

Insufficiency of Piecewise Evolution

S. Kazadi, Y. Qi, I. Park, N. Huang, P. Hwu, B. Kwan, W. Lue, H. Li

May 6, 2001

Abstract

We describe an evolutionary design paradigm called piecewise evolution. This evolutionary design paradigm allows the gradual evolution of a piece of hardware using discrete functional stages. The paradigm removes designs from a population of designs which effectively lose functionality already discovered. Significant improvements in the evolution time of simple one-bit adders are reported. However, evolution of more complex devices does not seem to share the improvements in evolutionary speed of simple devices. These results are discussed in the context of epistasis and deceptiveness.

Introduction

In their 2000 paper on a novel evolutionary algorithm called *phased evolution* [1] Hounsell and Arsian describe a method of evolution which purports to have a significant advantage over the standard methods. This method, in which individual outputs of feed-forward networks are developed and later combined through rather standard techniques, is decidedly closer to standard digital circuit design techniques than those of other evolutionary methods. The great strength of this method comes from its ability to generate independent parts of the circuit without interference from other evolutionary stages, and then to combine the subcircuits using a logic reduction phase.

Indeed, this type of design is not without precedent. Several authors have investigated systems of this sort, with the most direct similarity coming from Kazadi [2][3]. In these papers, *conjugate schema* are introduced. Conjugate schema are subbases under which the evolution of a particular solution in a genetic algorithm system behaves as though the parts of the solution corresponding to individual subbases are independent of one-another. That is, if a basis B was itself the union of two subbases $B = B_1 \cup B_2$ and a fitness function f could be written as $f = f_1 + f_2$ where $f : \langle B \rangle \mapsto \mathfrak{R}$, $f_1 : \langle B_1 \rangle \mapsto \mathfrak{R}$, and $f_2 : \langle B_2 \rangle \mapsto \mathfrak{R}$, then B_1 and B_2 are conjugate schema of the function f over $\langle B \rangle$. It was demonstrated that significant gains could be gleaned from utilizing these structures.

It is essentially the same effect that Hounsell and Arsian are obtaining from their phased evolution. In this case, the fitness, if you will, is made up of several deliberately separate functions of the design. That is, the fitness of a complete structure is deliberately designed to satisfy the paradigm

$$f = f_1 + \dots + f_n \tag{1}$$

In effect, the design paradigm satisfies the functional design condition of conjugate schema. However, this cannot be placed in the language of conjugate schema because of the somewhat open-ended evolutionary space, unless one considers the space to be a direct product of several infinite-dimensional subspaces.

Indeed, other studies [4][6][9] have investigated the use of compartmentalization at the encoding level of evolutionary algorithms. These studies have indicated a theoretical advantage emanating from the use of compartmental evolutionary models, and have illuminated the role the lack of such compartmentalization

has in creating difficulty in evolutionary computations. These would seem to fall nicely in line with the performance improvements Hounsell and Arsian report.

There are, however, several weaknesses with phased evolution as described. Firstly, and perhaps most importantly, most devices are not linearly separable as in equation (1). Certainly, visual recognition systems, high level planners, and processors, are not linearly separable. This poses a rather significant constraint on the effectiveness of the method, as opposed to any other method. Moreover, this method does not illuminate any emergent compartmentalization, as both conjugate schema and transposons (in natural and evolutionary computation systems) seem to do. Finally, phased evolution fails to reuse previously evolved structures and has no explicit mechanism for creating and using neutral DNA [8]. This is a significant detriment to any evolutionary process, as it requires rediscovery of the structure of any redundant part.

In the present study, we describe an alternative evolutionary paradigm we call *piecewise evolution*. This is similar in spirit to phased evolution, but the separation comes not from the implementation, but rather from the function. The rest of the paper is arranged as follows. Section 1 describes the connectionist models we use in our study. Section 2 introduces the piecewise evolutionary paradigm. Section 3 provides the details of the computational studies done on this system. Finally, Section 4 provides a brief discussion of this and related future work, along with some concluding remarks.

1 Connectionist Hardware Model

We choose a connectionist model as the test bed for our study. This model is made up of two principle components. These components are both active and inactive nodes, and passive connections. Networks of these individual devices make up the completed device. A sample is shown in Figure 1.1.

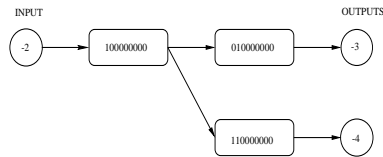


Figure 1.1: This shows a device built out of active and inactive nodes and passive connections. Multiple connections indicate a higher coupling between devices than single connections.

Passive nodes are nodes that either carry signals from the outside into the network, or carry signals from the network to the outside. These are hereafter referred to as *I/O nodes*. Active nodes receive signals from other nodes and send out specific signals in response to the sum of the incoming signals. Each active node is uniquely defined by a nine-digit activation number. This number uniquely specifies the output of the node given an input between zero and eight, with each output ranging between zero and nine. As an example, a node may have a number given by 100100100. This number means that if the node is receiving an input of zero, three, or six, it will send a signal of one in the next iteration. All other inputs will elicit a zero output. At each iteration, the inputs are summed from all other connected nodes, with multiple connections increasing the effect of a given node. The output from that node in the next iteration is the output indicated on the I/O table.

One notable difference of this model from that of a standard neural network is that there are no weights associated with the connections. In this way, all connections are *passive*, as they do not alter the signal passed to the next node. This requires multiple connections if emphasis is called for in the design.

Our model also makes use of nodes specified by negative numbers. These are I/O nodes, and allow information from the outside world to enter the network or to leave the network. Thus, in Figure 1.1, the -2 represents a node receiving information from the outside world while the -3 and -4 nodes giving information to the outside world.

Connections between nodes are directional, leading from a given node to another specified node.

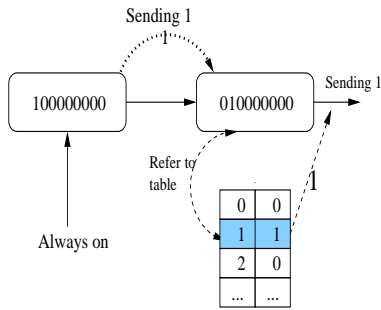


Figure 1.2: This image indicates how signals are defined, how they are carried between nodes, and the effects of multiple connections.

Each connection deposits a signal in the ending node equal in magnitude to that specified in the originating node's output table. This means that if multiple connections are present between two nodes, the magnitude of the signal deposited in the ending node is a multiple of the original signal. No restrictions are placed on the way in which connections may be constructed with the exception that connections may not exist between two I/O nodes.

The model is quite versatile in its ability to produce simple or complex computational structures. For instance, a simple one bit adder may be produced by utilizing six nodes, four of which are I/O nodes, as indicated in Figure 1.3.

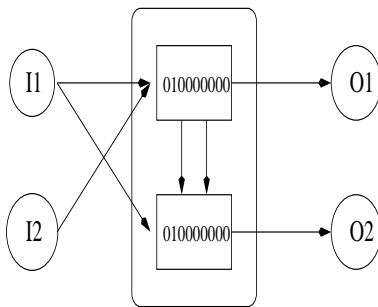


Figure 1.3: This figure depicts a simple one-bit adder.

On the other hand, one may create an oscillator by using the structure depicted in Figure 1.4

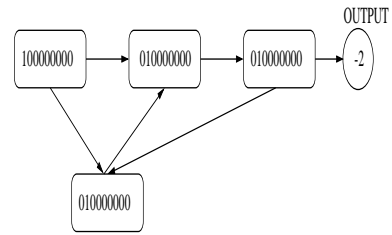


Figure 1.4: This figure depicts a simple oscillating device.

The generality of the system makes it an excellent model for evolution. Moreover, the ability of the specific active processors to be individually modified allows one to build special purpose nodes rather than trying to create complex structures based on rather inflexible uniform node designs. This is illustrated in Figure 3.1, in which the alteration of a single node makes it possible for an adder to have a symmetric design rather than requiring a nonsymmetric design. The ability of the model to make use of multiple node designs makes it rather versatile, and serves to improve its performance.

All of our simulations utilize an evolutionary system consisting primarily of a 100 x 100 toroidal lattice. Each spot on the lattice contains a single string encoding the network along with a score associated with the string. The score is a measure of how well the network carries out a specific task. Associated with the score is a life cycle length, expressed in iterations. This length indicates how long this particular string will wait before it replicates. The replication involves randomly choosing a nearby lattice spot and copying the string in question into the chosen spot. Half the time, a random mutation occurs when copying the string. If the space is occupied, a number is randomly chosen. The copy will continue if the random number exceeds a threshold defined by the relative scores of the two strings, effectively "killing" the existing string. No sexual reproduction of any kind is currently used in this study. The mutations used in this study, listed in the appendix, are rather simple mutations that represent very small changes to the existing network design. Despite resulting in rather small modifications of the network morphology, these mutations can often times cause very profound change in the network functionality.

2 Piecewise Evolutionary Model

Two major constraints in artificial hardware evolution are the factorial increase in the rate of evolution and epistasis due to complex interactions within a device. In previous studies [3], we have verified that noncompartmentalized evolutionary models result in an increase in computation time which is factorial in the number of building blocks. This requires us to adopt compartmentalization in the encoding of evolutionary algorithms. The same study indicates that compartmentalization will reduce the expected computation time from factorial in the number of building blocks to an expected computation time which is factorial in the number of building blocks per compartment multiplied by a linear term in the number of compartments. Compartmentalization produces the significant advantage in computational time because it allows discrete modules with specific functionalities to evolve individually.

However, by separating a device into independent compartments, the interaction between elements in different compartments is denied. Information cannot be shared freely among different functional groups. In reality, the complexity of devices requires interaction between different functional groups and even interaction between a single element in one compartment and one or more elements in another compartment. For example, in an adder, the addition of lower-level inputs directly affects the outputs at higher levels, as in $1+1=10$ (binary).

It would seem then that an evolutionary model in which individual elements are independently built up would be an equitable compromise between the need to divide the evolutionary process into several rather independent stages and the need for interaction between the different parts of the device. With this in mind, we have developed a method for carrying out the evolution of these simple devices which caters to both needs. Piecewise evolution consists of a series of consequent evolutions of different pieces of a device based on their functionalities, allowing the interactions of the pieces with newly incorporated or mutated elements. New functionalities are acquired from new interactions with the preceding functional groups. Different modules evolve individually in order based on their functionalities, but they are not

physically isolated.

Piecewise evolution consists of dividing the total task of a device up into smaller subtasks which are individually thought to be achievable in a small amount of time. By reducing the complexity of the function in a single evolution and therefore the number of building blocks in the functional group, the possibility of epistasis drops, which further improves the computation time. As an example, we illustrate the evolution of a 2-bit adder.

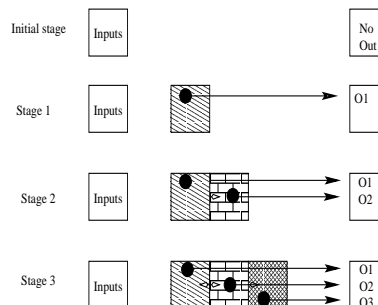


Figure 2.1: Piecewise evolution of a 2-bit adder

In this example, we first partition the total function of a device into four sub-functions. These are the evolution of the correct input nodes (which could be done manually rather than evolved, particularly since it is the easiest part of the evolution), the evolution of the circuitry leading to output one, evolution of the circuitry leading to output two, and the evolution of the circuitry leading to output three. Therefore, the evolution is carried out in four consequent sub-evolutions, or stages during which a new piece of the device is evolved, building upon the previous pieces. The order of the stages is determined by the functions of the pieces evolved in the stage. The more basic pieces, which have greater influence over the functionalities of the other pieces, are evolved first. Only after the adder has the required inputs and outputs can it develop ability to add. Hence, the first stage is to evolve the required inputs and outputs. Less significant bits are evolved first, as they affect the result of the more significant bits, and their circuitry can be utilized in the creation of higher bits.

Occasionally small changes are made to the piece that already satisfies the maximum requirement of the previous stage. Then the new piece is evaluated according to its performance on the function in interest of the stage. Only when the new piece satisfies the

interest of the stage completely, is the fitness of any design pertinent to the next stage taken into consideration. Once a given functionality is discovered in the population, all other members of the population are reevaluated based on a strict need to have this functionality. Other designs which do not have the particular functionality already are effectively removed from the population from this point forward. Moreover, any design which reverts to a state not including the given function will be removed from the population as well. The designs with the newly developed function continue to replicate and mutate, with their fitness values now including the fitness scores corresponding to the next stage of evolution. New nodes and connections are built on the pieces from the previous stage, and the design is evaluated against the standards of the new stage until the plateau of the next stage is reached again.

This model not only allows modules with different functions to evolve individually, but also permits elements in different functional groups to interact freely. As a result, no redundant structures are required, though there is no restriction on their creation. This allows devices which are not linearly separable to be created. At the same time, it incrementally evolves relatively simple functional elements compared to the complex device it is building.

3 Application of Piecewise Evolution

As evolutionary hardware design is still in its infancy, complex devices are still out of reach. We therefore must apply our evolutionary model to rather simple devices. However, the choice of these devices must be driven by the need to demonstrate the ability to develop algorithms which allow for the generation of devices of maximum complexity. We therefore choose as our testbed the one and two bit adders.

Recall that the I/O table for a 1 bit adder without carry is

I_1	I_2	O_1	O_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 3.1: The I/O table of a one bit adder without carry.

This adder has outputs given by

$$O_1 = I_1 I_2 \quad (2)$$

and

$$O_2 = \bar{I}_1 I_2 + I_1 \bar{I}_2 . \quad (3)$$

In our system, these are extremely easy to build, as indicated in Figure 3.1.

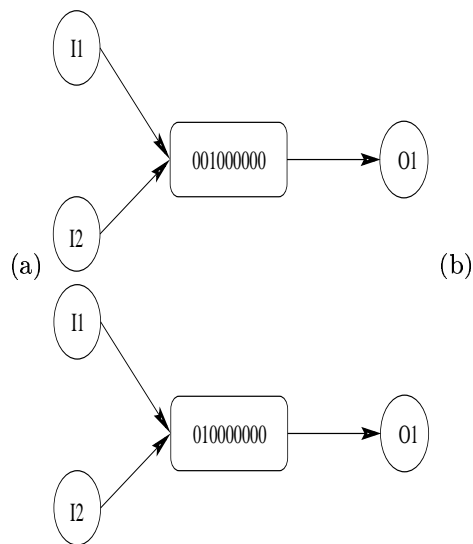


Figure 3.1: This gives the simplest circuit required for the design of these adder output circuits.

Since these can be built out of completely different subcircuits which have no overlap in active states, we can see that the entire circuit is separable. That is, the set of all input states for which the outputs are active have an empty set intersection. This means that the circuit is capable of being built in such a way that either an active state in one output negates that of the other output, or that the two groups are independently built. This is essentially the same as having a compartmentalized evolution, which, as we have seen elsewhere [3],[4] is beneficial. In other words, this design would seem to be able to be written as

$$D = D_1 + D_2 \quad (4)$$

where each factor on the right corresponds to an independent evolutionary step

On the other hand, the I/O table for the two-bit adder is

I_1	I_2	I_3	I_4	O_1	O_2	O_3
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Table 3.2: This is the I/O table for the two bit adder.

This table produces the output equations

$$O_1 = I_1 I_3 + I_4 (I_1 I_2 + I_2 I_3) \quad (5)$$

$$O_2 = (\bar{I}_1 I_3 + I_1 \bar{I}_3) (\bar{I}_2 + \bar{I}_4) + I_2 I_4 (\bar{I}_1 \bar{I}_3 + I_1 I_3) \quad (6)$$

$$O_3 = \bar{I}_2 I_4 + I_2 \bar{I}_4. \quad (7)$$

These equations are considerably more complex and require significantly more effort than the previous steps. Moreover, their implementation is neither simple nor intuitive, and a simple design cannot be made from the devices we are using. The design of each part is given in Figure 3.2.

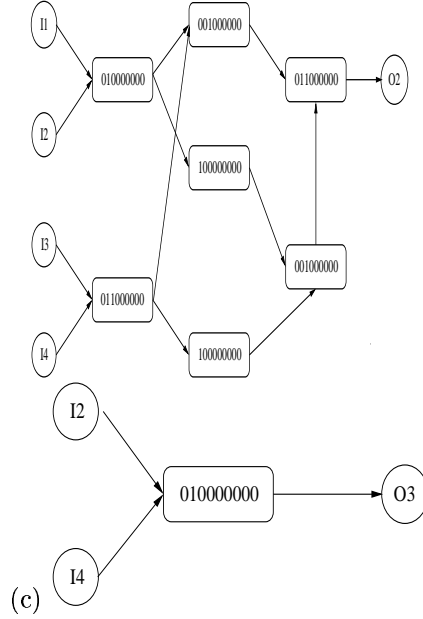
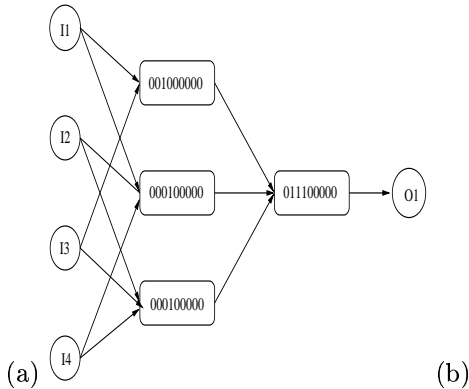


Figure 3.2: The design of a two bit adder.

The difficulty in evolving the design in Figure 3.2 arises from the complexity of each of the steps, and the overlap of each of the outputs with one another. The overlap encourages deceptive evolutionary steps in which beneficial steps are taken by connecting functional parts of one subcircuit to functional parts of another subcircuit. This deceptiveness can lead to epistasis, essentially terminating the evolutionary progression.

In view of these considerations, we may ascertain that the advantages obtained by utilizing piecewise evolution on these two models will be significantly greater in the first case than in the second case. This is because the use of piecewise evolution allows deceptive connections to be formed between subsequent stages. Since the first problem does not have any deceptive connections between phases, the use of piecewise evolution serves to eliminate steps in which disadvantageous steps are attempted, effectively streamlining the evolution. In the second, the deceptive interactions between different steps are minimized, but not eliminated, as deceptiveness occurs between any pairs of outputs. Thus, we expect that the advantage due to piecewise evolution to be limited in the second case.

Indeed this is precisely what is found. In the case of the one bit adder, we have already seen in Figure 2.1 a

particularly efficient adder. One of two adders is typically evolved when this design is undertaken. Typical instantiations are depicted in Figure 3.3. In both cases, six nodes are used. In one case, two identical nodes are used with asymmetric connections, and in the second case, two different nodes are used with symmetric connections.

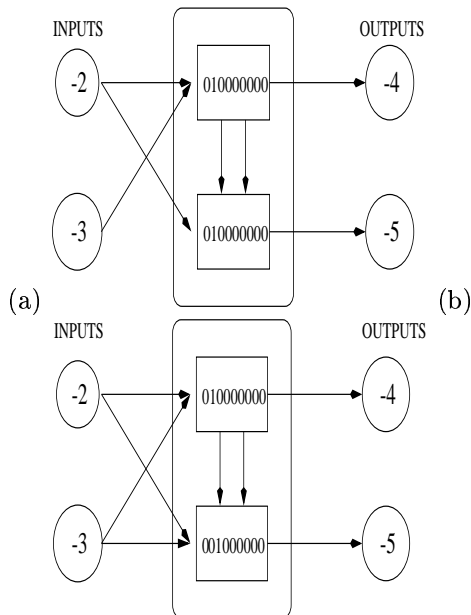


Figure 3.3: Two typical evolved one bit adders. Both have asymmetries, with one in the node, and the other in the connections.

The evolution time of a piecewise adder is 3387.5 ± 328.6 iterations as opposed to the evolution of a non-piecewise one bit adder, which has the evolution time of 4578.5 ± 304.2 . The minimum time of completion of the design for the piecewise evolution was 2562 iterations, while for the non-piecewise evolution, it was 1387 over forty individual runs. Due to the limitations in computational power, the hundreds of runs required to develop an accurate measurement of the probability distribution have not been done, and it is unclear if this minimum represents a significant data point.

An evolved two-bit adder is remarkably compact in view of the previous considerations, as viewed in Figure 3.4. This compactness is possible due to the use of rather complex processors.

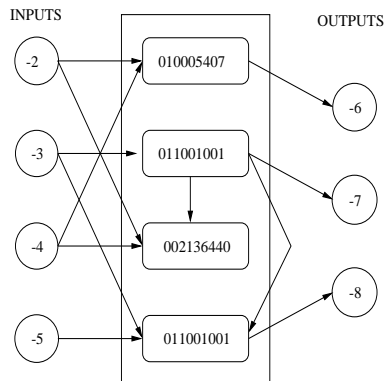


Figure 3.4: An evolved two bit adder.

In this case, our system’s behavior is as we expected. The performance of the piecewise evolution is identical to that of the non-piecewise evolution. The average evolution time of the piecewise evolution is 34675.8 ± 3785.8 iterations, while that for non-piecewise evolution, is about the same. The non-piecewise evolution time is 30538.2 ± 4834.3 , which is statistically identical to the piecewise evolution. This would seem to support our understanding of the system as stated above.

4 Discussion and Conclusions

The advent of new high speed programmable chips [5] allows for the study of evolvable technology in unprecedented ways. This would seem to allow us to create new circuitry *in vivo*, as it were. The advantages are numerous, and include the ability to incorporate the physics as it is, rather than as we think it is. Moreover, the time required for testing individual populations of circuits is currently rather small, and will likely continue to decrease, making this a good test bed for the desired evolution.

Evolvable hardware is advantageous because it would seem to bypass the difficulty incurred when designing rather large circuits of multiple inputs. This would seem to be the case when long design cycles are no longer required for large data sets. Moreover, it is capable of handling fuzzy data sets in ways that standard digital logic cannot, as well as handling analog reconfigurable hardware.

The use of learning circuitry and evolvable circuitry, however, has several undesirable aspects. Of

these, perhaps the most difficult to surmount is the development of complex circuitry even when the input/output relations are completely known. The difficulty, as we have demonstrated, comes about because of the inability, in a particular model, to reduce the system to a simple system of a limited complexity. That is, the development of individual parts of the device cannot be divorced from one another, and this leads to difficulty in the evolution of complex devices.

In this study, we created a system similar to one that has been shown elsewhere to have a significant advantage in the development of simple digital adders [1]. However, the previous system was created in such a way that it consisted of two different steps: synthesis and integration. Of course, in the end, the difficulties clarified here have not been addressed by the use of either phased or piecewise evolution. Rather, individual methods have been developed which bypass the complexity as much as possible. In the end, one would expect this sidestepping approach to fail long before the first microcontroller is designed using evolutionary techniques.

What would seem to be the correct line of future work in this area would be the investigation of viable ways of breaking down complex problems into simple connected parts. The use of these parts significantly reduces the complexity of the evolutionary design task, as has been verified elsewhere. The design of a method of generating functional groups and linkages would provide a framework in which to undertake the evolution, and may significantly improve the evolutionary speed.

The fact that our piecewise evolution failed to produce superior results underscores the assertion that simply breaking up the problem into independent output units does not guarantee improvement. While the details of our study are both model and evolutionary implementation-dependent, the result would seem to be more general than our study. Therefore, it would seem appropriate that we should abandon this approach to generating evolutionary methods in favor of one which allows functional connected subgroups to be built up and evolved within that scaffolding, obeying a compartmentalized backbone. This would seem to be true of both hardware and software systems, with the former perhaps more rigidly requiring this structure.

Appendix

In this model, mutations are restricted to rather simple, one-step mutations. These are as follows:

1. Connection Mutations
 - (a) Add a connection
 - (b) Delete a connection
 - (c) Flip a connection
 - (d) Duplicate a connection
 - (e) Change a connection
2. Node Mutations
 - (a) Add a node
 - (b) Delete a node
 - (c) Duplicate a node
 - (d) Change a node (I/O table)

These mutations occur in groups of 1-5 , with the number of mutations randomly chosen from a uniform probability. No crossover mutations are used, and reproduction occurs without an explicit population-based reproductive event, as in genetic algorithms and genetic programming.

References

- [1] Hounsell B. and Arsian T. *A Novel Evolvable Hardware Framework for the Evolution of High Performance Digital Circuits*. Whitley D., Goldberg D., Cantu-Paz E., Spector L., Parmee I., and Beyer H., eds. **Proceedings of the Genetic and Evolutionary Computation Conference, 2000**, Las Vegas, Nevada, USA, July 10-12, 2000.
- [2] Kazadi S. *Conjugate Schema in Genetic Search*. Back, T., ed. **Proceedings of the Seventh International Conference on Genetic Algorithms**. East Lansing, Michigan, USA, July 19-23, 1997.
- [3] Kazadi S. *Conjugate Schema and Basis Representation of Crossover and Mutation*. **Evolutionary Computation**. 6 (2): 129-160, 1998.

- [4] Kazadi S., Lee D., Modi R., Sy J., and Lue W. *Levels of Compartmentalization in Artificial Life*. **Proceedings of Artificial Life VII**, Bedau, McCaskill, Packard, and Rasmussen, eds. MIT Press, 81-89, 2000.
- [5] Nicholson A. *Evolution and Learning for Digital Circuit Design*. **Proceedings of GECCO 2000**, pp.519-524, 2000.
- [6] Koza J., Bennet III F., Andre D., and Keane M. **Genetic Programming III**. Darwinian Invention and Problem Solving. San Francisco : Morgan Kaufmann Publishers, 1999.
- [7] Yu T. and Miller J. *Neutrality and the Evolvability of Boolean Function Landscape*. **Proceedings of the Fourth European Conference on Genetic Programming**, Lake Como, Italy, April 18-20, 2001.
- [8] Thompson A. and Wasshuber C. *Design of Single Electron Systems through Artificial Evolution*. **Int. J. Circuit Theory and Applications**, 28(6): 585-599, 2000.
- [9] Hornby G., Lipson H., and Pollack J. *Evolution of Generative Design Systems for Modular Physical Robots*. **Proceedings of the IEEE International Conference on Robotics and Automation**, 2001.