Bidirectional Incremental Evolution in Extrinsic Evolvable Hardware

Tatiana Kalganova
School of Computing, Napier University
219 Colinton Road, Edinburgh, EH14 1DJ, UK
t.kalganova@dcs.napier.ac.uk

Abstract

Evolvable Hardware (EHW) has been proposed as a new technique to design complex systems. Often, complex systems turn out to be very difficult to evolve. The problem is that a general strategy is too difficult for the evolution process to discover directly. This paper proposes a new approach that performs incremental evolution in two directions: from complex system to sub-systems and from sub-systems back to complex system. In this approach, incremental evolution gradually decomposes a complex problem into some sub-tasks. In a second step, we gradually make the tasks more challenging and general. Our approach automatically discovers the sub-tasks, their sequence as well as circuit layout dimensions. Our method is tested in a digital circuit domain and compared to direct evolution. We show that our bidirectional incremental approach can handle more complex, harder tasks and evolve them more effectively, then direct evolution.

1. Introduction

Evolvable Hardware (EHW) has been introduced as a target architecture for complex system design based on evolution. However, evolution of a complex system can be a very difficult task. The reasons for this include:

- Direct evolution in inefficient at evolving a long chromosome string. This problem has been solved by introducing the variable length chromosome [1], function-level evolution [2], [3], automatically defined functions [4], divide-and-conquer approach (as a increased complexity evolution) [5], [6].
- The number of generations required to perform a task can increase drastically with the complexity of the task. This is often called a stalling effect. For example, a two-bit multiplier (4 inputs, 4 outputs) can be easily evolved after $5\ 000$ generations [7]. Similarly, evolution of a fully functional three-bit multiplier (6 inputs, 6 outputs) at gate-level requires between $3\ 000\ 000$ [7] and $10\ 000\ 000$ generations [8]. According to this data we can predict that billions of generations are required to evolve a fully functional 4-bit multiplier (8 inputs, 8 outputs). A similar trend can be observed in evolutionary robotics. In order to overcome this problem several researchers in the field of robotics have demonstrated that incremental evolution can be successfully applied to stochastic dynamic problems when implemented using neural networks [9], [10], [11]. In incremental evolution, neural networks learn complex general behaviour by starting with simple behavior and incrementally making the task more general.

We observe that while methods like the variable length chromosome [1], etc. are able too handle long chromosome strings, they fail to address the stalling effect of evolution. Although, neural network approaches address stalling effect issues, the definition of their heuristics have been a deterrent to its application in EHW.

Our research shows that both of the problems mentioned above can be solved using bidirectional incremental evolution. Our proposed approach significantly differs from the incremental evolution approaches proposed in the past. It incorporates two main approaches: Divide-and-Conquer and Incremental evolution. The principle of our approach is to *divide* a complex task into simpler sub-tasks, to *evolve* each of these sub-tasks and then *merge* incrementally the evolved sub-systems, reassembling a new evolved complex system. The evaluation tasks and their circuit layout dimensions can be identified using either standard decomposition methods or EHW-oriented decomposition. In our method first the evolution defines the partitioning vectors of any given complex system. These vectors are used to decompose the system into a number of sub-systems. Each of sub-systems is then evolved separately. If a sub-systems is not sufficiently partitioned, new partitioning vectors are defined and the subsystem is decomposed further. This may continue until a final evolvable sub-system set is defined. Once all sub-systems have been evolved, they are merged incrementally into one cost-optimised system.

2. An Extrinsic EHW

Bidirectional incremental evolution can be performed using any extrinsic or intrinsic EHW. In this work, the extrinsic EHW approach first proposed by J. Miller [12], is used to evolve sub-tasks. The EHW technique is based on evolving the functionality and connectivity of a rectangular array of logic cells whose dimensions are defined by the circuit layout. Each logic cell in this array is uncommitted and can be removed from the network if it proves to be redundant. The inputs to any cell in the combinational network may be logical constants and primary inputs as well as the outputs of logic cells which are in columns to the left of the cell in question. Each logic function describing the behaviour of logic cell is specified in advance. The circuit layout is evolved together with circuit functionality using a rudimentary $(1 + \lambda)$ evolutionary strategy with uniform mutation [13], [14]. Evaluation of the circuit is carried out using dynamic fitness function that allows us to evolve fully functional circuits as well as reduce their cost in terms of the number of logic gates used [13], [3].

3. Decomposition of Logic Function

In order to define the partitioning vectors in system, some knowledge about its decomposability has to be applied. In this work we will consider both Shannon's and output decompositions. *Decomposition* means breaking a large logic block into several relatively smaller ones.

According to the Shannon's expansion theorem the logic function can be decomposed as follows. Let $f(x_1, x_2, ..., x_n)$ be any Boolean expression of n variables. Then Shannon's theorem in the SOP (sum-of-products) form says that

$$f(x_1, x_2, ..., x_n) = f(1, x_2, ..., x_n) \cdot x_1 + f(0, x_2, ..., x_n) \cdot \overline{x_1}. \tag{1}$$

The *decomposition by outputs* is a technique to break a logic function with many outputs into several functions with fewer outputs. The functions with fewer outputs can be designed independently.

4. Bidirectional Incremental Evolution

The essential idea of incremental evolution if to *scale* the evaluation function (i.e., the "fitness function" against which, say, a complex circuit is evolved) over time, with the aim of minimizing the overall time spent evolving a circuit that achieves the prescribed task. Incremental evolution consists of gradually evolving one complex task in a series of tasks of increasing complexity [15], [16].

Bidirectional incremental evolution consists in gradually decomposing a complex task into a series of tasks of decreasing complexity and in gradually evolving a complex task into a series of tasks of increasing complexity. In bidirectional incremental evolution, the scaling of the fitness function is performed during evolution. Decomposition of the system can be undertaken using standard decomposition methods or specially designed EHW-oriented decomposition. The decomposed sub-functions are evolved separately if they achieve an adequate level of complexity or can be partitioned into smaller sub-functions. Once, the first stage is completed, evolved fully functional sub-systems are gathered into larger complexity sub-systems and evolution is carried out on these sub-systems. During this stage, evolution is applied in order to decrease the total cost of the evolved complex system.

The main advantage of the method is that evolution is not carried out in one operation on the complete evolvable hardware unit, but rather is a top-down and a up-bottom ways. The number of inputs and outputs in an evolved subsystem can be limited to allow faster evolution. The two main stages of evolution are discussed in detail in the following sub-sections.

4.1. Stage 1. Evolution toward a modularised system

During this stage the evolution is undertaken in order to decompose problem into smaller problems. It can be performed using (1) standard functional decomposition methods, or (2) EHW-oriented decomposition.

Standard decomposition methods Any of methods from the decomposition theory can be applied in order to divide a complex system into sub-systems. Interactive decomposition, cascaded decomposition, joint decomposition, etc. can be applied to the complex system if the problem of combinational circuit design is considered. There is one disadvantage of using these methods in current approach: they are designed for target implementation technology or architectural configurations and they do not adapt to specific features of evolution. This disadvantage can be removed if EHW-oriented decomposition is applied.

EHW-oriented decomposition EHW-oriented decomposition discovers the evaluation tasks as well as their sequence automatically. The main idea of this approach is to define the *easily evolved* sub-functions and evolve them separately. Note that the fully functional or nearly fully functional design has to be obtained for each decomposed sub-function.

The truth table of an n-input m-output logic function given on p input combinations contains an input matrix $I(n \times p)$ and an output matrix $O(m \times p)$ and can be described by this pair of matrices as $(I(n \times p), O(m \times p))$. The logic function is completely specified if $p = 2^n$, i.e. it is given on *all* input combinations. For instance, the completely specified 6-input 7-output logic function can be described by truth table $(I(6,64), O(7,64))$, where $n = 6$, $m = 7$ and $p = 64$.

The following metrics can be used in order to define the quality of an evolved circuit.

The percentage of correct output bits corresponding to the j-th output, $\mathbf{y_j}$ is calculated as follows:

$$f_{\mathbf{y_j}^o} = \frac{\sum_{i=1}^{p} |y_j - d_j|}{p} * 100; \qquad (2)$$

where $|y_i - d_i|$ is the absolute difference between the actual output y_i and the desired output d_i; $\mathbf{y_j}$ is the vector of the j-th circuit output. If $f_{\mathbf{y_j}^o} = 100.0$ the circuit implements the output y_j completely.

The percentages of correct output bits in the j-th output for input combinations with $x_i = 0$ and $x_i = 1$ are computed as follows:

$$f_{\mathbf{y_j}^o}|_{x_i=0} = \frac{\sum_{i=1}^{p_{x_i=0}} |y_j - d_j|}{p_{x_i=0}} * 100; \qquad (3)$$

$$f_{\mathbf{y_j}^o}|_{x_i=1} = \frac{\sum_{i=1}^{p-p_{x_i=0}} |y_j - d_j|}{p - p_{x_i=0}} * 100; \qquad (4)$$

where $p_{x_i=0}$ is the number of input combinations in the truth table with $x_i = 0$. If $f_{\mathbf{y_j}^o}|_{x_i=0} = 100.0$, the circuit completely implements the truth table generated from the output y_j with condition that $x_i = 0$.

The EHW-oriented decomposition is performed to evolve an n-input m-output logic function given on p input combinations as follows:

1. Define the termination condition of evolution, for example, the number of generations N_{gen}.

2. Evolve the initial complex system implementing the truth table $T_0 = (I(n \times p), O(m \times p))$ during N_{gen} (gate- or function-level EHW with randomly generated initial population and with circuit layout evolution).

3. Keep the result of evolution: the genotype of the best chromosome.

4. Calculate $f^o_{\mathbf{y_j}}$, $(j = 1, ..., m)$ and $f_{\mathbf{y_j}^o}|_{x_i=0}$, $f_{\mathbf{y_j}^o}|_{x_i=1}$, $(i = 1, ..., n)$.

5. Choose the output partitioning vector $\mathbf{v_y}$ defined by the circuit outputs with higher $f^o_{\mathbf{y_j}}$;

6. IF p is large, THEN Choose the input partitioning vector $\mathbf{v_i}$ defined by the higher value of $f_{\mathbf{y_j}^o}|_{x_i=0}$ or $f_{\mathbf{y_j}^o}|_{x_i=1}$ for the circuit outputs specified in step 5, generate the product partitioning vector $\mathbf{v_p}$ determined by input combinations with $x_i = 0$ or $x_i = 1$, ELSE Go to step 7.

7. Generate the 3 (if the step 6 has been executed) or 2 (if step 6 has not been executed) truth tables of sub-functions: (1) The easily evolved function is defined by the truth table with $\mathbf{v_o}$ and $\mathbf{v_p}$ partitioning vectors: $T_1 = (I(n \times |\mathbf{v_p}|), O(|\mathbf{v_o}| \times |\mathbf{v_p}|))$; (2) The second sub-system contains the remaining output combinations for the best chosen product partitioning vector $\mathbf{v_p}$: $T_2 = (I(n \times p - |\mathbf{v_p}|), O(|\mathbf{v_o}| \times p - |\mathbf{v_p}|))$; (3) The third sub-system contains the remaining data from the truth table T_0: $T_3 = (I(n \times |\mathbf{v_p}|), O(m - |\mathbf{v_o}| \times |\mathbf{v_p}|))$.

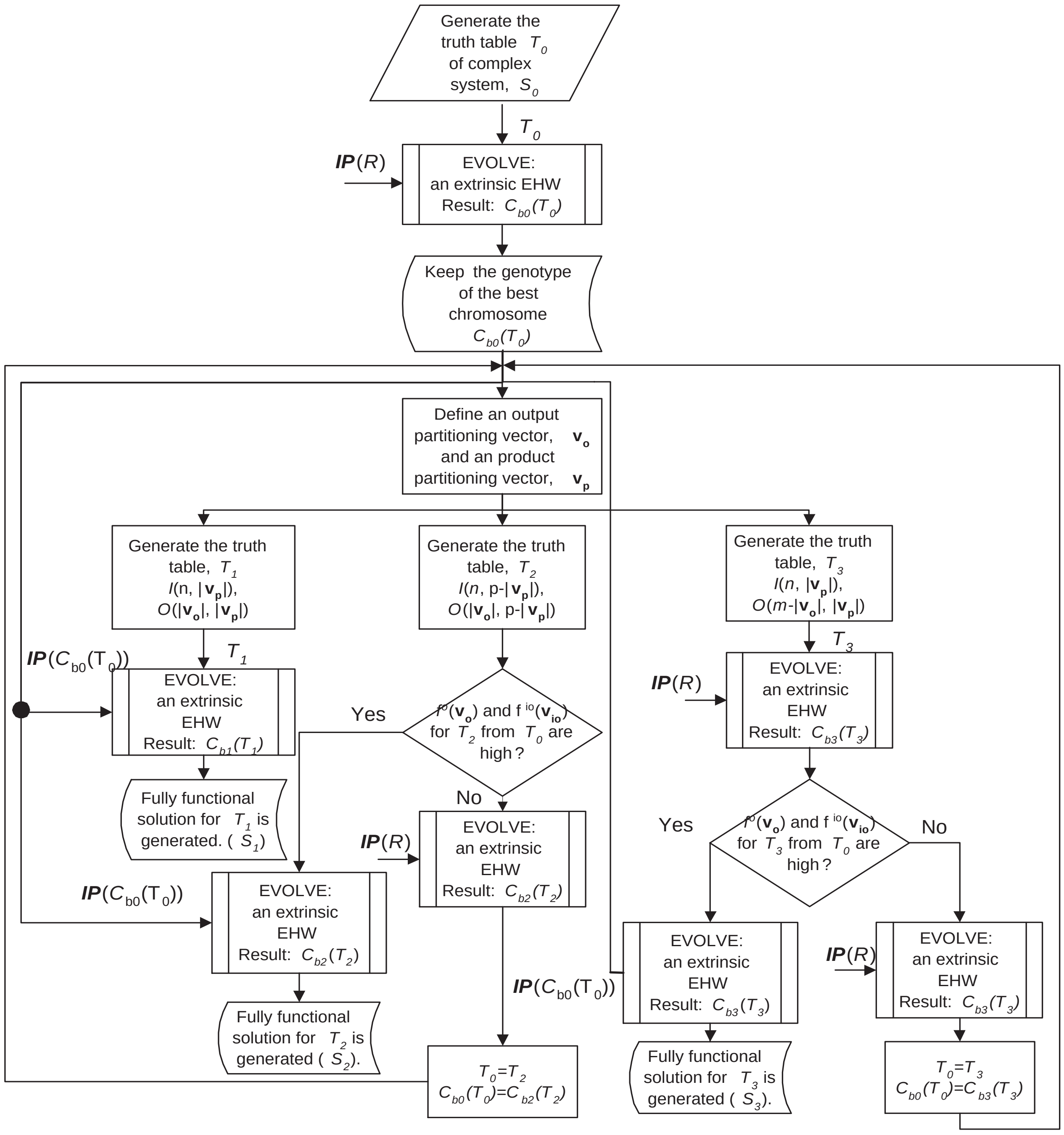
8. Evolve separately the sub-systems defined in the previous step. The initial population of the sub-system described by the truth table T_1 is the final population of EHW executed at step 2. The sub-system T_2 is evolved using either the final population obtained at step 2 or a randomly generated population. The initialisation process depends on the parameters $f^o_{\mathbf{y_j}}$, $f_{\mathbf{y_j}^o}|_{x_i=0}$ and $f_{\mathbf{y_j}^o}|_{x_i=1}$ for this part of the truth table in T_0. The sub-system T_3 is evolved using the randomly generated initial population during N_{gen} generations.

9. IF the complexity level for T_2 and T_3 is not sufficient, THEN Consider the sub-system as a complex system and proceed the steps 3-8 for each sub-system, ELSE Go to step 10.

10. The evolvable sub-systems are generated.

The sub-system $S_1(T_1)$ is generated with the best metrics. If these metrics are equaled to 100, the fully functional circuit can be extracted from the system $S_0(T_0)$. In other case, the initial population is generated based on chromosome genotype specified by $S_0(T_0)$. The fitness function of chromosomes in the initial population is relatively high, since it is defined by chosen metrics. Therefore, the sub-system $S_1(T_1)$ is easily evolvable. In this case the genetic material obtained during previous evolutionary process of complex task is used in evolution of less complex task.

The diagram of EHW-oriented decomposition described above is given in Fig. 1. The block "EVOLVE" contains two inputs: (1) The truth table, T specifying which function is evolved during the evolution process, and (2) Initialisation, defined how the initial population is generated. The output of this block contains the genotype of the best evolved chromosome $\mathbb{C}_{bc}(T)$. This chromosome will be used to generate the initial population. This initial population will further participate in the evolution of easy evolved sub-systems. The initial population of sub-system S_3 is generated randomly, because the truth table T_3 contains the worst values of $f_{\mathbf{y_j}^o}$ and either $f_{\mathbf{y_j}^o}|_{x_i=0}$ either $f_{\mathbf{y_j}^o}|_{x_i=10}$ taken from the truth table T_0. This also brings some diversity in evolution of the complex system and allows the evolution to find a better "start" point for the next decomposed functions.



Generate the truth table T_0 of complex system, S_0
T_0
IP(R)
EVOLVE: an extrinsic EHW Result: $C_{b0}(T_0)$
Keep the genotype of the best chromosome $C_{b0}(T_0)$
Define an output partitioning vector, $\mathbf{v_o}$ and an product partitioning vector, $\mathbf{v_p}$
Generate the truth table, T_1 I(n, |$\mathbf{v_p}$|), O(|$\mathbf{v_o}$|, |$\mathbf{v_p}$|)
Generate the truth table, T_2 I(n, p-|$\mathbf{v_p}$|), O(|$\mathbf{v_o}$|, p-|$\mathbf{v_p}$|)
Generate the truth table, T_3 I(n, |$\mathbf{v_p}$|), O(m-|$\mathbf{v_o}$|, |$\mathbf{v_p}$|)
T_1
IP(C_{b0}(T$_0$))
EVOLVE: an extrinsic EHW Result: $C_{b1}(T_1)$
Yes
$f^o(\mathbf{v_o})$ and $f^{io}(\mathbf{v_{io}})$ for T_2 from T_0 are high?
No
Fully functional solution for T_1 is generated. (S_1)
IP(R)
EVOLVE: an extrinsic EHW Result: $C_{b2}(T_2)$
IP(C_{b0}(T$_0$))
EVOLVE: an extrinsic EHW Result: $C_{b2}(T_2)$
Fully functional solution for T_2 is generated (S_2).
IP(C_{b0}(T$_0$))
T_0=T_2 $C_{b0}(T_0)$=$C_{b2}(T_2)$
T_3
IP(R)
EVOLVE: an extrinsic EHW Result: $C_{b3}(T_3)$
Yes
$f^o(\mathbf{v_o})$ and $f^{io}(\mathbf{v_{io}})$ for T_3 from T_0 are high?
No
EVOLVE: an extrinsic EHW Result: $C_{b3}(T_3)$
IP(R)
EVOLVE: an extrinsic EHW Result: $C_{b3}(T_3)$
Fully functional solution for T_3 is generated (S_3).
T_0=T_3 $C_{b0}(T_0)$=$C_{b3}(T_3)$

Figure 1. The diagram of EHW-oriented decomposition. **IP** is the initial population; **IP**(R) denotes the randomly generated initial population; T_i is the i-th truth table; $C_{bi}(T_i)$ is the best chromosome evolved using an extrinsic EHW for the function given by the truth table T_i; $\mathbf{IP}(C_{bi}(T_i))$ is the initial population generated using the best chromosome genotype obtained after the evolution process for the i-th truth table.

In the scheme of EHW-oriented decomposition the evolution is terminated after a fixed number of generations, N_{gen}. We choose this termination condition because the fitness function is usually improved significantly during the first generations. As an alternative way, the evolution can be terminated once the fitness value $(\mathcal{F}_{start} + \Delta\mathcal{F})$ is achieved, where F_{start} is the best fitness generated at the initial population; ΔF is the value on which the fitness function is expected to be improved.

Another problem that can arise in EHW-oriented decomposition is the definition of circuit layout for each sub-task that can be solved. This can be avoided by allowing to use circuit layout evolution together with circuit functionality proposed first in [13].

4.2. Stage 2. Evolution toward an optimised system

The assembling of the system is based on the specific features of the output and functional decompositions. For example, the output decomposition guarantees that each sub-system is synthesized separately and it is *completely* independent. In the case of functional decomposition, the corresponding outputs generated for various input combinations in different sub-systems have to be connected together using an one-control multiplexer. Analysis of experimental results shows that it is reasonable to assemble the sub-systems decomposed by functional decomposition first and then the sub-systems separated using output decomposition.

5. Experimental Results

In this section we will consider how the bidirectional incremental evolution performs. The 7-input 10-output logic function (z5xp1_d.pla) has been evolved using both direct and bidirectional incremental evolutions. Hence, $n = 7$, $m = 10$ and $p = 128$. This function is taken from standard benchmark library for combinational logic design. The search has been performed using a rudimentary $(1 + \lambda)$ evolutionary strategy with dynamic fitness function, uniform mutation and population of 5 individuals.

In bidirectional incremental evolution, the system is first evolved toward its modularisation. Once the system has been decomposed into sub-systems with sufficient complexity, the system is evolved toward to its optimisation. These two processes are considered in detail in the following subsections for two types of experiments.

5.1. Experiment A.

In this experiment the complex system is divided into sub-systems by means of output decomposition only.

Stage 1. Evolution toward to a modularised system The evolution of sub-systems is performed during no more than 15000 generations.

During this stage of bidirectional incremental evolution, the circuit structure remains the same, but the evaluation parameters are changed. For instance, the chromosome $\mathbb{C}$ is evaluated by truth table T_1. T_1 can be decomposed into smaller truth tables T_2 and T_3. In this case, the chromosome $\mathbb{C}$ can be estimated by truth tables T_2 and T_3. Obviously, in this case the evaluation by one of the tables will show better results.

The bidirectional incremental evolution begins with attempts to evolve the complex system. The evolved circuits are evaluated by a given truth table of 7 inputs and 10 outputs. The evolution is terminated after 5000 generations and the best chromosome defines the first generated sub-system S_0 that is evaluated according to the metrics given in Eq. 2–3. The result of this evaluation is given in Table 2.

Analysing these data we observe that $f^o_{y_7} = f^o_{y_8} = f^o_{y_9} = 100.0$. This means that the circuit S_0 is fully functional if it is evaluated by the truth table T_1 generated from all input combinations of outputs y_7, y_8 and y_9. Therefore, we can define the first sub-system that can be evaluated according to truth table T_1. From this two truth tables T_1 and T_2 can be generated.

Because the circuit S_0 evaluated by the truth table T_1 is fully functional, the evolution of system S_1 can start with an initial population generated from the circuit S_0. In this case, the evolutionary process will tend to reduce the number of active gates in the circuit.

Since $f^o_{y_j}$ for the outputs $y_0, y_1, y_2, y_3, y_4, y_5$ and y_6 are not equal to 100.0, the circuit S_1 evaluated by the truth table T_2 is not fully functional. Therefore, the next sub-systems have to be defined.

As a result of EHW-oriented decomposition the following sub-systems can be defined:

1. S_0: $T_0 = (I(7,128), O(10,128))$; $Y = \{y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9\}$. *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: Random. Best chromosome - $\mathbb{C}_0$. **Conclusion**: Decomposition by outputs: S_1 - easy evolved; S_2.

2. S_1: $T_1 = (I(7,128), O(3,128))$; $Y = \{y_7, y_8, y_9\}$. *Purpose of evolution:* reduce the number of active gates in circuit; *Initialisation* - $\mathbb{C}_0$ (fully functional circuit). Best chromosome - $\mathbb{C}_1$. **Conclusion**: a fully functional circuit with reduced number of active gates is evolved.

3. S_2: $T_2 = (I(7,128), O(7,128))$; $Y = \{y_0, y_1, y_2, y_3, y_4, y_5, y_6\}$. *Purpose of evolution:* evolving fully functional circuit; *Initialisation:* Random. Best chromosome - $\mathbb{C}_2$. **Conclusion**: Decomposition by outputs: S_3 - easy evolved; S_4.

4. S_3: $T_3 = (I(7,128), O(3,128))$; $Y = \{y_0, y_5, y_6\}$. *Purpose of evolution:* reduce the number of active gates in circuit; *Initialisation* - $\mathbb{C}_2$ (fully functional circuit). Best chromosome - $\mathbb{C}_3$. **Conclusion**: a fully functional circuit is evolved.

5. S_4: $T_4 = (I(7,128), O(4,128))$; $Y = \{y_1, y_2, y_3, y_4\}$. *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: Random. Best chromosome - $\mathbb{C}_4$. **Conclusion**: Decomposition by outputs: S_5 - easy evolved; S_6.

6. S_5: $T_5 = (I(7,128), O(1,128))$; $Y = \{y_1\}$; *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: $\mathbb{C}_4$. Best chromosome - $\mathbb{C}_5$. **Conclusion**: A fully functional circuit with reduced number of active logic gates is evolved.

7. S_6: $T_6 = (I(7,128), O(3,128))$; $Y = \{y_2, y_3, y_4\}$. *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: Random. Best chromosome - $\mathbb{C}_6$. **Conclusion**: Decomposition by outputs: S_7 - easy evolved; S_8.

8. S_7: $T_7 = (I(7,128), O(1,128))$; $Y = \{y_2\}$; *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: $\mathbb{C}_6$. Best chromosome - $\mathbb{C}_7$. **Conclusion**: A fully functional circuit with reduced number of active logic gates is evolved.

9. S_8: $T_8 = (I(7,128), O(1,128))$; $Y = \{y_3\}$; *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: $\mathbb{C}_6$. Best chromosome - $\mathbb{C}_8$. **Conclusion**: A fully functional circuit with reduced number of active logic gates is evolved.

10. S_9: $T_9 = (I(7,128), O(1,128))$; $Y = \{y_4\}$; *Purpose of evolution*: evolving a fully functional circuit; *Initialisation*: $\mathbb{C}_6$. Best chromosome - $\mathbb{C}_9$. **Conclusion**: A fully functional circuit with reduced number of active logic gates is evolved.

Once the EHW-oriented decomposition is complete, the fully functional circuit can be generated as shown in Fig. 2. This can be done if the synthesis of the optimised circuit is not required to be performed. Since, only output decomposition is allowed in this example, each of the sub-circuits is implemented independently and realizes the set of outputs from the complex system. Obviously, that this circuit structure is not optimal, since the circuit optimisation has not been performed.

So, the bidirectional incremental evolution schedule can be defined as decision tree and can be summarised as shown in Fig. 3. This sequence of tasks forces the evolution first to develop its partitioning vectors and then to optimise the

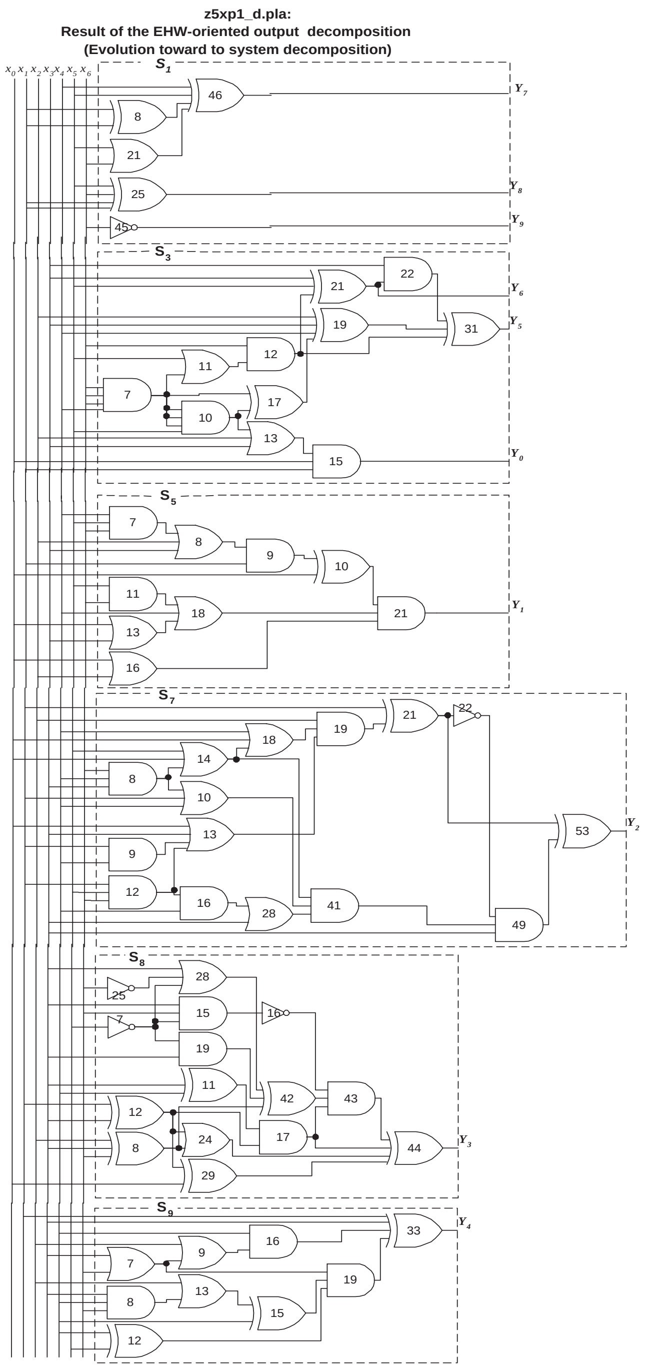

z5xp1_d.pla:
Result of the EHW-oriented output decomposition
(Evolution toward to system decomposition)
x0 x1 x2 x3 x4 x5 x6
S1
S3
S5
S7
S8
S9
Y7
Y8
Y9
Y6
Y5
Y0
Y1
Y2
Y3
Y4

Figure 2. z5xp1_d.pla: Bidirectional incremental evolution (Stage 1: Evolution toward to a modularised system).

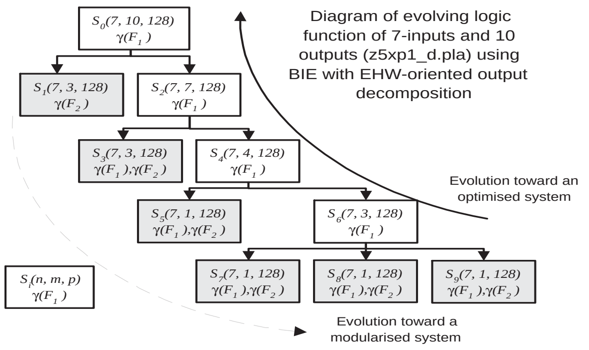

S_0(7, 10, 128) γ(F_1)
S_1(7, 3, 128) γ(F_2)
S_2(7, 7, 128) γ(F_1)
S_3(7, 3, 128) γ(F_1),γ(F_2)
S_4(7, 4, 128) γ(F_1)
S_5(7, 1, 128) γ(F_1),γ(F_2)
S_6(7, 3, 128) γ(F_1)
S_7(7, 1, 128) γ(F_1),γ(F_2)
S_8(7, 1, 128) γ(F_1),γ(F_2)
S_9(7, 1, 128) γ(F_1),γ(F_2)
S_i(n, m, p) γ(F_1)
Diagram of evolving logic function of 7-inputs and 10 outputs (z5xp1_d.pla) using BIE with EHW-oriented output decomposition
Evolution toward an optimised system
Evolution toward a modularised system

Figure 3. z5xp1_d.pla: Diagram of performing a bidirectional incremental evolution (Experiment A). The i-th system S_i is evaluated according to the truth table of n inputs, m outputs and p input-output combinations. This evaluation process can be defined as $S_0(n, m, p)$. Dynamic fitness function [13] contains evaluation of: (1) circuit functionality, F_1; (2) the number of logic gates used in the circuit, F_2. $\Upsilon(F_j)$ defines the evaluation process performed using criteria F_j.

Table 1. z5xp1_d.pla: Metrics of sub-systems obtained during evolution. The systems are decomposed according to the metrics shown in bold. For example, after analysis of the system S_2, the easy evolved system S_3 is composed from the outputs y_0, y_5 and y_6. The remaining outputs are evaluated in the following system S_4.

S_j	$f^o_{y_j}$						
	y_0	y_1	y_2	y_3	y_4	y_5	y_6
S_2	**99.22**	89.06	88.28	70.31	59.37	**84.37**	**93.75**
S_4		**92.18**	85.19	83.59	87.5		
S_6			**88.28**	**95.31**	**90.62**		

structures of assembled sub-tasks. The mapping of the performance of these tasks into the scale of evolution process is shown in Fig. 4. Note that once stage 1 of the bidirectional incremental evolution is completed, the fully functional circuit can be assembled without optimisation process. Fig. 4 depicts the performance of direct and bidirectional incremental evolution. The comparison is made for the evolution process when the circuit functionality has been evolved. Therefore, the evolution of sub-system S_1 is not taken into account, since it is fully functional after the evolution of system S_0. The fully functional circuit can be generated after 75 000 generations. During these generations the problems with less complexity has been evolved. The fully functional circuit implemented z5xp1_d.pla has been evolved using direct evolution after $5\ 000\ 000$ generations. The best functionality circuit evolved after 150 000 generations using direct evolution is 91.40625. That is relatively low. Also, during first stage of bidirectional incremental evolution the size of fully functional circuit evolved has been reduced from 157 logic gates to 85 (see Table 3). This shows that even during first stage of bidirectional incremental evolution the size of circuit can be reduced if a dynamic fitness function is used in the evaluation process.

Stage 2. Evolution toward to an optimised system The main purpose of this evolution stage is to optimise the size of the evolved circuit. During this stage of bidirectional incremental evolution, the sub-circuit of larger complexity is assembled from various sub-circuits of lower complexity, and the truth table of this circuit is also assembled from the two truth tables describing the behaviour of sub-systems. Evolution is performed under the new chromosome genotype with new evaluation criteria.

For example, the sub-circuits $\mathbb{C}(T_2)$ and $\mathbb{C}(T_3)$ and evaluated by truth tables T_2 and T_3 are evolved separately. More complex task can be define by the truth table T that is assembled from the truth tables T_2 and T_3: $T \in T_2 \cup T_3$. This task is described by chromosome $\mathbb{C}(T_2, T_3)$. The fully functional circuit implemented the truth table T is generated based on the sub-circuits $\mathbb{C}(T_2)$ and $\mathbb{C}(T_3)$. In this case, the chromosome genotype $\mathbb{C}(T_2, T_3)$ is synthesized based on chromosome genotypes $\mathbb{C}(T_2)$ and $\mathbb{C}(T_3)$.

The truth tables of these two sub-circuits are assembled into a larger one. The generated new genotype creates the initial population for the optimisation process. The evolution process is first undertaken on the new generated circuit. Then a circuit of larger complexity is generated and evolution repeats again. The process will continue until the complex circuit is assembled and optimised using evolutionary process.

This process can also reduce the size of the evolved circuit drastically. For example, the size of circuit synthesised after the first stage of bidirectional incremental evolution has been reduced from 85 logic gates to 54.

Thus, the actual size of fully functional z5xp1 circuit has been reduced from 157 logic gates to 54 (see Table 3). In other words the optimisation process reduced the number of logic gates by a factor of about 3.

5.2. Experiment B

In this experiment the complex system is divided into sub-systems by means of output and Shannon's decompositions. In order to compare the differences between the evolution processes, the evolution of first system S_0 is performed for both experiments.

The sub-system S_1 is generated according to the same specification given in experiment A. Some of metrics $f^{io}_{y_j}$ in system S_0 are equal to 100.0.

Table 2. Parameters of S_0 sub-system obtained during evolution of z5xp1_d.pla. The metrics specified in Eq. 2 and Eq. 3 are calculated for sub-system S_0. $f^o_{\mathbf{y_j}}$ defines that the sub-function described by output y_j is fully functional. Metrics $f^{io}_{y_j}|_{x_i=0}$ and $f^{io}_{y_j}|_{x_i=1}$ correspond to the Shannon's decomposition by variable x_i. Hence, $f^{io}_{y_j}|_{x_i=0} = f^{io}_{y_j}|_{x_i=1} = 100$ for all variables x_i. If $f^o_{\mathbf{y_j}} < 100$ and $f^{io}_{y_j}|_{x_i=0} = 100$ then the sub-circuit described the sub-function with $x_i = 0$ is fully functions. The same implied for metric $f^{io}_{y_j}|_{x_i=1}$.

y_j	$f^o_{\mathbf{y_j}}$	$f^{io}_{y_j}$													
		x_0		x_1		x_2		x_3		x_4		x_5		x_6	
		0	1	0	1	0	1	0	1	0	1	0	1	0	1
y_0	92.9689	**100**	86	**100**	86	99	88	86	**100**	94	93	94	93	94	93
y_1	89.0625	80	99	**100**	79	71	69	54	86	69	71	68	72	71	69
y_2	69.5312	61	79	71	69	54	86	69	71	68	72	71	69	69	71
y_3	82.8125	77	90	91	75	88	79	88	79	79	88	85	82	82	85
y_4	60.9375	60	63	60	63	66	57	63	60	63	60	60	63	66	57
y_5	81.25	82	82	82	82	82	82	75	88	88	75	88	75	75	88
y_6	62.5	63	63	63	63	63	63	63	63	75	50	75	50	50	75
y_7	**100.0**	100	100	100	100	100	100	100	100	100	100	100	100	100	100
y_8	**100.0**	100	100	100	100	100	100	100	100	100	100	100	100	100	100
y_9	**100.0**	100	100	100	100	100	100	100	100	100	100	100	100	100	100

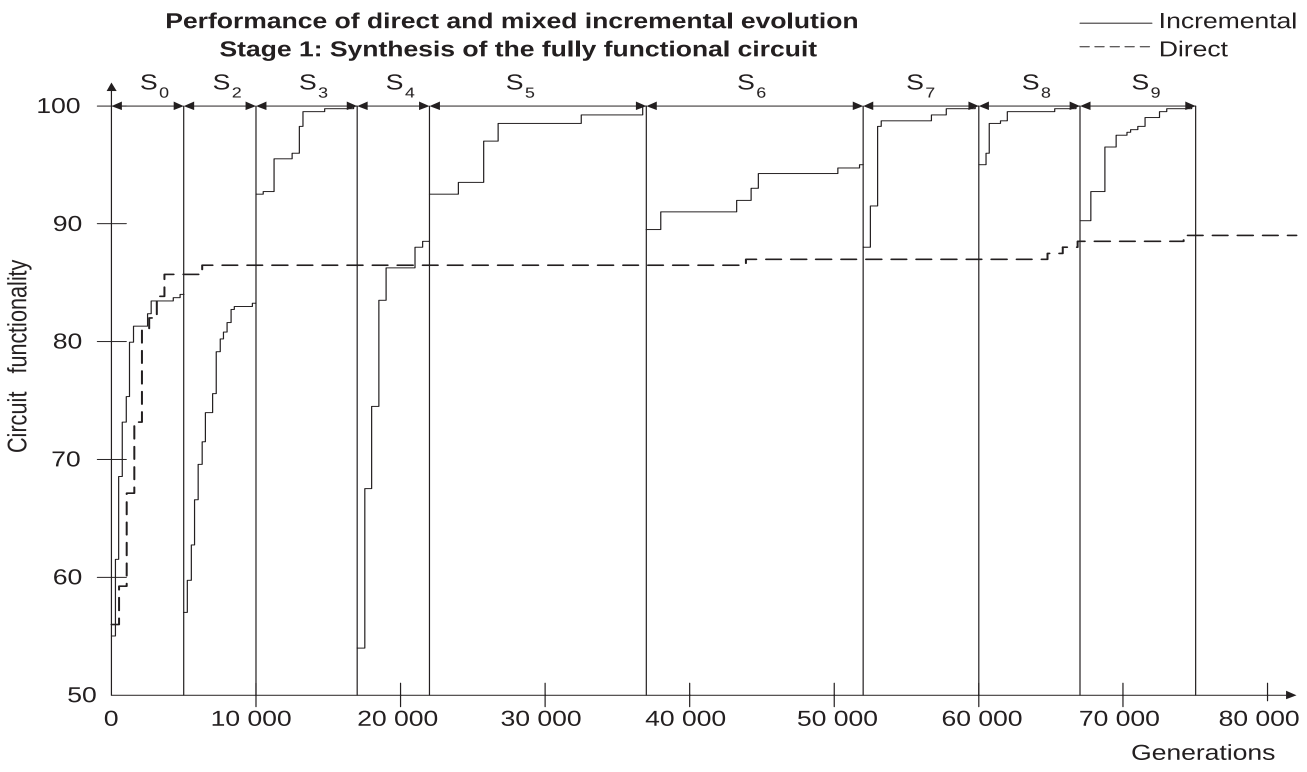


Figure 4. z5xp1_d.pla: Performance of direct and bidirectional incremental evolution in the circuit design problem. The maximum fitness per generation is plotted for each of the two approaches. The direct evolution (dotted line) makes slight progress at the first and stalls after about 10 000 generations. The plot is an average of 100 simulations. Incremental evolution, however, proceeds through several task transitions (seen as abrupt drop-offs in the plot), and eventually solves the goal-task. The incremental plot is a result of one simulation.

Table 3. z5xp1_d.pla: History of the incremental evolution with the EHW-oriented output decomposition.

Sub-system					ES performance		
S_i	n	m	p	Y	N_{gen}	$F_2(N_f)$	$F_2(N_f^{opt})$
Direct evolution							
S	7	10	128	Y	150 000	-	-
$\overline{F_1^{bf}} = 91.40625$							
Stage 1: Evolution toward to a modularised system							
S_0	7	10	128	Y	5 000	-	-
S_1	7	3	128	$\{y_7, y_8, y_9\}$	5 000	20	9
S_2	7	7	128	Y	5 000	-	-
S_3	7	3	128	$\{y_0, y_5, y_6\}$	15 000	25	21
S_4	7	4	128	Y	5 000	-	-
S_5	7	1	128	$\{y_1\}$	15 000	23	17
S_6	7	3	128	Y	15 000	-	-
S_7	7	1	128	$\{y_2\}$	15 000	30	25
S_8	7	1	128	$\{y_3\}$	15 000	26	24
S_9	7	1	128	$\{y_4\}$	15 000	23	13
Total: $\sum F_2(N_f^{opt})$					110 000	157	85
Stage 2: Evolution toward to an optimised system							
Total: $\sum F_2(N_f^{opt})$					200 000	85	54

For example, $f^{io}_{y_0}|_{x_1=0} = f^{io}_{y_1}|_{x_1=0} = 100.0$m i.e. the circuit implements completely the truth table generated from the outputs y_0 and y_1 and with condition that $x_1 = 0$. This truth table contains 6 inputs, 2 outputs and 64 input combinations. Therefore, if the Shannon's decomposition is allowed to be performed, the next fully functional sub-circuit contains the structure defined by this truth table. The truth table generated with condition that $x_1 = 1$ has the same parameters. Obviously, it is easier to evolve a circuit with fewer input combinations.

Note, that this is not the only possible set of sub-circuits that can be generated by analysing the metric data of S_0. One can decompose the system in terms of variable x_0, since $f^{io}_{y_0}|_{x_0=0} = 100.0$ or in terms of variable x_3, since $f^{io}_{y_0}|_{x_3=1} = 100.0$. We choose the partitioning vector mentioned above because in this case we divide sub-system not only in terms of variable x_1, but also the two outputs y_0 and y_1 can be involved in this partitioning process.

Fig. 5 shows bidirectional incremental evolution allowing both output and Shannon's decompositions. This diagram is different from one illustrated in Fig. 3. 13 sub-systems had to be generated in order to perform the first stage of bidirectional incremental evolution. The generated tasks are easier then one discussed in Experiment A, since they evolve functions with fewer number of inputs and therefore a smaller number of input/output combinations. Another difference is that this process can be performed in

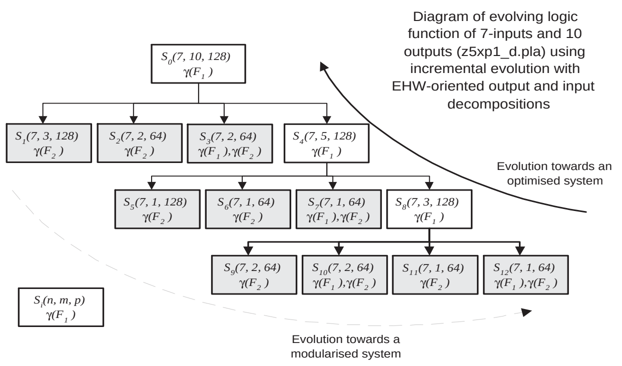


Diagram of evolving logic function of 7-inputs and 10 outputs (z5xp1_d.pla) using incremental evolution with EHW-oriented output and input decompositions
$S_0(7, 10, 128)$ $\gamma(F_1)$
$S_1(7, 3, 128)$ $\gamma(F_2)$
$S_2(7, 2, 64)$ $\gamma(F_2)$
$S_3(7, 2, 64)$ $\gamma(F_1),\gamma(F_2)$
$S_4(7, 5, 128)$ $\gamma(F_1)$
Evolution towards an optimised system
$S_5(7, 1, 128)$ $\gamma(F_2)$
$S_6(7, 1, 64)$ $\gamma(F_2)$
$S_7(7, 1, 64)$ $\gamma(F_1),\gamma(F_2)$
$S_8(7, 3, 128)$ $\gamma(F_1)$
$S_9(7, 2, 64)$ $\gamma(F_2)$
$S_{10}(7, 2, 64)$ $\gamma(F_1),\gamma(F_2)$
$S_{11}(7, 1, 64)$ $\gamma(F_2)$
$S_{12}(7, 1, 64)$ $\gamma(F_1),\gamma(F_2)$
$S_i(n, m, p)$ $\gamma(F_1)$
Evolution towards a modularised system

Figure 5. z5xp1_d.pla: Diagram of bidirectional incremental evolution (Experiment B).

parallel since 3 or more sub-systems can be evolved at the same time. Therefore, the evolution with output and Shannon's decompositions can be implemented using a parallel system, which will decrease the computation time drastically.

During first stage of our method the size of circuit has been reduced from 245 logic gates to 139. This is still very large circuit in comparison with the one generated in experiment A (85 logic gates). $100\ 000$ generations are required to complete the first stage.

The circuit optimisation has been carried out during $100\ 000$ generations. A circuit of 60 logic gates has been synthesised. This circuits larger then one evolved in experiment A by 4 logic gates.

6. Conclusion

This paper shows that even when a task is too difficult to evolve directly, evolution can be applied incrementally in two directions to achieve the desired complex behaviour. In this approach, evolution is performed in two directions: from complex system to sub-systems and from sub-systems to complex system. The experimental results show that bidirectional incremental evolution performs significantly better then direct evolution. Also, it has been demonstrated how the evolution process can be different when the various types of decomposition are allowed. The approach should be applicable to many real EHW applications, where often a natural hierarchy of behaviours from simple to complex exists.

The experiments on evolving digital logic function constitute a starting point for research on methods that evolve complex general behavior in two directions and on methods that investigate the functional interconnectivity inside given complex problem. There are many tasks that have to be solved in this direction.

7. Acknowledgement

The author would like to acknowledge Prof. J.W. Mills, Indiana University for his fruitful comments made about this work during ICES'2000 and Dr. M.Rutter and O.Lewis for proof-reading a draft.

References

[1] Iwata M., Kajitani I., Yamada H., Iba H., and Higuchi T. A pattern recognition system using evolvable hardware. In *Proc. of the Fifth International Conference on Parallel Problem Solving from Nature (PPSNIV)*, volume LNCS 1141 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, 1996.

[2] Murakawa M., Yoshizawa S., Kajitani I., Furuya T., Iwata M., and Higuchi T. Hardware evolution at function level. In *Proc. of the Fifth International Conference on Parallel Problem Solving from Nature (PPSNIV)*, Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 1996.

[3] Kalganova T. An extrinsic function-level evolvable hardware approach. In Poli R., Banzhaf W., Langdon W.B., Miller J., Nordin P., and Fogarty T.C., editors, *Proc. of the Third European Conference on Genetic Programming, EuroGP2000*, volume 1802 of *Lecture Notes in Computer Science*, pages 60–75, Edinburgh, UK, 2000. Springer-Verlag.

[4] Koza J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[5] Torresen J. A divide-and-conquer approach to evolvable hardware. In Sipper M., Mange D., and Perez-Uribe A., editors, *Proc. Of the 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES'98)*, volume 1478 of *Lecture Notes in Computer Science*, pages 57–65, Lausanne, Switzerland, 1998. Springer-Verlag, Heidelberg.

[6] Torresen J. Increased complexity evolution applied to evovable hardware. In *Smart Engineering System Design, ANNIE'99*. St. Louis, USA, 1999.

[7] Miller J.F., Kalganova T., Lipnitskaya N., and Job D. The genetic algorithm as a discovery engine: Strange circuits and new principles. In *Proc. of the AISB'99 Symposium on Creative Evolutionary Systems, CES'99*, ISBN 1-902956-03-6, pages 65–74. Edinburgh, UK, The Society for the Study of Arificial Intelligence and Simulation of Behaviour, April 1999.

[8] Vassilev V. and Miller J. The advantages of landscape neutrality in digital circuit evolution. In Miller J., Thompson A., Thomson P., and Fogarty T.C., editors, *Proc. Of the 3rd Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES 2000)*, volume 1801 of *Lecture Notes in Computer Science*, pages 252–263, Edinburgh, UK, 2000. Springer.

[9] Gomez F. and Miikkulainen R. Incremental evolution of complex general behaviour. *Adaptive Behaviour.*, 5:317–342, 1997.

[10] Gomez F. and Miikkulainen R. Solving non-markovian control tasks with neurevolution. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI'99)*, Stockholm, Sweden, 1999. Denver: Morgan Kaufmann.

[11] Filliat D., Kodjabachian J., and Meyer J.A. Incremental evolution of neural controllers for navigation in a 6-legged robot. In Sugisaka and Tanaka, editors, *Proc. of the Fourth International Symposium on Artificial Life and Robotics*. Oita Univ. Press, 1999.

[12] Miller J. F., Thomson P., and Fogarty T. C. Genetic algorithms and evolution strategies. In Quagliarella D., Periaux J., Poloni C., and Winter G., editors, *Engineering and Computer Science: Recent Advancements and Industrial Applications*. Wiley, 1997.

[13] Kalganova T. and Miller J. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In Stoica A., Keymeulen D., and Lohn J., editors, *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, pages 54–63. IEEE Computer Society, July 1999.

[14] Miller J. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO'99)*, volume 1 of *ISBN 1-55860-611-4*, pages 1135–1142, Orlando, USA, July 1999. Morgan Kaufmann, San Francisco, CA.

[15] Floreano D. and Mondada F. Hardware solutions for evolutionary robotics. In Husbands P. and Meyer J-A., editors, *Proc. of the First European Workshop on Evolutionary Robotics*. Berlin: Springer-Verlag, 1998.

[16] Fukunaga A.S. and Kahng A.B. Improving the performance of evolutionary optimization by dynamically scaling the evolution function. In *Proc. of the 1995 IEEE Conference on Evolutionary Computation*, volume 1, pages 182–187, Perth, Australia, 29-1 1995. IEEE Press.