Towards Compositional Coevolution in Evolutionary Circuit Design

Michaela Sikulova
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno,
Czech Republic
Email: isikulova@fit.vutbr.cz

Gergely Komjathy
Brno University of Technology
Faculty of Information Technology
Bozetechova 2, 612 66 Brno,
Czech Republic
Email: xkomja00@stud.fit.vutbr.cz

Lukas Sekanina
Brno University of Technology
Faculty of Information Technology
IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno,
Czech Republic
Email: sekanina@fit.vutbr.cz

***Abstract*—A divide and conquer approach is one of the methods introduced to get over the scalability problem of the evolutionary circuit design. A complex circuit is decomposed into modules which are evolved separately and without any interaction. The benefits are in reducing the search space and accelerating the evaluation of candidate circuits. In this paper, the evolution of non-interacting modules is replaced by a coevolutionary algorithm, in which the fitness of a module depends on fitness values of other modules, i.e. the modules are adapted to work together. The proposed method is embedded into Cartesian genetic programming (CGP). The coevolutionary approach was evaluated in the design of a switching image filter which was decomposed into the filtering module and detector module. The filters evolved using the proposed coevolutionary method show a higher quality of filtering in comparison with filters utilizing independently evolved modules. Furthermore, the whole design process was accelerated 1.31 times in comparison with the standard CGP.**

I. Introduction

Evolutionary circuit design is one of possible utilizations of evolvable hardware. First survey papers dealing with evolvable hardware and evolutionary circuit design (e.g. [22] from 1996) clearly identified main advantages of this method: the evolutionary circuit design approach can explore a much wider space of design alternatives than conventional methods or human designers; no a priory knowledge is needed; constrains can easily be handled; and novel designs can, in fact, be discovered in a fully automated manner. Potential problems of the method were also surveyed already in 1996 [22] and the so-called *scalability problem* was mentioned on the first place. Unsurprisingly, it is still considered among the most serious issues in current surveys on evolvable hardware such as [2]. Let us restrict ourselves to digital circuits only in this paper.

The scalability problem is usually understood as a problem in which the evolutionary algorithm (EA) is able to provide a very good solution to a small problem instance; however, only unsatisfactory solutions can be generated for larger problem instances. Because a kind of evolutionary algorithm is employed, the user is supposed to design a suitable problem encoding, search strategy (i.e. search algorithm including genetic operators) and fitness function. All these components can suffer from the scalability problems.

As complex solutions are usually represented by long chromosomes, it is difficult to establish a fast and accurate search method which will be capable of finding good solutions in the corresponding complex search spaces. This problem is referred to as the *scalability of representations* problem. Moreover, in order to evaluate a complex candidate solution, a time consuming fitness function has to be undertaken in which the evaluation time typically grows exponentially with the size of the problem. This difficulty is known as the *scalability of the fitness calculation*.

Hence various approaches have been developed to eliminate the aforementioned scalability problems. All these approaches bring some *knowledge* into the problem which enables us to either reduce the search space, simplify the fitness calculation, or perform a more intelligent search. The most important approaches will be surveyed in the following paragraphs.

A *divide and conquer* approach firstly (either deterministically or heuristically) divides the target circuit into modules (or subcircuits) and then evolves a solution for each module separately [18]. The benefits are twofold: reducing the search space and simplifying the fitness calculation. The approach can be applied iteratively [17].

Functional level evolution allows utilizing complex circuit elements (such as adders, multipliers or comparators) instead of elementary gates [8]. Relatively complex circuits can then be encoded using a shorter chromosome and the search space is thus restricted. The functional level evolution is often combined with a suitable decomposition scheme [13].

Another class of approaches, inspired by the biological development, employs an *indirect problem encoding* [4], [1]. A program is encoded in the chromosome rather than a circuit. When executed, the program is capable of constructing a complete phenotype from an initial solution, which is usually called the embryo. While designing a suitable developmental encoding is tricky, main advantages are shortening the genotype and obtaining a natural support for modularity and scalability in resulting circuits.

The fitness evaluation time can be reduced by using well prepared (i.e. short) training sets, fitness estimation techniques or functional equivalence checking algorithms (see an overview in [20]). All these approaches are usually accompanied by application-specific acceleration techniques such as parallel circuit simulation, phenotype precompilation or parallel implementation.

At the level of the search algorithm, modularization has been introduced for genetic programming (e.g. in the form of automatically defined functions [5]) as well as Cartesian genetic programming [21]. In the framework of *co-evolutionary algorithms*, several methods have been developed to reduce the computational requirements. A survey will be given in Sec. II.

An important factor which influences the choice of a suitable technique is whether the task is to evolve a functional solution using available training data or whether, in addition to functionality, the goal is to minimize the number of components or other objectives. For example, if a system is decomposed into several modules and they are evolved separately, the resulting circuit usually contains multiple instances of the same subcircuit allocated in several modules. The question is whether some EA-based method can be used to minimize the number of gates; or a conventional method has be employed because no EA method is capable of doing so for a complex circuit; or the solution is simply left unoptimized because sufficient resources are available on a chip.

In this paper, we propose to extend the well-known divide and conquer method introduced to evolvable hardware many years ago. In the original version of the method, the implementations of modules are evolved separately. The goal of this paper is to show that when a co-evolution of modules is allowed, the overall evolution time can be reduced. The proposed method will be evaluated in the task of switching image filter design.

The rest of this paper is organized as follows. Sec. II introduces principles of coevolutionary algorithms, Cartesian genetic programming (CGP) and utilization of coevolution in CGP. In Sec. III, a new compositional coevolutionary approach to CGP is presented in the task of evolutionary switching filter design. Sec. IV compares the proposed coevolutionary algorithm with non-coevolutionary CGP and discusses the experimental results. Finally, conclusions are given in Sec. V.

II. Coevolution and Cartesian Genetic Programming

This section introduces the principles of coevolutionary algorithms, Cartesian genetic programming and coevolutionary Cartesian genetic programming.

A. Principles of Coevolutionary Algorithms

Coevolutionary algorithms (CoEAs) are characterized by utilizing more populations of individuals (of the same or different type). Properties of individuals in one population can be changed in response to properties of individuals in other populations and vice versa. Fitness of individuals is then established by interaction with individuals from the other populations. CoEAs are traditionally used to evolve interactive behavior which is difficult to evolve with an EA employing absolute fitness function. The state of the art of coevolutionary algorithms has recently been summarized in [9].

Historically, the terms *cooperative* and *competitive* have been used to classify the domains to which coevolution is often applied. These terms appear from game theory, but they have not been appropriate for classifying problems over which CoEA operates or for algorithms themselves. According to [9], problems are primarily divided into classes based on what constitutes a solution. Two types of problems are mentioned in this context – *test-based* problems and *compositional* problems. A test-based problem is one in which the quality of a potential solution is determined by its performance when interacting with a set of tests. However, this paper deals with the second type – a compositional problem – in which the quality of solution to the problem involves an interaction among many components that together might be thought of as a team or assembly.

Compositional coevolution sprang from cooperative coevolutionary algorithms, wherein the originally stated aim was to attack the problem of evolving complicated objects by explicitly breaking them into parts, evolving these parts separately and then assembling the parts into a working whole [10]. There is a number of successful applications of CoEAs to compositional problems. Perhaps the most studied and nontrivial examples are neuro-evolutionary algorithms, which often involve a separation between components of neural networks to small assemblies of neurons and associated weight synapses, and the topology of a complete network, which functions as a blueprint for how assemblies are put together ([16], [7]).

B. Cartesian Genetic Programming

Cartesian genetic programming (CGP) is a variant of *genetic programming* (GP) that uses a specific encoding of directed acyclic graphs and a mutation-based search. CGP has been successfully employed in many traditional application domains of genetic programming such as symbolic regression, but it has been predominantly applied in evolutionary design and optimization of logic networks.

In standard CGP [6], a candidate program is modelled as a matrix of $n_\mathrm{c} \times n_\mathrm{r}$ (columns $\times$ rows) programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task.

The supported n_a-input node functions are defined in the set Γ. Each node input can be connected to either one of the program primary inputs or the output of a gate placed in previous l columns. The l-back parameter thus constrains where the node inputs can be connected to.

Each node is encoded by $n_\mathrm{a}+1$ genes. One gene is devoted to the node function, the remaining genes are the indexes of the node input connections. Therefore, every individual is encoded using $n_\mathrm{c} \cdot n_\mathrm{r} \cdot (n_\mathrm{a}+1) + n_\mathrm{o}$ integers. Fig. 1 shows an example of a candidate program and its encoding in the chromosome.

A simple $(1+\lambda)$ evolutionary strategy is used as a search mechanism in CGP. The initial population is constructed randomly, by a heuristic procedure or seeded by existing solutions. Every new population consists of the best individual of the previous generation (so-called parent) and its λ offspring. A new parent is selected as an offspring with equal or better fitness than the previous parent. In order to create the offspring individuals from the parent, a point mutation operator is used,

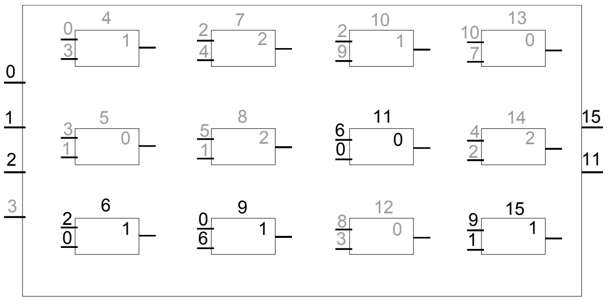


Fig. 1: A candidate program in CGP, where $n_c = 4$, $n_r = 3$, $l = n_c$, $n_i = 4$, $n_o = 2$, $n_a = 2$, Γ = {NOT (0), AND (1), OR (2)} and the chromosome is: $0, 3, \mathit{1}$; $3, 1, \mathit{0}$; $2, 0, \mathit{1}$; $2, 4, \mathit{2}$; $5, 1, \mathit{2}$; $0, 6, \mathit{1}$; $2, 9, \mathit{1}$; $6, 0, \mathit{0}$; $8, 3, \mathit{0}$; $10, 7, \mathit{0}$; $4, 2, \mathit{2}$; $9, 1, \mathit{1}$; $15, 11$.

which modifies h randomly selected genes to new randomly generated, but valid values. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

C. Coevolution in Cartesian Genetic Programming

The design process entails finding useful and productive ways of mapping problem particularities into the algorithmic framework. CGP has some important differences compared to genetic algorithms (GAs) and standard genetic programming. CGP uses very small populations (usually $1 + 4$ individuals) which implies that many generations have to be performed (compared to GAs and GP) to obtain a solution. This feature is the main aspect to consider while designing the coevolutionary interactions in CGP.

In our previous work, inspired by the principles of Hillis' *competitive coevolution* [3] and *coevolution of fitness predictors* introduced by Schmidt and Lipson [11], we proposed a coevolutionary CGP for *test-based problems* [15]. The generic coevolutionary principles have been adapted for CGP and a significant improvement (in terms of computational cost reduction) in comparison with CGP without coevolution has been obtained in 5 symbolic regression tasks [15] and in the task of evolutionary image filter design [14].

III. Proposed method

A. Initial Considerations

Successful design and application of any heuristic depends on a number of key representation decisions of how search knowledge is encoded and manipulated. When designing digital circuits using the compositional coevolution, the target circuit has to be divided into (sufficiently small) logical modules. Each of them is a single entity (*individual*) that has to be represented in terms of chosen evolutionary algorithm. In this work, the modules will be represented as Cartesian programs. Furthermore, individuals representing the same module form the *population*. The number of populations in compositional coevolution is equal to the number of modules in the target circuit.

In coevolutionary methods, there is another type of collection, typically referred to as an *archive*. It is a collection of individuals (of the same or different type) that spans multiple generations of a coevolutionary algorithm [9]. In CoAEs, the individual fitness is evaluated using interaction with other individuals. As individuals in each population are changed in dependence on other individual changes, the best obtained individuals are not necessarily present in the last generation. Archives are thus utilized as a kind of search memory and usually contain the final solution. In the course of evolution, the individuals showing the best-scored composition with other modules are added to the archive. Another approach employs an archive of individuals for each module, i.e. the number of archives is equal to the number of populations.

When using multiple populations, a strategy of *interactions* between individuals in the same population, or especially in different populations, has to be defined. The simplest choice is to enable all possible interactions (i.e. *complete mixing*). However, in order to reduce the computational cost, only some of all possible interactions are allowed and assessed.

In the case of CGP, where only one parent is selected in every generation, we propose to evaluate individuals of a given population using the top-ranked individual (module) from other populations. Our recent work dealing with the *test-based* type of coevolution had shown that frequent interactions between populations do not lead to correctly working solutions, because of very fast changes in the fitness calculation procedures [15]. As CGP uses small populations, the search process needs more generations to adapt to the changes of the selective pressure. We propose to add to each population a new archive containing the top-ranked individual (resp. individuals) available to evaluate individuals from other populations. Individuals stored in the archives are updated according to the user-defined policy (e.g. after some number of generations is reached) whose setting influences the CGP search progress.

B. Case Study – Coevolution of Image Filters and Noise Detectors

The principles of compositional coevolution will be applied in the image filter design task which will be carried out by coevolutionary CGP. The goal is to eliminate the so-called impulse noise (salt-and-pepper noise) which is a basic type of non-linear noise typically affecting a single pixel in different regions of the image. A conventional solution is usually based

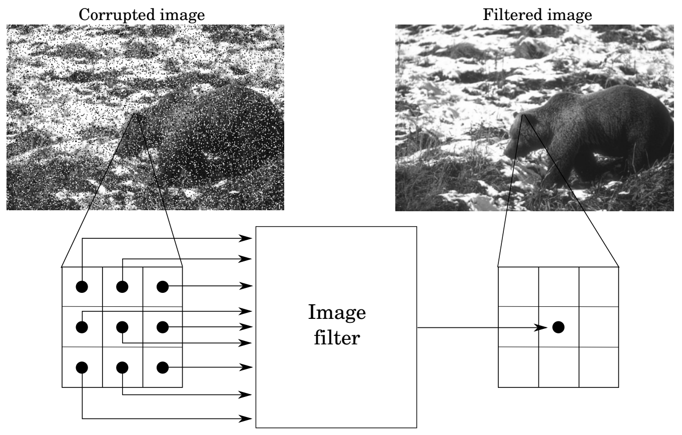


Fig. 2: An image filter with the 3×3 pixel window.

#	function	#	function
0	255	8	$i_1 \gg 1$
1	i_1	9	$i_1 \gg 2$
2	$\overline{i_2}$	10	$(i_1 \ll 4) \vee (i_2 \gg 4)$
3	$i_1 \vee i_2$	11	$i_1 + i_2$
4	$\overline{i_1} \vee i_2$	12	$i_1 +^s i_2$
5	$i_1 \wedge i_2$	13	$(i_1 + i_2) \gg 1$
6	$\overline{i_1 \wedge i_2}$	14	$\max(i_1, i_2)$
7	$i_1 \oplus i_2$	15	$\min(i_1, i_2)$

TABLE I: CGP node functions according to [12].

on median filters. Various approaches to evolutionary design of this type of filters by means of the standard CGP have been surveyed in [12].

As the considered filters evolved using CGP operate over a filtering window (the so-called kernel) consisting of 3×3 pixels, each candidate filter can utilize up to nine 8-bit inputs, i.e. $n_i = 9$. The filters produce a single pixel, i.e. $n_o = 1$. Table I gives a set of functions working over two pixels i_1 and i_2 that are typically used for image filter evolution. Fig. 2 shows an overall scheme of the method.

In the fitness function, the goal is to maximize the peak signal-to-noise ratio (PSNR) between an incorrupted version of the training image w and the resulting image (v) produced by a candidate filter. The PSNR is defined as

$$PSNR = 10 \cdot log_{10} \frac{255^2}{\frac{1}{MN}\sum_{i,j}(v(i,j) - w(i,j))^2}, \quad (1)$$

where $M \times N$ is the size of the image.

However, for higher noise intensities or arbitrary values of noisy pixels (in the range 0 – 255), these simple approaches usually lead to introducing additional noisy pixels, smudging and detail loosing in resulting images. Hence advanced filtering techniques have been developed including the concept of noise detection and iterative filtering.

In [19], the image filtering approach based on CGP has been extended in such a way that an image filter is evolved together with a noise detector. The image filter and detector are encoded in a single CGP grid of nodes, which produces two outputs. The first one is the filtered pixel value and the second one is a Boolean value representing the output of the detector. If the detector produces log. 1 then the first output is taken and utilized as the result of filtering. Otherwise, if the detector produces log. 0, it is indicated that the central pixel of the kernel is not corrupted and the result of filtering is the unmodified central pixel of the kernel. The resulting filter is called the switching filter (see Fig. 3).

1) Decomposition and Representation: Switching filters are good candidates for decompostion. In this task, the whole circuit is decomposed into two modules: (1) the *image filter* eliminating the noise; and (2) the *noise detector*. Fig. 3 shows the overall scheme of the target circuit. Each module is represented using the CGP representation (see Sec. II-C) and uses a grid of 8×5 of nodes performing functions given in Table I. Each individual from the filter population has one 8-bit output, where the filtered value $(0-255)$ of the pixel is expected. While the detector also outputs an 8-bit value, only the most significant bit is utilized to control the switch – shown in Fig. 3.

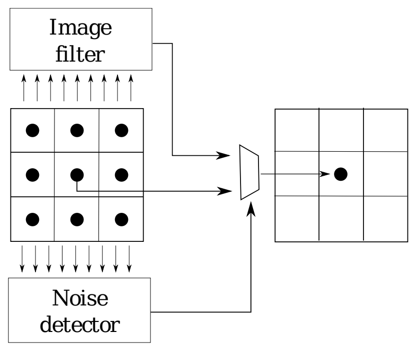


Fig. 3: Decomposition to the filter module and the noise detector module.

2) Interaction of Populations and Fitness Evaluations: At first the *complete mixing* approach has been employed. The interactions of individuals in the population of filters with each individual in the population of detectors (and vice-versa) has been allowed in every generation. All pixels (and their neighbourhoods) in corrupted image were executed by a pair of filter and detector. The resulting filtered image (as shown in Fig. 3) was compared to the uncorrupted image in terms of PSNR. The most successful pair was selected to produce a filter offspring and a detector offspring and a new iteration of the search process has been initiated. However, this type of interaction has led to filters with poor quality of filtering in comparison with filters without detectors evolved for the same number of generations. Moreover, the cost of evaluation has exponentially grown.

Inspired by our previous work on *test-based* coevolution in CGP, another approach to interaction has been employed. Filters and detectors are evolved separately for some number of generations. Therefore, three archives have been introduced: (1) the archive of filters ($\mathrm{A}_{\mathrm{filters}}$) with filtering results (i.e. filtered images); (2) the archive of detectors ($\mathrm{A}_{\mathrm{detectors}}$) with the results of detection (i.e. images with pixels classified as corrupted); and (3) the archive of top-ranked target circuits ($\mathrm{A}_{\mathrm{targets}}$) with the most successful composed pairs.

Two fitness values are calculated for each individual. The *first fitness of filter* ($\mathrm{fitness}_f$) is evaluated in terms of PSNR between the image filtered using the filter (without detector) and the reference image (according to scheme in Fig. 2). The *second fitness of filter* is established using the composition with the detector actually present in $\mathrm{A}_{\mathrm{detectors}}$ ($\mathrm{fitness}_{f+d}$, in terms of PSNR). Then the individual with the best $\mathrm{fitness}_f$ is selected as a parent. If more individuals have the same $\mathrm{fitness}_f$ value then another criterion is taken into account – the individual with the best $\mathrm{fitness}_{f+d}$ is selected.

Similarly, the *first fitness of detector* ($\mathrm{fitness}_d$) is evaluated as the *score* of correctly classified pixels:

$$\mathrm{fitness}_d = \sum_{i=1}^{j} \mathrm{f}\left(y\left(i\right)\right), \text{ where} \quad (2)$$

$$\mathrm{f}\left(y\left(i\right)\right) = \begin{cases} 0 & \text{if pixel is incorrectly classified} \\ 1 & \text{if pixel is correctly classified} \end{cases} \quad (3)$$

where $y(i)$ is the output of detector y for pixel i. The *second fitness of detector* is established using the filter which is

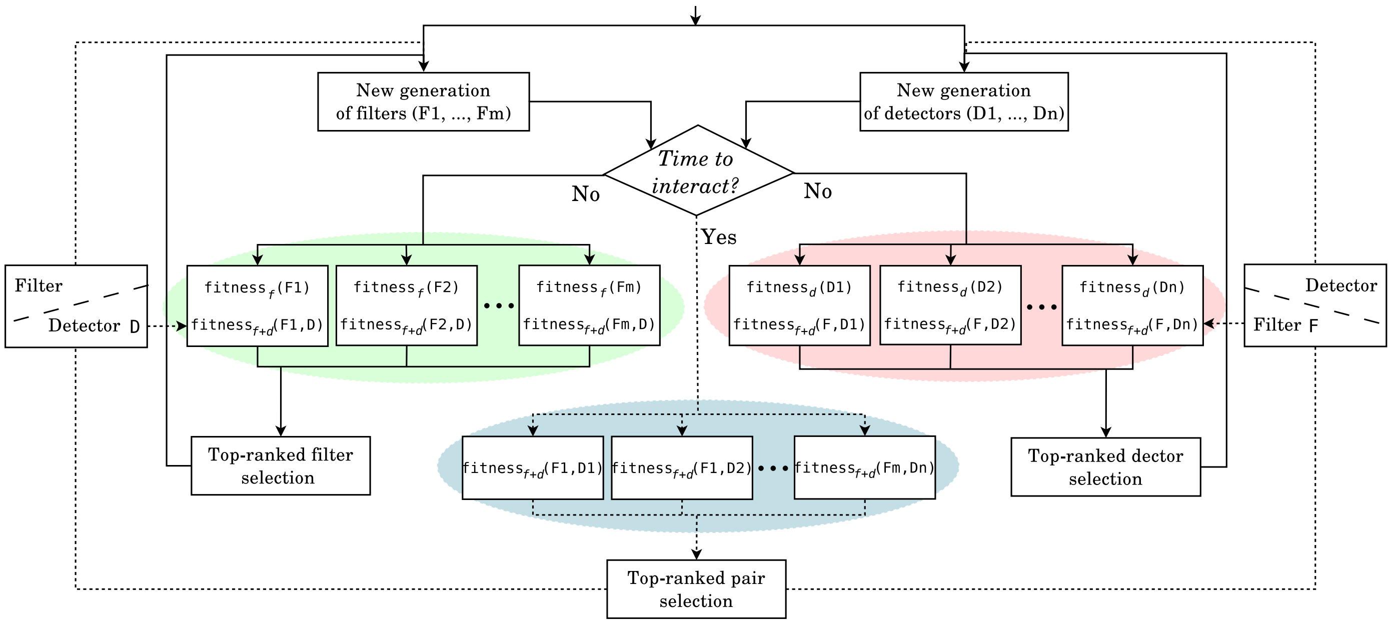


Fig. 4: The overall coevolutinary interaction scheme.

actually present in the archive of filters ($\mathrm{fitness}_{f+d}$, in terms of PSNR). If more individuals have the same $\mathrm{fitness}_d$ value then the individual with the best $\mathrm{fitness}_{f+d}$ is selected.

The evaluation of filters and detectors is carried out simultaneously. After producing a given number of generations, the global coevolutionary interaction is accomplished. Each individual from the current generation in the filter population is evaluated in composition with each detector from the current generation in the detector population in terms of PSNR ($\mathrm{fitness}_{f+d}$). If more pairs achieve the best score $\mathrm{fitness}_{f+d}$, the pair with the least number of nodes (in sum of filter and detector), in order to minimize the number of nodes, is selected as the best pair.

If the best pair has better $\mathrm{fitness}_{f+d}$, then it replaces the previous pair in $\mathrm{A}_{\mathrm{targets}}$; in the case of identical score, the number of used nodes is considered. The filter module of the best pair is sent to $\mathrm{A}_{\mathrm{filters}}$ (where is used to evaluate detectors) and becames the parent of a new generation in filter evolution. The detector module is sent to $\mathrm{A}_{\mathrm{detectors}}$ (where is used to evaluate filters) and becames the parent of a new generation in detector evolution. The overall scheme of interactions is shown in Fig. 4.

After producing a predefined number of generations, the target circuit can be composed of the best-evolved individuals from each population or it can be found in the archive of target circuits.

The evaluation of each module may require training data differing from training data used to evaluate the whole circuit, because each module has performed just a part of the whole task. When filters and detectors are evolved separately, some pixel evaluations are usually unnecessary if the whole training image is used. Training data for the filter module can be composed of corrupted pixels only, which can significantly reduce the computational cost of filter evaluation. This property is also considered in our experiments.

IV. Experimental Results

This section presents benchmark problems, experimental setup and experimental evaluation of the proposed coevolutionary approach and its comparison with non-coevolutionary CGP.

A. *Test Scenarios*

In order to evaluate the proposed approach, five scenarios and thus five types of salt-and-pepper noise filters were designed using CGP:

F1: filter evolved using standard CGP and all pixels in training image (no detector);

F2: filter and detector evolved separately, filter evolved using just corrupted pixels;

F3: filter and detector evolved separately, filter evolved using all pixels in training image;

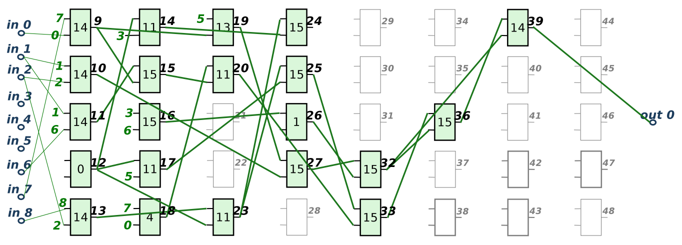
F4: coevolved filter with detector, filter evolved using just corrupted pixels;

F5: coevolved filter with detector, filter evolved using all pixels in training image.

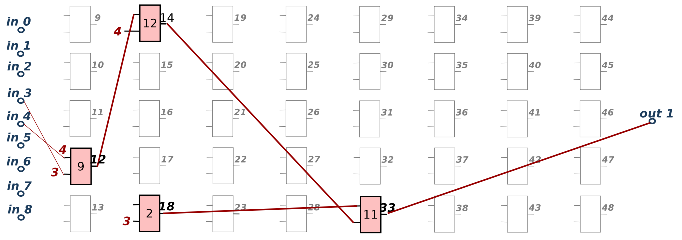
The salt-and-pepper noise is characterized by noisy pixels with the value of either 0 or 255 (for 8-bit gray-scaled images). In order to obtain training images, the Lena image with the size of 256×256 pixels was corrupted by 10%, 20%, 30%, 40% and 50% salt-and-pepper noise. The evolved filters were tested on 6 different images with the size of 512×512 pixels containing the same type of noise.

B. *Experimental setup*

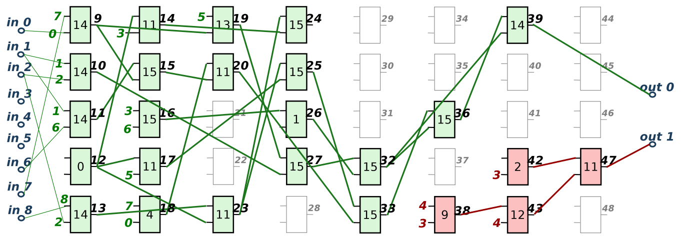
The experimental setup of CGP for filters as well as detectors is as follows: $n_c = 8$, $n_r = 5$, $l = 8$, $n_i = 9$,



(a) Image filter.



(b) Noise detector.



(c) Composed image filter with noise detector.

Fig. 5: An example of coevolved image filter and noise detector and user-defined composition into one grid. Node functions are numbered according to Table I.

$n_o = 1$, $\lambda = 3$, every node has two inputs, the number of mutations per new individual is $h = 6$ and Γ contains the functions from Table I.

The initial populations are randomly seeded. The archive of noise detectors is initialized with a detector containing a constant outputting circuit (255) in order to indicate all pixels as corrupted. The archive of image filters is initialized with the best filter of the first generation of the filter evolution.

The global interaction in the compositional coevolutionary design is activated every 100 generations of the filter evolution. For every scenario, the evolution/coevolution is terminated after 100,000 generations of the filter evolution. When detectors are evolved separately, 100,000 generations of the evolution of detectors are performed. Each experiment is repeated 30 times.

C. The Size of Evolved Filters

It can be seen in Table II that filters utilizing a detector need five additional functional blocks on average regardless of design approach. Detectors contain from 2 to 11 blocks. Examples of a coevolved filter, detector and their composition into one circuit are displayed in Fig. 5.

Filter type	Noise intensity				
	10 %	20 %	30 %	40 %	50 %
	Number of used blocks				
F1	12.9	14.2	15.1	15.1	15.7
F2 + F3	13.9+5.1 (19)	13.9+5.5 (19.4)	13.9+6.3 (20.2)	13.3+7.1 (20.4)	13.9+6.3 (20.2)
F4 + F5	14.4+4.8 (19.2)	14.1+5.1 (19.2)	14.3+5.7 (20.0)	14.4+6.1 (20.5)	14.2+6.1 (20.3)

TABLE II: The mean number of used blocks (by filter + detector).

Filter type	Noise intensity				
	10 %	20 %	30 %	40 %	50 %
	Time $\times 10^3$ [s]				
F1	10.6	10.8	10.7	11.0	**10.6**
F2	9.2	9.9	11.8	11.9	12.0
F3	16.0	16.2	16.6	16.4	16.5
F4	**6.0**	**6.7**	**7.4**	**9.1**	11.8
F5	22.8	23.0	23.4	24.1	24.1

TABLE III: Mean time to execute 100,000 generations (performed on the Intel Core i7-3632QM).

D. The Time of Evolutionary Design

The scenarios F1 – F5 were implemented using OpenMP instructions and four individuals were evaluated simultaneously ($2 + 2$ while coevolving filters and detectors). In the case of coevolution, two evolutionary processes were running in parallel and synchronized for interactions.

The time of evolution is mainly affected by the size of training data (i.e. the number of pixels in the training image) and by the number of functional nodes that candidate filters utilize. Table III shows the mean time to execute 100,000 generations for scenarios F1 – F5. The time required for filters with separately evolved detectors (F2, F3) is assigned as the sum of 100,000 generations of the fitler evolution and 100,000 generations of the detector evolution.

The most time-consuming design process is the coevolution of switching filters in which filters are evaluated using all corrupted and incorrupted pixels (F5). Note that independent evolutionary runs of filters and detectors have not engaged the interactions and communinacation. Furthermore, due to less number of used blocks in detectors than in filters, the detector evolution runs faster and executes more generations than the evolution of filters.

Employing just corrupted pixels for evaluation of filters reduced the mean time of evolution for images with lower noise intensities. In this case, the coevolutionary design (F4) outperforms scenario F1, because it uses less training pixels. It also outperforms scenario F2, because the evolution of filters runs faster than the evolution of detectors – therefore evolution of detectors executes less generations than the evolution of filters.

E. The Quality of Evolved Filters

The visual quality of evolved filters has been expressed in terms of PSNR. Fig. 6a shows the progress of fitness of the switching filter during one run of the independent evolution of filter and detector components (scenario F2). Fig. 6b shows that the coevolutionary approach (scenario F4) allows the

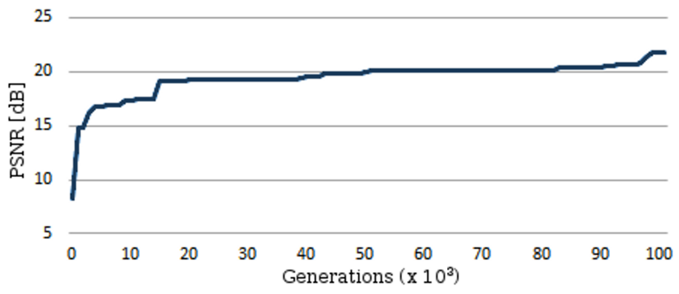
Noise intensity		Filter type						
		F1	F2	F3	F4	F4-A	F5	F5-A
		PSNR [dB]						
10 %	Min	15.4	17.8	17.0	15.3	**20.1**	15.1	18.2
	Mean	31.2	33.0	32.8	32.9	**33.2**	31.0	31.7
	Max	36.9	38.7	37.6	38.4	**38.9**	35.6	37.2
20 %	Min	**18.9**	15.4	14.5	14.6	17.9	13.9	17.9
	Mean	25.4	26.2	25.9	26.1	**26.5**	25.3	26.2
	Max	30.7	31.2	30.8	31.1	**31.5**	30.6	30.6
30 %	Min	17.7	13.1	12.6	13.7	**18.8**	13.0	18.2
	Mean	22.3	**24.9**	24.7	24.8	**24.9**	23.7	24.0
	Max	27.8	**28.6**	28.5	28.0	28.0	28.1	28.2
40 %	Min	16.9	12.0	12.2	11.7	17.4	13.3	**18.4**
	Mean	20.0	20.1	20.6	20.8	**21.9**	21.5	**21.9**
	Max	25.5	**27.7**	27.2	26.8	26.9	26.6	26.6
50 %	Min	16.2	11.0	11.5	15.3	18.6	14.4	**18.8**
	Mean	19.1	19.7	19.7	20.6	**20.9**	19.8	20.6
	Max	23.1	24.6	24.5	24.2	**24.7**	24.1	24.2

(a) Training image.

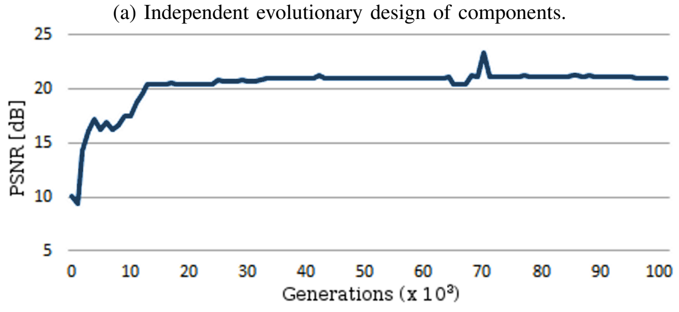
Noise intensity		Filter type						
		F1	F2	F3	F4	F4-A	F5	F5-A
		PSNR [dB]						
10 %	Min	15.0	17.0	16.5	13.7	**18.5**	14.4	18.4
	Mean	29.3	32.6	32.1	32.9	**33.0**	29.3	30.7
	Max	35.8	**42.2**	40.0	40.5	40.8	37.4	38.5
20 %	Min	**17.5**	11.0	13.0	11.5	15.3	13.7	15.5
	Mean	24.3	25.1	24.7	24.1	**25.9**	24.0	24.4
	Max	29.1	32.6	30.9	33.1	**34.5**	34.0	34.3
30 %	Min	**16.0**	9.3	9.5	11.2	11.5	10.8	12.1
	Mean	20.1	21.6	21.1	22.9	**23.2**	22.7	22.9
	Max	25.2	28.5	28.0	29.5	**29.8**	27.8	28.6
40 %	Min	**15.7**	8.1	9.1	9.0	9.6	9.9	10.8
	Mean	18.5	17.3	17.2	18.2	**19.1**	18.2	18.8
	Max	23.9	**28.6**	26.6	28.1	28.2	27.9	27.9
50 %	Min	**14.1**	7.3	8.3	8.5	8.6	9.4	9.5
	Mean	17.9	16.7	17.3	17.8	**18.9**	17.7	18.3
	Max	22.5	24.6	24.7	25.2	25.4	25.1	**25.9**

(b) Test images.

TABLE IV: The PSNR values of evolved filters calculated from 30 independent runs for each filter type and each noise intensity.



(a) Independent evolutionary design of components.



(b) Coevolution of components.

Fig. 6: Fitness of a switching image filter during one run of the evolution.

fitness value to be decreased – this feature is caused by the fact that the fitness depends on the interaction with other individuals and the current population does not contain the best individual.

Table IV shows the minimum, mean and maximum PSNR values of filtered images using scenarios F1 – F5. Table IV also shows the importance of using the $\mathrm{A}_{\mathrm{targets}}$ archive as a memory of the best-coevolved target circuit. In columns F4 and F5 there are PSNR values for switching filters which are composed of the best individuals of the last generations of coevolution. Columns F4-A and F5-A are devoted to switching filters situated in $\mathrm{A}_{\mathrm{targets}}$ in the end of coevolution.

In the case of coevolutionary design, switching filters composed of filters evaluated using just corrupted pixels (scenario F4) show better quality of filtering than filters evaluated using all corrupted and incorrupted pixels (scenario F5). The difference is approximately 0.6 dB for training image, 1.0 dB for test images. However, the difference is negligable for independent evolutionary design of filters and detectors (0.04 dB for training image, 0.18 dB for test images).

It can be seen that coevolved switching filters (F4-A) show comparable or even better mean PSNR in comparison to filters without a detector (F1, 1.9 dB for training image, 2.0 dB for test images) as well as to switching filters where filters and detectors are designed independently (F2, 0.7 dB for training image, 1.3 dB for test images).

Fig. 7 shows the visual quality of different test images processed using the top-ranked coevolved switching filters.

V. Conclusion

In this paper, a coevolutionary approach to the divide and conquer method has been presented. Interactions of modules of the target circuit have been allowed during the search process, whereas the previous approach evolved the modules separately. The composed resulting circuit then contains modules that are adapted to work together. We have shown that the compositional coevolution can improve the evolutionary design conducted by CGP. In the task of evolutionary salt-and-pepper noise filter design we obtained switching filters with higher quality of filtering (PSNR) in comparison with independent evolutionary design of filters and detectors – in average 0.7 dB for training image and 1.3 dB for test images. Furthermore, using the coevolutionary approach we obtained a speedup 1.31 times in comparison with evolutionary design of image filters without detectors.

Our future work will be devoted to the design of more complex circuits using compositional coevolutionary CGP. As the evolutionary digital circuit design using CGP has been accelerated in FPGA, our goal will be to simplify the interaction between components during coevolution in order to implement the compositional coevolution to FPGA and thus accelerate the search process.

ACKNOWLEDGMENT

This work was supported by the Czech science foundation project 14-04197S, the BUT project FIT-S-14-2297 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

REFERENCES

[1] T. G. W. Gordon and P. J. Bentley, "Towards development in evolvable hardware," in *Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*. IEEE Computer Society Press, 2002, pp. 241–250.

[2] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 183–215, 2011.

[3] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D*, vol. 42, no. 1, pp. 228–234, 1990.

[4] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

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

[6] J. F. Miller, Ed., *Cartesian Genetic Programming*, ser. Natural Computing Series. Springer Verlag, 2011.

[7] G. A. Monroy, K. O. Stanley, and R. Miikkulainen, "Coevolution of neural networks using a layered pareto archive," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '06. New York, NY, USA: ACM, 2006, pp. 329–336.

[8] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi, "Evolvable Hardware at Function Level," in *Parallel Problem Solving from Nature PPSN IV*, ser. LNCS, vol. 1141. Springer, 1996, pp. 62–71.

[9] E. Popovici, A. Bucci, R. Wiegand, and E. De Jong, "Coevolutionary principles," in *Handbook of Natural Computing*. Springer Berlin Heidelberg, 2012, pp. 987–1033.

[10] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary computation*, vol. 8, no. 1, pp. 1–29, 2000.

[11] M. D. Schmidt and H. Lipson, "Coevolution of Fitness Predictors," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 736–749, 2008.

[12] L. Sekanina, S. L. Harding, W. Banzhaf, and T. Kowaliw, "Image processing and cgp," in *Cartesian Genetic Programming*. Springer Berlin Heidelberg, 2011, pp. 181–215.

[13] A. P. Shanthi and R. Parthasarathi, "Practical and scalable evolution of digital circuits," *Applied Soft Computing*, vol. 9, no. 2, pp. 618–624, 2009.

[14] M. Sikulova and L. Sekanina, "Acceleration of evolutionary image filter design using coevolution in cartesian gp," in *Parallel Problem Solving from Nature - PPSN XII*, ser. LNCS 7491. Springer Verlag, 2012, pp. 163–172.

[15] ——, "Coevolution in cartesian genetic programming," in *Genetic Programming*, ser. LNCS 7244. Springer Verlag, 2012, pp. 182–193.

[16] K. O. Stanley and R. P. Miikkulainen, *Efficient evolution of neural networks through complexification*. Citeseer, 2004.

[17] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for evolvable hardware," *IEEE Transaction Systems, Man and Cybernetics, Part B*, vol. 36, no. 5, pp. 1024–1043, 2006.

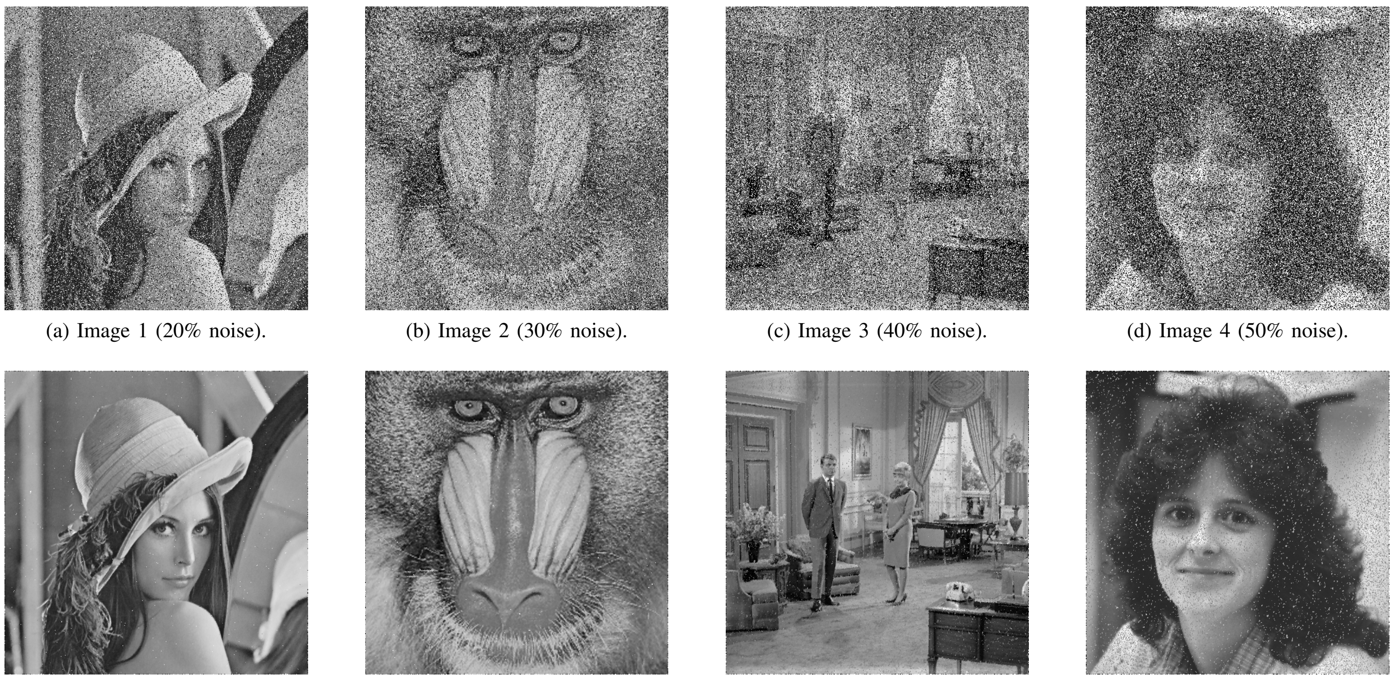
[18] J. Torresen, "A Divide-and-Conquer Approach to Evolvable Hardware," in *Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98*, ser. LNCS, vol. 1478. Springer, 1998, pp. 57–65.

[19] Z. Vasicek, M. Bidlo, and L. Sekanina, "Evolution of efficient real-time non-linear image filters for fpgas," *Soft Computing*, vol. 17, no. 11, pp. 2163–2180, 2013.

[20] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 305–327, 2011.

[21] J. A. Walker and J. F. Miller, "The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 4, pp. 397–417, 2008.

[22] X. Yao and T. Higuchi, "Promises and Challenges of Evolvable Hardware," in *First International Conference on Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 1259. Springer, 1996, pp. 55–78.


(a) Image 1 (20% noise). (b) Image 2 (30% noise). (c) Image 3 (40% noise). (d) Image 4 (50% noise).

(e) Filtered image 1. (f) Filtered image 2. (g) Filtered image 3. (h) Filtered image 4.

Fig. 7: The test images filtered using the best coevolved switching filters (one step correction).