20)}$	$7_{(21)}$	†$_{(22)}$	$10_{(23)}$
10	$6_{(24)}$	$9_{(25)}$	†$_{(26)}$	$7_{(27)}$
Outputs	$5_{(28)}$	$9_{(29)}$	$8_{(30)}$	$10_{(31)}$

The column "Connection Points" refers to the connection points in the phenotype (Figure 12). "BCG Program—Allele(Gene)" represents the genotype where the number of each gene is given in parentheses. † The cells' third input is unused, as all of the gates in this example have only two inputs.

Table 3. Functional and behavioural knowledge in program case (in binary)

Function Inputs	Behaviour	Function Outputs	Input meaning (Decimal)	Output (Decimal)
$0_0 0_1 0_2 0_3$	$0_4 0_5 0_6 0_7 0_8 0_9 0_{10}$	$0_{10} 0_8 0_9 0_5$	0×0	0
...	...	...	...	...
$1_0 1_1 1_2 1_3$	$1_4 1_5 1_6 1_7 0_8 0_9 1_{10}$	$1_{10} 0_8 0_9 1_5$	3×3	9

The subscripts in the first three columns refer to the inputs and the cells in a BCG program.

following four indexes. The functional index gives the number of incorrect outputs and the function type. The structural index gives the frequencies of common gates. The behavioural index gives frequencies of common behaviours and the sub-programs index is a list of the sub-programs the program is made up of. Matching of Cases is achieved using the Nearest Neighbour Matching function that gives the ranking of Cases [12]. In this way the index needs only be calculated once, and additional Cases can be indexed in linear time proportional to the number of Cases in the base.

The aim of the experiments was to examine the evolved programs and show that these share some common sub-programs that can be used to build larger programs. Two different sub-programs were identified: those that are directly connected to inputs, and those for which this is not the case. The frequencies of occurrences of these two types can differ significantly. It is necessary to examine very specific types of sub-programs to avoid the combinatorial explosion of enumerating all possible sub-programs.

4.4. Experiments to identify reusable sub-programs

In these experiments two types of arithmetic multipliers were examined: the "2.5"-bit multiplier (2 bits by 3 bits) and the three-bit multiplier. Trying to extract principles by studying two and three-bit multipliers is a very difficult problem because the three-bit multiplier is considerably more complicated. The "2.5"-bit multiplier provides an useful bridge between these two circuits and thus may assist attempts to find scalable principles of construction. Experiments were carried out to produce 50 perfect solutions for both the 2.5 and three-bit multipliers. The maximum number of cells allowed was equal to the number required in the most efficient conventional designs. In the case of the "2.5"-bit multiplier 17 two-input gates are needed, while the three-bit multiplier requires 30. Two different sets of gates were used. The experimental setup was as follows:

Three-bit multiplier population size 5, mutation—3 genes on average, gates {6, 7, 10}, geometry 1×30, levels-back 30.

Three-bit multiplier population size 5, mutation—3 genes on average, gates 6–15, geometry 1×30, levels-back 30.

"**2.5**"**-bit multiplier** population size 5, mutation—3 genes on average, gates 6–15, geometry 1 × 15, levels-back 15.

MUX gates were not allowed in these experiments as they do not generally occur in the conventional circuits and also they make the comparison to the conventional circuits much more difficult.

All of the programs examined were processed to make a basic Case-Base as outlined in Section 4.3. These experiments firstly involved an examination of sub-programs that are directly connected to inputs and secondly the examination of those that are not. Figure 13 shows the connections between the gates in the 2-into-1 sub-programs. One identified sub-program is shown in Figure 14 (2-into-1 example). Sub-programs like the "sibling example", shown in this figure are the subject of future work.

The results of the experiment showed that the input sub-programs (the inputs are connected to the primary program inputs) closely followed the human design. In the cases studied the products of pairs of inputs are calculated. These products can be seen in the examples of the two-bit multiplier for the evolutionary design and the human design shown in Figures 12 and 14, respectively.

The evolutionary designs are markedly different from the conventional designs in the way that they reuse parts of the circuit. Miller *et al*. [17] noted the unusual reuse of a product of two low significant bits directly in the output of a high significance product. These unusual features are shown in Figure 12. It can be seen that the output $\mathbf{P_1}$ is reused to generate $\mathbf{P_3}$, where as in Figure 14 (the conventional human design) $\mathbf{P_1}$ is not used in any other part of the circuit.

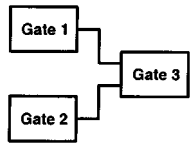

Figure 13. The 2-into-1 sub-program layout.

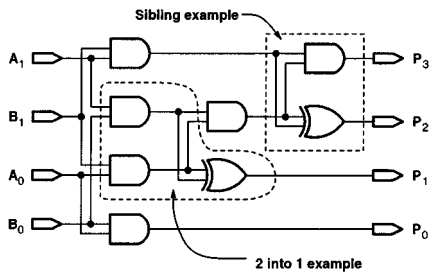

Figure 14. Two examples of the sub-program formats shown here in the conventional two-bit multiplier (Figure 18a).

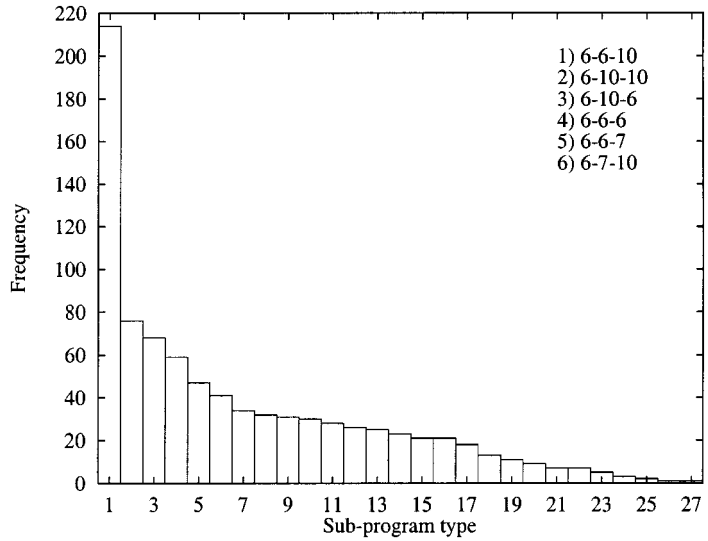

Figure 15. The frequencies of 2-into-1 sub-programs for 50 three-bit multiplier circuits with expert recommendations for available gate functions 6, 7, and 10 (Table 1). The six most frequent sub-programs are listed.

Figure 15 shows the frequencies of 2-into-1 sub-programs for 50 three-bit multiplier circuits, with expert recommendations for available gate functions 6, 7, and 10 that were intended to promote elegant solutions. The recommendations were based on prior experience of the evolved solution for the two-bit multiplier (Figure 12). The first six bars represent sub-programs: `6-6-10`, `6-10-10`, `6-10-6`, `6-6-6`, `6-6-7`, and `6-7-10`. The first bar in Figure 15 is the frequency of sub-program "2-into-1 example" in Figure 14 and it is used significantly more than the other sub-programs. The fifth bar in Figure 15 is the frequency of the sub-program shown in Figure 12. The sub-programs represented by the first four bars in Figure 15 are common in conventional designs; however the sub-program shown in Figure 12 is novel and is not used in the conventional human design. It is clear that much of the evolutionary two-bit multiplier can be reused to build a three-bit multiplier. This implies that the larger solutions tend to contain the same sub-programs as the smaller solutions.

Figure 16 shows the frequencies of the 2-into-1 sub-programs for 50 three-bit multiplier circuits using gates 6 to 15. In this experiment no assumptions were made about suitable gate functions. It was hoped that the experiment would reveal the fundamental building blocks of the multiplier circuits. There were 309 different 2-into-1 sub-program types. The six most frequent sub-programs were `6-6-10`, `6-15-11`, `15-15-10`, `6-6-7`, `15-6-11`, and `15-6-13`. It can be seen that the dominant sub-program is once again `6-6-10`. This confirms that the conventional structures are most common. It is noteworthy that the gate function 15 occurs very often. This was unexpected as the use of a `NAND` gate is not familiar in conventional multiplier design.

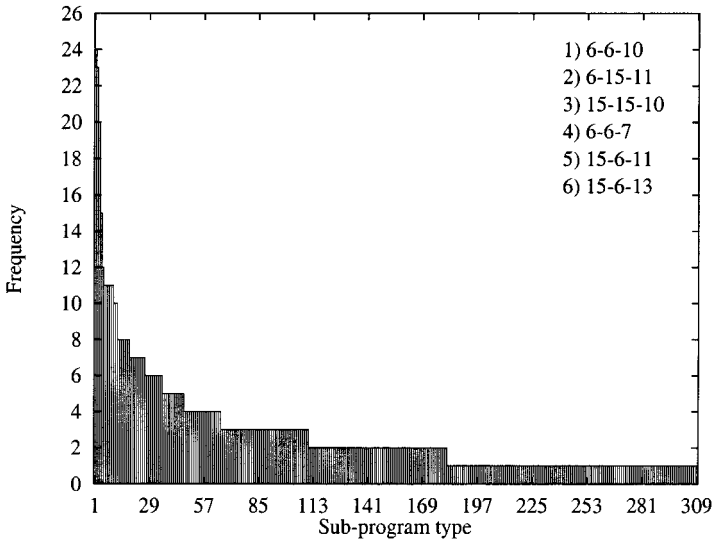

Figure 16. The frequencies of 2-into-1 sub-programs for 50 three-bit multiplier circuits without expert recommendations for gate functions 6 to 15. The six most frequent sub-programs are listed.

Figure 17 shows the frequencies of the 2-into-1 sub-programs for 50 "2.5"-bit multiplier circuits using gates 6 to 15. There were 124 sub-programs identified. The first six were `6-6-10`, `6-15-7`, `6-6-7`, `6-15-11`, `15-15-10`, and `6-6-6`. It should be noted that taken in isolation some of the sub-programs are equivalent. For instance, `6-6-10` and `15-15-10` could be considered to be logically identical however it may be that some of the internal connections are reused in another part of the circuit. Once again, the figure shows that `6-6-10` is dominant. At this stage it is not known which sub-programs are responsible for the efficiency of these designs. For instance, the sub-program `6-6-7` is here the third most frequently occuring and the fifth and the fourth in the previous two experiments, respectively.

It has been seen that all multipliers largely use the same type of sub-programs. This shows that they could potentially be reused by a CBR system to design multiplier programs with more than three-bit multiplication.

4.5. Discussion and future work

This work has shown that it is possible to automatically extract and apply principles of design in the complex domain of BCG programming. These principles contain knowledge pertaining to structured sub-programs, and give an understanding of the evolved programs. Previous work in this area has been in domains where the solutions produced by evolutionary algorithms had definite components that can be easily made into a Case-Base [14].

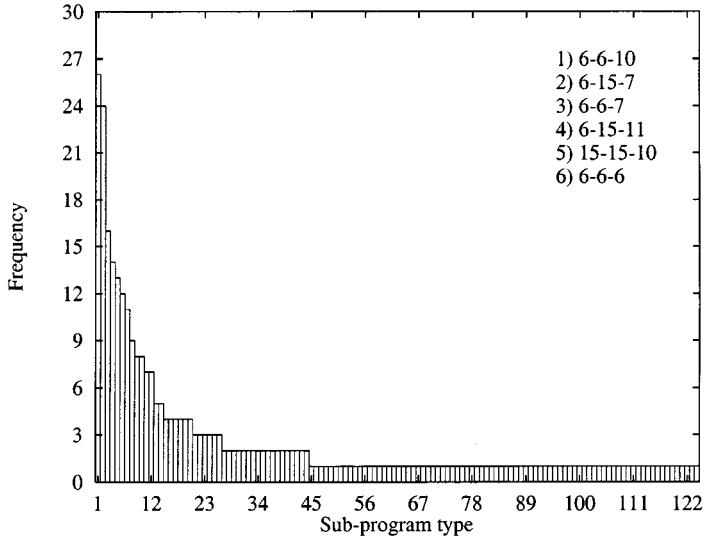

Figure 17. The frequencies of 2-into-1 sub-programs for 50 "2.5"-bit multiplier circuits without expert recommendations for gate functions 6 to 15. The six most frequent sub-programs are listed.

The work presented here addresses the difficult task of identifying components for Case building in a domain where these components are not obvious. The techniques developed are seen to produce digital circuits in the form of gate array programs that may be more efficient than their equivalent human design. They also show how to learn principles involved in evolved digital circuits.

Further analysis will involve examining different types of sub-program format from that shown in Figure 13, for example the "sibling example" in Figure 14. Interpretation becomes more difficult using a larger number of function choices, and it has been shown that the CBR system can automatically identify sub-programs that were known to human designers, and identify novel sub-programs. This allows the CBR system to automatically suggest limited function types for the algorithms used in evolving new programs to improve performance. It may also be possible to use the CBR system to seed the evolutionary algorithm, a technique proved in [14]. Their approach uses evolution to adapt an existing solution to a new problem.

In the experiments reported here the maximum number of cells chosen was equal to the number required to build the conventional circuit. It is already known that the larger the maximum number of gates allowed to construct the circuit the easier it becomes to evolve [18]. This could imply that in these experiments the more conventional sub-programs are likely to dominate. Further experiments where the maximum number of cells is less than the conventional should be revealing.

The research work being carried out is still in its early stages. Current results obtained are based on simple gate array circuit programs. Future work entails

using the above techniques in the creation of programs that are too large to evolve.

From the point of software reuse, this work shows that it is possible to achieve a high level of automation of software reuse in BCG programming where precise requirements can be specified, and behaviour can be completely analysed. Future work will examine the portability of the proposed approach to other software engineering problems.

Experiments of a similar nature to those described in this section have been carried out on two-bit adders. The analysis showed once again that there were some sub-programs that were much more frequent than the majority of the sub-programs. In these experiments recommendations for atomic components were made to promote elegant solutions. These recommendations were compared to those suggested by the frequency analysis of the sub-programs and to those suggested by the frequencies of the atomic gates. The experiments showed that the expert recommendations gave rise to high numbers of duplicate solutions in the sets. Further to this the sub-program recommended atomic gate selections gave rise to fewer duplicate solutions and a higher average fitness in each set. The average number of generations required to produce a solution remained close to that of the expert recommendation based set. The expert based results and the Case-Base based results were compared to results based on the frequencies of the atomic gates. The recommendations suggested by the frequencies of the atomic gates gave much lower numbers of perfect solutions, larger numbers of duplicate solutions and a lower average fitness. This suggests that the solutions with the highest fitnesses tend to be modular in nature. It is expected that larger problems in the same class will show greater differences between the techniques. It is also expected that these results may not hold for all problem classes, as some problem classes are not currently known to have modular solutions.

5. Conclusions

In Part I of this work it was shown how an evolutionary algorithm together with a process of assembling and testing could be used to produce novel and efficient designs for digital arithmetic circuits [17]. One of the central ideas was to look at whether it might be possible to extract new principles which would allow the construction of efficient multiplier circuits of arbitrary size by studying evolved examples of two and three-bit multipliers. In one sense this problem is ideal for artificial evolution. Firstly, this is because the evaluation process for a genotype representing a circuit is extremely fast as it relies on precisely those simple bit-level operations that modern CPUs were designed for. Secondly, the binary nature of the evolved circuits makes them relatively simple to understand. There are two ways in which it might be possible through artificial evolution to try to build efficient large systems. One is to try to discover a general scalable principle of design. The second is to try to produce as efficient and large building blocks as possible. In order to improve the chance of success in both these aims this paper has looked at two aspects: first, the nature of the fitness landscapes associated with

these digital circuits, and second, to try to develop automated ways of extracting sub-principles in evolved circuits.

These two aspects are complementary in their approach: the first looks at the area around the genotypes, on the fitness landscape, and the second looks at the internal workings of the phenotypes themselves. Examination of the relationship between these two aspects is the subject of future work. It is hoped that this will show where evolutionary techniques should stop being used and CBR techniques should be applied.

The structure of fitness landscapes has been studied in terms of their smoothness, ruggedness and neutrality. This has been done using an information analysis [32, 33] on a time series that is obtained by sampling the fitness values on a random walk. A major impediment in studying the structure of circuit evolution landscapes is that they originate from two completely different alphabets responsible for the gate functionality and the connectivity of the evolved digital circuits. Thus, it has been found that it might be better to consider these landscapes as a product of three subspaces associated with the gate functionality, internal and output connectivity of the gate array. Hence, the genotypes sub-divide into three chromosomes with different characteristics. It has been shown that the landscapes are vastly neutral with sharply differentiated plateaus and these in turn are related to the scale the objective function. The landscapes were found to become more continuous with increasing scale.

Even with a computer that could deliver large numbers of correct designs an expert would have the problem of examining all of the evolved circuits to extract principles. It has been shown that it is possible to automatically identify and apply the evolutionary design principles contained within the phenotypes. This greatly reduces the knowledge acquisition bottleneck, a primary factor in the creation of a Case-Base in any CBR system [4]. In previous work where evolutionary algorithms were used in conjunction with CBR the genotypes have had clearly identified modules [14], making the construction of the Case-Base simple. It was shown here that by examining the frequency of occurrence of small sub-circuits (2-into-1) that a sort of circuit "fingerprint" can be constructed. This not only confirms the familiar conventional principles but reveals novel sub-circuits which are good building blocks in the evolved circuits. It was shown that the principles identified in small scale multipliers (two-bit) match those identified in two larger scale multipliers, the two-and-a-half-bit multiplier and the three-bit multiplier. This suggests that there are principles in these multiplier circuits that may hold true for all sizes of multipliers and as such may be used to create larger scale multipliers that are beyond the reach of current evolutionary processing power.

Another important factor is that modular construction bypasses the necessity for testing of the truth table for the whole circuit, only the module need be tested. For example the ripple carry adder principle tells us that any number of one bit carry adder units may be chained together to produce a perfectly functional larger carry adder, like the conventional sixteen-bit carry adder. This reduces the problem of exhaustive testing of very large circuits. This identification of principles facilitates experts in understanding the nature of the evolutionary designed solutions. These processes enable the creation of a Case-Base, the foundation for a reasoning

system, that could be used to solve the scaling problem, and leads the way to the automated reasoning techniques of CBR.

There is still a long way to go in this field. Work continues via information analysis of digital fitness landscapes to produce a refined search method. It is also of interest to investigate why the evolutionary algorithm chooses particular principle modules. Is this because these modules are preferred components in the final working circuits or because these modules happen to be easier to evolve, or perhaps *both*? This might be found by examining the structure of the subspaces associated with the identified modules. Attempts are underway to define *behavioural* fingerprints of sub-circuits which should eliminate the problem of many genotypes producing the same function. New test problems are being defined which will potentially allow simpler generalisable principles to be extracted. Further developments will be reported in due course.

Acknowledgments

Dominic Job would like to thank British Telecommunications PLC for supporting this research work.

Appendix A: Some conventional and evolved digital circuits

The appendix gives the schematics of some conventional and evolved digital circuits which were seen in Part I [17] and are involved in the discussion of this paper. The circuits depicted in Figure 18 are the conventional and evolved two-bit multipliers.

The most efficient three-bit multiplier circuits that were found so far are shown in Figures 19 and 20. Figure 21 shows two representations of the four-bit parity function evolved using different sets of gates.

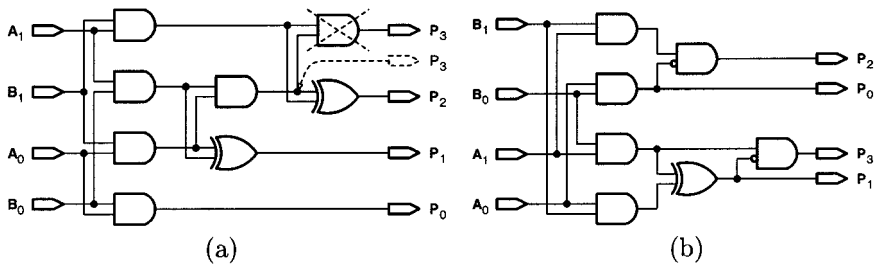

Figure 18. Most efficient (a) conventional and (b) evolved two-bit multipliers.

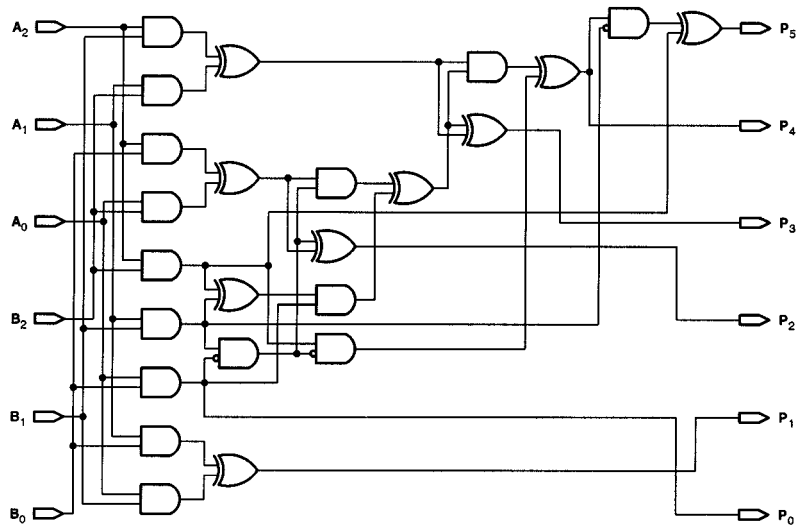

Figure 19. Evolved three-bit multiplier (24 two-input gates).

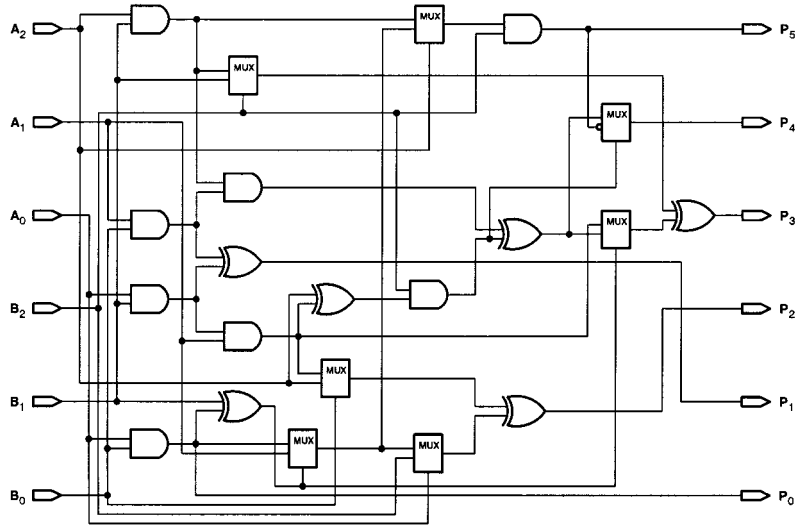

Figure 20. Evolved three-bit multiplier (21 gates = 14 two-input gates + 7 MUX).

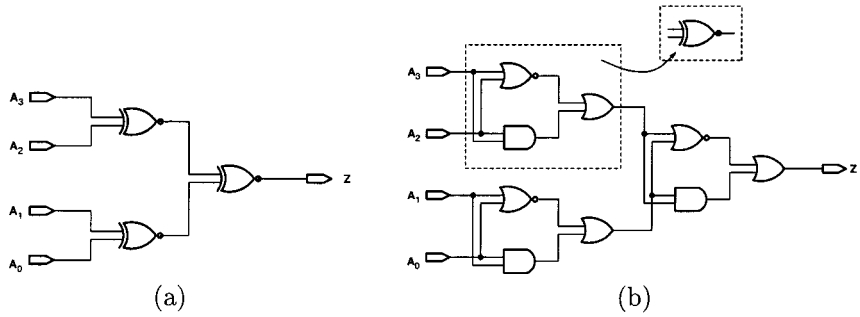

Figure 21. Two representations of the four-bit parity function with (a) gate XNOR and (b) gates AND, OR, and NOR.

References

1. T. Bäck, F. Hoffmeister and H. P. Schwefel, "A survey of evolutionary strategies," in Proc. 4th Int. Conf. on Genetic Algorithms, R. Belew and L. Booker (eds.), Morgan Kaufmann: San Francisco, CA, 1991, pp. 2–9.
2. G. Box and G. Jenkins, Time Series Analysis, Forecasting and Control. Holden Day, 1970.
3. C. Giraud-Carrier, "Flare: Induction with prior knowledge," in Proc. Expert Systems 1996, vol. XIII of Research and Development in Expert Systems, SGES Publications, 1996, pp. 173–181.
4. K. Hanney, "Learning adaptation rules from cases," Technical report, Department of Computer Science, Trinity College, University of Dublin, Ireland, M.Sc. thesis, 1996.
5. W. Hordijk, "A measure of landscapes," Evolutionary Computation vol. 4 (4) pp. 335–360, 1996.
6. W. Hordijk, "Correlation analysis of the synchronising-ca landscape," Phys. D vol. 107 pp. 255–264, 1997.
7. W. Hordijk and P. F. Stadler, "Amplitude spectra of fitness landscapes," Adv. Complex Systems vol. 1 pp. 39–66, 1998.
8. J. Hunt, "Evolutionary case base design," in Progress in Case-Based Reasoning First UK workshop, Springer-Verlag: Berlin, 1995.
9. D. Job, V. Shankararaman and J. Miller, "Hybrid ai techniques for software design," in Proc. 1999 Conf. on Software Engineering & Knowledge Engineering, 1999, pp. 315–319.
10. T. Jones, "Evolutionary algorithms, fitness landscapes and search," Ph.D. thesis, University of New Mexico, Albuquerque, NM, 1995.
11. S. Kauffman, "Adaptation on rugged fitness landscapes," in Lectures in the Sciences of Complexity, SFI Studies in the Sciences of Complexity, D. Stein (ed.), Addison-Wesley: Reading, MA, 1989, pp. 527–618.
12. J. Kolodner, Case-Based Reasoning, Morgan Kaufmann: San Mateo, CA, 1993.
13. P. Maguire, V. Shankararaman, R. Szegfue and L. Morss, "Application of case-based reasoning to software reuse," in Progress in Case-Based Reasoning, Lecture Notes in Artificial Intelligence, I. Watson (ed.), Springer-Verlag: Berlin, 1995, pp. 165–174.
14. M. L. Maher and A. G. de Silva Garza, "The adaptation of structural systems designs using genetic algorithms," in Information Processing in Civil and Structural Engineering Design, CIVIL-COMP Press, 1996, pp. 189–196.
15. B. Manderick, M. de Weger and P. Spiessens, "The genetic algorithm and the structure of the fitness landscape," in Proc. 4th Int. Conf. on Genetic Algorithms, R. K. Belew and L. B. Booker (eds.), Morgan Kaufmann: San Mateo, CA, 1991, pp. 143–150.

16. J. F. Miller, "An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach," in Proc. 1st Genetic and Evolutionary Computation Conf., W. Banzhaf, J. Daida, A. E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.), San Francisco, CA: Morgan Kaufmann, 1999, vol. 2, pp. 927–936.
17. J. F. Miller, D. Job and V. K. Vassilev, "Principles in the evolutionary design of digital circuits—part I," J. Genetic Programming and Evolvable Machines vol. 1 (1) pp. 7–35, 2000.
18. J. F. Miller and P. Thomson, "Aspects of digital evolution: Evolvability and architecture," in Parallel Problem Solving from Nature V, vol. 1498 of Lecture Notes in Computer Science, A. E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel (eds.), Springer: Berlin, 1998, pp. 927–936.
19. J. F. Miller, P. Thomson and T. Fogarty, "Designing electronic circuits using evolutionary algorithms, arithmetic circuits: A case study," in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, D. Quagliarella, J. Periaux, C. Poloni and G. Winter (eds.), Wiley: Chichester, UK, 1997 pp. 105–131.
20. M. Mitchell, S. Forrest and J. Holland, "The royal road for genetic algorithms: Fitness landscapes and ga performance," in Proc. 1st Eur. Conf. on Artificial Life, J. Varela and P. Bourgine (eds.), MIT Press: Cambridge, MA, 1991, pp. 245–254.
21. H. Mühlenbein and D. Schlierkamp-Voosen, "The science of breeding and its application to the breeder genetic algorithm (bga)," Evolutionary Computation vol. 1 (4), pp. 335–360, 1993.
22. C. M. Reidys and P. F. Stadler, "Neutrality in fitness landscapes," Technical Report, 98-10-089, Santa Fe Institute, 1998, Submitted to Appl. Math. Comput.
23. H.-P. Schwefel, Numerical Optimization of Computer Models, John Wiley & Sons: Chichester, UK, 1981.
24. M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," IEEE Trans. Evolutionary Computation vol. 1 (1) pp. 83–97, 1997.
25. B. Smyth, "Case based design," Ph.D. thesis, Department of Computer Science, Trinity College, University of Dublin, Ireland, 1996.
26. P. F. Stadler, "Towards theory of landscapes," in Complex Systems and Binary Networks, R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck and F. Zertuche (eds.), Springer-Verlag: Berlin, 1995, pp. 77–163.
27. P. F. Stadler, "Landscapes and their correlation functions," J. Math. Chem. vol. 20 pp. 1–45, 1996.
28. P. F. Stadler and W. Grünter, "Anisotropy in fitness landscapes," J. Theor. Biol. vol. 165 pp. 373–388, 1993.
29. P. F. Stadler and G. P. Wagner, "Algebraic theory of recombination spaces," Evolutionary Computation vol. 5 (3) pp. 241–275, 1997.
30. A. Thompson, P. Layzell and R. S. Zebulum, "Explorations in design space: Unconventional electronics design through artificial evolution," IEEE Trans. Evolutionary Computation vol. 3 (3) pp. 167–196, 1999.
31. V. K. Vassilev, "Information analysis of fitness landscapes," in Proc. 4th Eur. Conf. Artificial Life, P. Husbands and I. Harvey (eds.), MIT Press: Cambridge, MA, 1997a, pp. 116–124.
32. V. K. Vassilev, "An information measure of landscapes," in Proc. 7th Int. Conf. on Genetic Algorithms, T. Bäck (ed.), Morgan Kaufmann: San Francisco, CA, 1997b, pp. 49–56.
33. V. K. Vassilev, T. C. Fogarty and J. F. Miller, "Information characteristics and the structure of landscapes," Evolutionary Computation vol. 8 (1) pp. 31–60, 2000.
34. V. K. Vassilev, J. F. Miller and T. C. Fogarty, "Digital circuit evolution and fitness landscapes," in Proc. Congress on Evolutionary Computation, IEEE Press: Piscataway, NJ, 1999a, vol. 2, pp. 1299–1306.
35. V. K. Vassilev, J. F. Miller and T. C. Fogarty, "On the nature of two-bit multiplier landscapes," in Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware, A. Stoica, D. Keymeulen and J. Lohn (eds.), IEEE Computer Society: Los Alamitos, CA, 1999b, pp. 36–45.
36. E. D. Weinberger, "Correlated and uncorrelated fitness landscapes and how to tell the difference," Biol. Cybernetics vol. 63 pp. 325–336, 1990.
37. S. Wright, "The roles of mutation, inbreeding, crossbreeding and selection in evolution," in Proc. 6th Int. Conf. on Genetics, D. F. Jones (ed.), 1932, vol. 1, pp. 356–366.