Baradavka I. and T. Kalganova (2003) Assembling Strategies in Extrinsic Evolvable Hardware with Bi-directional Incremental Evolution. Proc. of the 6th European Conference on Genetic Programming, EuroGP2003, Essex, UK. Published by Springer-Verlag. Vol. 2610. pp. 276-285.

Assembling Strategies in Extrinsic Evolvable Hardware with Bidirectional Incremental Evolution

Igor Baradavka and Tatiana Kalganova

Department of Electronic and Computer Engineering,
Brunel University, Uxbridge, UB8 3PH, UK
{igor.baradavka, tatiana.kalganova}@brunel.ac.uk
http://www.brunel.ac.uk/~eestttk/

Abstract. Bidirectional incremental evolution (BIE) has been proposed as a technique to overcome the "stalling" effect in evolvable hardware applications. However preliminary results show perceptible dependence of performance of BIE and quality of evaluated circuit on assembling strategy applied during reverse stage of incremental evolution. The purpose of this paper is to develop assembling strategy that will assist BIE to produce relatively optimal solution with minimal computational effort (e.g. the minimal number of generations).

1 Introduction

For almost a decade evolvable hardware (EHW) has been limited to evolution of relatively small logic circuits [1], [2], [3]. The solution of scalability problem will open the possibility to use evolvable hardware technique for real-world applications.

Recently a number of techniques have been developed to overcome this problem. Some of them are focused on speeding up the genetic algorithm computations using principles of parallelism [4], [5], [6]. Other approaches to the problem have used variable length chromosomes [7], function-level evolution [8], [9], automatically defined functions [10]. Further, a divide-and-conquer approach (known also as an increased complexity evolution) has been introduced [11], [12], [13], [14]. The basic idea of this approach is to split problems into sub-problems and evolve each sub-problem separately. The principle of this approach is very similar to incremental evolution introduced to solve complex tasks using evolutionary processes [15], [16], [17]. In incremental evolution approach the complexity of evolved tasks increases with evolution. This allows to reduce the computational effort to solve the task and overcome the "stalling" effect in evolution [18], [16]. The approach can successfully evolve relatively large circuits if there is human participation at the decomposition stage. The final solution is unlikely optimal because it is assembled from separately evolved logic circuits. Bidirectional Incremental Evolution (BIE) allows to eliminate these two drawbacks [19]. In BIE approach the EHW-oriented decomposition has been introduced to provide automatic decomposition of circuits and perform direct incremental evolution (DIE). The optimisation of obtained solution has been also carried out during reverse incremental evolution (RIE), where the sub-tasks are step-by-step assembled and further optimised using evolutionary process. Although larger circuits have been evolved using BIE with significant reduction in the evolution duration, the assemble strategies during RIE have not been considered in detail. The use of correct assembling technique is very important because it can significantly influence on both duration of evolution and quality of evolved circuits. This paper is devoted to analysis of assembling strategies during RIE.

2 Bidirectional incremental evolution

The bidirectional incremental evolution contains 2 main evolutionary processes: (1) Direct Incremental Evolution (DIE); (2) Reverse Incremental Evolution (RIE).

2.1 Direct incremental evolution (DIE)

The main purpose of DIE is to obtain a fully functional solution of given task. This is achieved by decomposition of complex task to certain set of sub-tasks of reduced complexity when the stalling effect takes place [19]. Thus BIE approach *guaranties* that at the end of DIE stage of evolutionary process the fully functional solution is found.

An example of BIE is given in Fig.1. In this example DIE has been completed in two main decomposition steps. The DIE always starts with evolution of initial complex circuit S_0. Once the stalling effect has appeared, this circuit is decomposed into three smaller sub-circuits (see Step 1, Fig.1). The sub-circuits S_1 and S_2 are obtained as a result of Shannon's decomposition and sub-circuit S_3 is the left part of the circuit S_0. The evolution of the sub-circuits S_1 and S_3 has been completed but another EHW-oriented decomposition (Step 2 in Fig.1) is required to finish evolution of the sub-circuit S_2. During this step the sub-circuit S_2 is decomposed into another three sub-circuits using both output and Shannon's decompositions.

2.2 Reverse incremental evolution (RIE)

The solution of the initial complex task is usually found during the DIE. With relation to the EHW the result of DIE is a set of combinational logic circuits. This set implements the set of logic functions or sub-tasks. These sub-tasks are defined during EHW-oriented decomposition. The design that implements the initial complex logic function could be easily achieved by joining all evolved sub-circuits together. The main disadvantage of this approach is that obtained complex circuit is far from the optimal solution. It happens because the evolution during DIE is performed separately for each sub-circuit. Usually there exist some

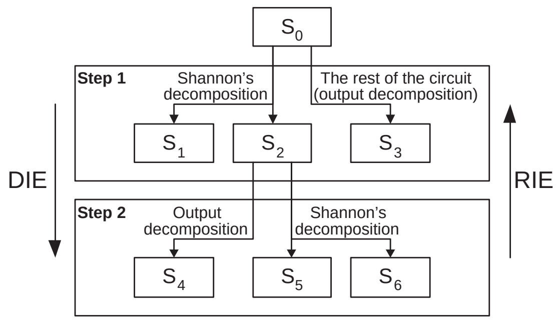


Fig. 1. An example of evolutionary process that may take place during DIE.

common parts between different sub-circuits. To overcome this problem further reverse incremental evolution of assembled circuits can be performed.

During RIE the evolution towards optimised circuit is carried out and the most important role during this stage is played by assembling process. At this stage the logic circuits are assembled back and further optimised using evolutionary process. This process consists of three assembling sub-processes:

1. Assembling process at RIE level;
2. Assembling process at decomposition level;
3. Assembling process at chromosome level.

Assembling strategy at RIE level defines how many logic circuits can be assembled at once in order to produce successful and fast evolution. For example, sub-circuits S_1, S_2 and S_3 can be assembled together at once (Fig.2.B). From another point of view the evolution of sub-circuits S_1 and S_2 can be carried out as first step and only then the combined evolution of S_1, S_2 and S_3 can be considered.

The type of the decomposition used during DIE determines the way how the sub-circuits are linked together into a larger circuit during RIE. Such decomposition dependent circuit linkage is performed during the second assembling sub-process. For example, sub-circuits S_5 and S_6 shown in Fig.1 can be merged using a set of multiplexors because they has been created during DIE as the result of Shannon's decomposition of S_2. Next, S_4 and S_{5-6} can be linked together. No additional sub-circuits are necessary at this step because the output decomposition has been used.

The way how the logic gates of sub-circuits are assembled together into one chromosome is considered during third assembling sub-process. This defines the positions of logic gates in the assembled chromosome. Since the new chromosome is built from several separate circuits the result of this linkage must be the combinational logic circuit.

This paper concentrates on the extrinsic EHW proposed in [9], [20]. The chromosome representation of circuit is shown in Fig.3. The circuit is described by set of logic gates and the circuit output genes. Each logic gate is defined by

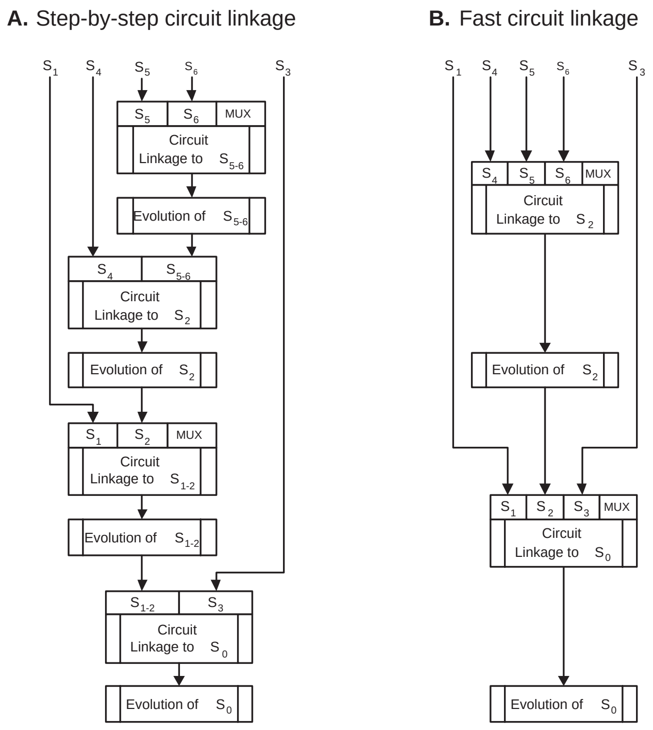
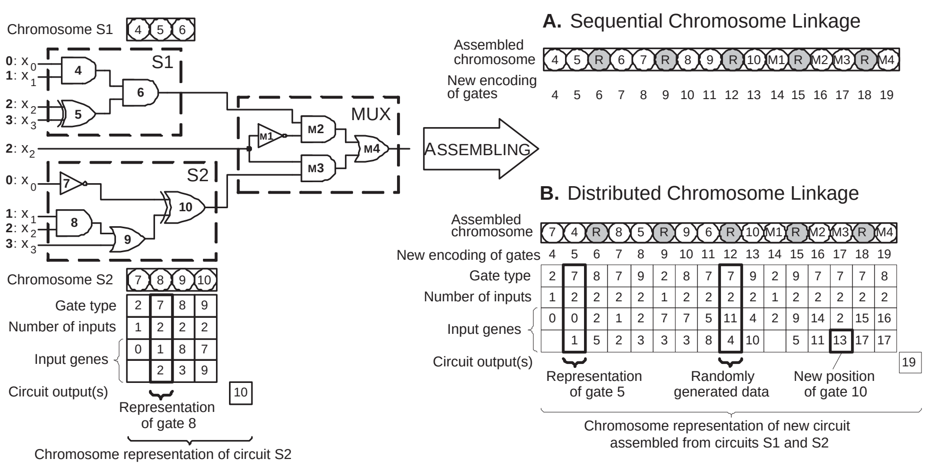


Fig. 2. Assembling strategies in RIE. There are two possible schemes of the evolutionary process RIE according to the decompositions shown in Fig.1, where MUX is the set of multiplexors.



Chromosome S2	7	8	9	10
Gate type	2	7	8	9
Number of inputs	1	2	2	2
Input genes	0	1	8	7
		2	3	9
Circuit output(s)				10

Assembled chromosome	4	5	R	6	7	R	8	9	R	10	M1	R	M2	M3	R	M4
New encoding of gates	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Assembled chromosome	7	4	R	8	5	R	9	6	R	10	M1	R	M2	M3	R	M4	
New encoding of gates	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
Gate type	2	7	8	7	9	2	8	7	7	9	2	9	7	7	7	8	
Number of inputs	1	2	2	2	2	1	2	2	2	2	1	2	2	2	2	2	
Input genes	0	0	2	1	2	7	7	5	11	4	2	9	14	2	15	16	
		1	5	2	3	3	3	8	4	10		5	11	13	17	17	
Circuit output(s)																	19

Fig. 3. Generation of new chromosome from 2 sub-circuits created as the result of Shannon's decomposition by input x_2, where R is a redundant randomly generated logic gate. Inputs and gates are sequentially encoded in the same numeric space.

the following genes: cell type, number of inputs, inputs (e.g. connectivity with other logic gates and circuit inputs).

3 Assembling processes

In this section we will consider in detail specific futures of all assembling processes mentioned in previous section.

3.1 Assembling strategies at RIE level

The success of evolutionary process is entirely depends on how the evolutionary process towards optimised system can be carried out. From this point of view it is important to consider how the circuits are linked to each other during RIE. We define two basic linkage strategies:

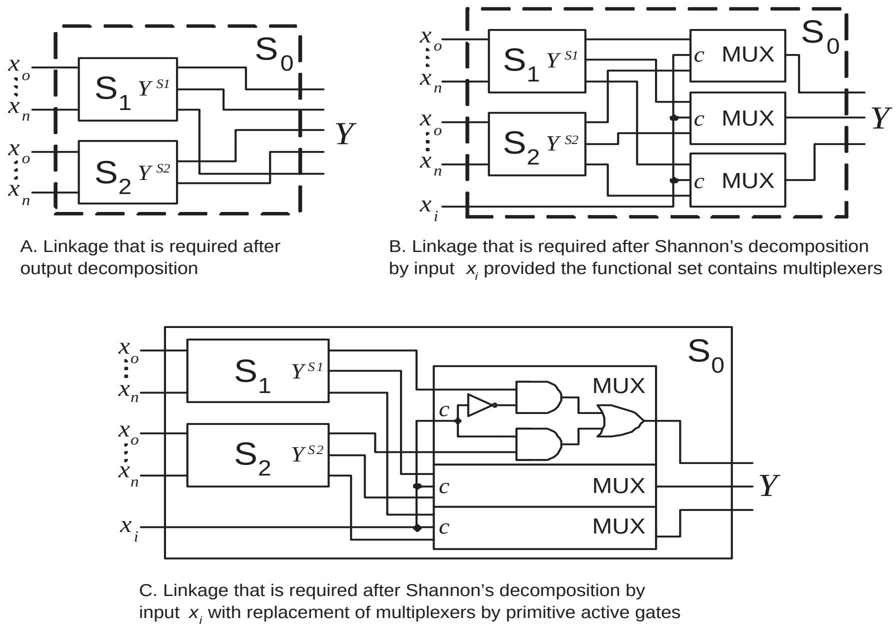
1. Step-by-step circuit linkage.
2. Fast circuit linkage.

The idea of step-by-step linkage is to assemble only two sub-circuits at once and perform evolutionary process under assembled circuit. Step-by-step linkage allows gradually increase the complexity of the circuit and optimise it at the early stages of evolution. This process is illustrated in Fig. 2.A. A large number of evolutionary processes are used in order to synthesise required circuit. But these evolutionary processes are carried out under the relatively easy tasks. As the result relatively small number of generations is required to successfully complete their evolution. Therefore, they require less computational efforts. For example, let us consider BIE shown in Fig.1. There are 3 sub-circuits that have been generated during Step 2. The step-by-step circuit linkage at Step 2 involves two evolutionary processes. First, the circuit S_{5-6} assembled from sub-circuits S_5 and S_6 is evolved. Then the sub-circuit S_4 is added to already optimised S_{5-6} and evolutionary process is carried out again for S_2 circuit. Next optimised circuit S_2 is assembled with S_1 and optimisation of result circuit S_{1-2} is carried out. The linkage of S_{1-2} and S_3 to S_0 finalises the Step 1 (see Fig.1) and final evolution of entire circuit S_0 is performed.

The idea of the fast circuit linkage is to evolve the assembled circuit without intermediate steps. For instance, the optimisation process shown in Fig.2.B is performed just in two steps. In this case the number of optimisation sub-tasks is two times less, but the evolution will be undertaken under more complex tasks in comparison with step-by-step circuit linkage and require more computational effort to optimise every task.

3.2 Circuit linkage at the decomposition level

There is one global difference between output and Shannon's decompositions. The circuits evolved using output decomposition are independent from each other. This means that they do not require any additional "linkage" sub-circuit in order to obtain a fully functional solution of assembled circuit (Fig.4.A).



x_o
x_n
S_1 Y^{S1}
S_2 Y^{S2}
S_0
Y
A. Linkage that is required after output decomposition
x_i
MUX
c
B. Linkage that is required after Shannon's decomposition by input x_i provided the functional set contains multiplexers
C. Linkage that is required after Shannon's decomposition by input x_i with replacement of multiplexers by primitive active gates

Fig. 4. Circuit linkage at the decomposition level, where Y, Y^{S_1} and Y^{S_2} are the outputs of the sub-circuits S_0, S_1 and S_2 respectively.

Shannon's decomposition always produces a pair of logic circuits. Each of these circuits is fully functional only for particular part of input-output combination matrix of initial circuit. As the result, a set of multiplexors is required to join the outputs of these circuits (Fig.4.B). The number of multiplexors is equal to the number of outputs in the circuits to be linked. For example, the logic circuit shown in Fig.4.B is generated using 3 multiplexors because each sub-circuit has 3 outputs. The implementation of the linkage circuit depends on the chosen functional set of logic gates. Thus, if the multiplexer is included into the functional set, the "linkage" circuit can be represented using multiplexors only. Otherwise multiplexer could be replaced by 4 primitive active logic gates. A set of multiplexors could share one NOT-gate as it is shown in Fig.4.C. Thus, in this example only 10 primitive active gates are needed to represent 3 multiplexors.

3.3 Assembling strategies at the chromosome level

Logic gates can be assembled differently inside chromosome. For example, in [19] the sequential chromosome linkage has been considered. In this case all logic gates of the sub-circuit S_1 are placed at the most left position of the newly generated chromosome. Then all logic gates of the sub-circuit S_2 are placed just next to the logic gates of the sub-circuit S_1 and so on. The additional linkage circuit is placed at the most right position of the chromosome (see Fig.3.A).

The process of assembling one chromosome from 2 sub-circuits using distributed chromosome linkage is shown in Fig.3.B. The logic gates from the sub-circuits S_1 and S_2 are permutated in such way that the least left gates from the both sub-circuits are placed at the least left positions of the newly generated chromosome. The number of randomly generated cells is defined by given percentage of redundant logic gates in the circuit.

For both sequential and distributed chromosome linkages the number of cells C_{NEW} in newly generated chromosome is defined according to the following formula:

$$C_{NEW} = \frac{\sum_1^k C_i + C_{MUX}}{r},$$

where C_i is the number of active logic gates in i-th circuit, C_{MUX} is the number of logic gates required for special linkage circuit, k is the number of sub-circuits to be linked, r is the redundancy rate. The redundancy rate defines the ratio of the number of redundant logic gates to the total number of logic gates in the circuit [21]. The redundancy rate is fixed for all sub-circuits to be assembled. If the multiplexer is required it is placed at the least right position of the chromosome. If the chosen functional set of logic gates includes only primitive active gates, every multiplexer is replaced by 3 or 4 primitive logic gates as shown in Fig.4.C.

4 Experimental results

The purpose of our experiments is to investigate how the assembling strategy can influence on the algorithm performance. In order to do so 3 functions from standard benchmark library have been used: mult3.pla, m1.pla and squar5.pla. The functions have been chosen in a such way that the same experiments were performed under circuits with relatively various complexity and structure. The complexity of circuits has been considered in terms of the number of inputs and outputs. Thus, 3-bit multiplier (mult3.pla) has 6 inputs and 6 outputs. At the same time, m1 function (m1.pla) has 5 inputs and 12 outputs and squar5 function (squar5.pla) has 5 inputs and 8 outputs.

The initial data for the experiments are given in Table 1. The rudimentary $(1+\lambda)$ evolutionary strategy has been used [9]. Any type of genes in chromosome genotype allowed to be changed with constant gene mutation probability. The functional set of logic gates contains {AND, OR, EXOR, NOT}. Each function has been successfully evolved at least 100 times. One of the noticeable features of the BIE is that it *guaranties* to evolve fully functional solution. Therefore, after each run of evolutionary algorithm the fully functional solution has been obtained. The obtained experimental results are summarised in Table 2. The analysis of results shows that it is more difficult to evolve m1 function rather then mult3 and squar5. It can be seen that the slowest strategy is A (combination of Sequential and Step-by-step assembling strategies) for all 3 circuits.

Let us consider the average number of active logic gates in final circuits that have been evolved. Best results were achieved by strategies A and C and the

Table 1. Initial data, where η_{fc} is the limitation of number of generations after last change of fitness function, # is "the number of ..."

Circuit	**mult3**	**m1**	**squar5**
Max # rows	1	1	1
Max # columns for DIE	40	50	40
Levels back for DIE	40	50	40
Max # of inputs in the logic gate	2	2	2
Population size	5	5	5
# generations for DIE	100000	100000	100000
# generations for RIE	500000	500000	500000
η_{fc} for initial system evolution	14000	14000	14000
η_{fc} for DIE with $\gamma(F_1)$ [20]	22000	22000	22000
η_{fc} for DIE with $\gamma(F_1 + F_2)$ [20]	7000	7000	7000
η_{fc} for RIE	45000	45000	45000
# successful runs of BIE	100	100	100
Cell mutation rate	5%	5%	5%

worst - by strategy B (Table 2). However strategy A performed slower then strategy C although the quality of optimisation was the same.

Therefore, based on the experimental results one may conclude that combination of the step-by-step circuit linkage and distributed chromosome linkage allows to obtain the most optimal solution with minimal computational effort in terms of the number of generations used.

5 Conclusions and future work

This paper describes the evolutionary design of combinational logic circuits in terms of use of different assembling strategies in bidirectional incremental evolution. The distinctive feature of the algorithm is that it *guaranties* the evolution of fully functional circuit. We have introduced distributed assembling strategy, which allows us not only to assemble logic circuits with random genes into one chromosome but to make sure that the logic gates from *all* sub-circuits participate equally in evolution. We have also proposed step-by-step assembling strategy at decomposition level that allow us to optimise small sub-circuits on early stages of evolution when the performance of BIE does not seriously degrade because of complexity of the circuit. These two aspects allow us to significantly improve the quality of evolved circuits with minimal amount of computational efforts.

We have investigated several assembling strategies at both RIE and chromosome levels. Analysis of experimental results allow us to make following conclusions:

1. Different logic functions behave similarly for different linkage strategies.

2. The sequential chromosome linkage reduces the quality of optimisation when the fast circuit linkage is used. When the step-by-step circuit linkage is used, it also reduces performance of RIE.

Table 2. Experimental results, where Avg. No. is the average number, Min. No. is the minimal number, n is the number of inputs, m is the number of outputs, *s-b-s* and *fast* is the step-by-step and the fast circuit linkage respectively.

Circuit	**Strategy**	**A**	**B**	**C**	**D**
	Assembling at chromosome level	Sequential		Distributed	
(n, m)	Assembling at RIE level	*s-b-s*	*fast*	*s-b-s*	*fast*
mult3 (6, 6)	Avg. No. of generations (BIE)	487000	361000	545000	368000
	Avg. No. of generations during RIE	329000	197000	346000	173000
	Avg. No. of gates in evolved circuit	46	52	46	51
	Min. No. of gates found	37	35	34	35
m1 (5, 12)	Avg. No. of generations (BIE)	788000	403000	766000	460000
	Avg. No. of generations during RIE	577000	196000	511000	207000
	Avg. No. of gates in evolved circuit	67	77	64	81
	Min. No. of gates found	61	68	49	69
squar5 (5, 8)	Avg. No. of generations (BIE)	530000	282000	484000	295000
	Avg. No. of generations during RIE	407000	148000	328000	141000
	Avg. No. of gates in evolved circuit	39	50	39	48
	Min. No. of gates found	34	34	29	32

3. The step-by-step circuit linkage and distributed chromosome linkage provides the best synthesis and optimisation of combinational logic functions. However, if the better performance is required, the fast circuit linkage and and distributed chromosome linkage can be recommended as the most cost-effective technique.

So we can conclude that the use of correct assembling strategy is very important in implementation of BIE.

A great deal of further work could be done in the area. Automatic control of duration of evolutionary process could be introduced in order to improve further the computational effort of BIE algorithm.

References

1. Coello C. A., Christiansen A. D., and Hernndez A. A. Towards automated evolutionary design of combinational circuits. *Computers and Electrical Engineering*, 2000.
2. Higuchi T., Murakawa M., Iwata M., Kajitani I., Liu W., and Salami M. Evolvable hardware at function level. In *Proc. of IEEE 4th Int. Conference on Evolutionary Computation, CEC'97.* IEEE Press, NJ, 1997.
3. Thompson A. *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution.* PhD thesis, University of Sussex, School of Cognitive and Computing Sciences., 1996.
4. Birge J.R. Stochastic programming, computation and applications. *INFORMS, Journal on Computing*, pages 111–133, 1997.
5. Zenious S.A. Vladiviriou H. Parallel algorithms for large-scale stochastic programming in parallel computing and optimisation. pages 413–469, 1997.

6. Poli R. Evolution of graph-like programs with parallel distributed genetic programming. In Bäck T., editor, *Genetic Algorithms: Proc. of the Seventh International Conference.*, pages 346–353. Morgan Kaufmann, San Francisco, CA, 1997.
7. 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.
8. 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.
9. 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.
10. Koza J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.
11. 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.
12. Torresen J. Increased complexity evolution applied to evovable hardware. In *Smart Engineering System Design, ANNIE'99.* St. Louis, USA, 1999.
13. Torresen J. Two-step incremental evolution of a prosthetic hand controller based on digital logic gates. In *Proc. of the 4th Int. Conference on Evolvable Systems, ICES.*, Lecture Notes in Computer Science. Springer-Verlag, 2001.
14. Torresen J. A scalable approach to evolvable hardware. *Genetic Programming and evolvable machines*, 3(3), 2002.
15. Gomez F. and Miikkulainen R. Incremental evolution of complex general behaviour. *Adaptive Behaviour.*, 5:317–342, 1997.
16. 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.
17. 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.
18. Harvey I. Artificial evolution for real problems. In Gomi T., editor, *Proc. of the 5th Intl. Symposium on Evolutionary Robotics, Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97)*, Tokyo, Japan, 1997. AAI Books.
19. Kalganova T. Bidirectional incremental evolution in ehw. In *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware.* IEEE Computer Society, July 2000.
20. 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.
21. Kalganova T. and Miller J. Circuit layout evolution: An evolvable hardware approach. In *Coloquium on Evolutionary hardware systems. IEE Colloquium Digest.*, London, UK, 1999.