Original scientific paper

Informacije MIDEM
Journal of Microelectronics,
Electronic Components and Materials
Vol. 48, No. 1(2018), 29 – 40

Analog Circuit Topology Representation for Automated Synthesis and Optimization

Žiga Rojec, Jernej Olenšek, Iztok Fajfar

University of Ljubljana, Faculty of Electrical engineering, Ljubljana, Slovenia

Abstract: For several decades, computers have helped analog designers with circuit simulation and evaluation. To further simplify and speed-up designer's work, novel methods are being introduced that help to fine-tune numerical parameters to meet the performance criteria. With a lack of capable engineers, a shortage of specific knowledge or time to design an analog building block, software for fully automated synthesis of both topology and parameters is becoming crucial. Most research in this field is based on circuit modifications according to evolutionary principles of survival of the fittest. One of the challenges of the design of appropriate software is a representation of a circuit topology that will allow topology modifications with the smallest possible computational effort. Many existing solutions suffer either from the uncontrolled growth of the size of the circuit (so-called bloat) or from the limitation of the topology structure to a set of predefined blocks. In this paper, we discuss an analog circuit topology representation in a form of a binary upper-triangular matrix that is both bloat safe and offers a large solution space. We describe the basic structure of the matrix, the redundancy phenomena of logical elements, and the translation of the matrix representation to a regular SPICE netlist. We use an evolutionary algorithm to evolve the topology matrix and a classical parameter optimization algorithm to tune the circuit parameters. Based on a high-level circuit definition and a fixed building-block bank, our topology representation technique showed success in a fully automatic synthesis of passive circuits. We demonstrate the ability to automatically discover a passive high-pass filter topology.

Keywords: Automated synthesis, analog circuits, computer-aided design, evolutionary algorithms

Zapis topologije analognega električnega vezja za namen avtomatske sinteze in optimizacije

Izvleček: V procesu načrtovanja analognih električnih vezij računalniki že desetletja sodelujejo kot orodje za simulacijo ter evalvacijo. V pomoč pri delu razvijalca so že sedaj na voljo orodja, ki so sposobna avtomatično optimizirati numerične parametre vezja in s tem doseči določene kriterije delovanja. Zaradi pomanjkanja inženirjev, znanja in časa za razvoj analognih sklopov je smiselno razmišljati o programski opremi, ki bi bila zmožna ne samo izbrati primerne parametre za doseganje zahtevanih lastnosti temveč tudi sestaviti ustrezno topologijo. Večina dosedanjega dela na tem področju temelji na spreminjanju posameznih delov topologij po evolucijskih principih. Eden od glavnih izzivov pri razvoju tovrstnega orodja je računalniška predstavitev topologije vezja na način, ki bo omogočal računsko čim manj zahtevno spreminjanje topologije. Ena od slabosti nekaterih obstoječih rešitev je velika možnost nekontrolirane rasti sheme vezja preko vseh meja med iskanjem rešitve (t.i. napihovanje, angl. bloat), druga pogosta pomanjkljivost pa je vnaprejšnja omejitev strukture topologije. V tem članku predlagamo zapis predstavitve topologije analognega električnega vezja v obliki binarne zgornje trikotne matrike, ki omogoča ogromen iskalni prostor, hkrati pa zagotavlja imunost pred razlezenjem med postopkom iskanja. Opisujemo osnovno strukturo primernega matričnega zapisa, fenomen redundance logičnih elementov ter razložimo pretvorbo matrike v standarden zapis vezja (angl. netlist) primeren za obdelavo v simulatorju SPICE. Matriko nato spreminjamo s pomočjo posebnega evolucijskega algoritma, številske parametre vezja pa z eno od obstoječih metod za numerično optimizacijo. Primernost zapisa za popolnoma avtomatično sintezo analognega vezja smo preizkusili na primeru razvoja pasivnih vezij. Na podlagi visokonivojske zahteve ter vnaprej znane knjižnice možnih električnih elementov je algoritem sestavil pasivni visokoprepustni filter.

Ključne besede: Avtomatska sinteza, analogna vezja, računalniško-podprto načrtovanje, evolucijski algoritmi

** Corresponding Author's e-mail: ziga.rojec@fe.uni-lj.si*

1 Introduction

The designing of an analog circuit is a demanding task even for a skilled analog designer. Due to constantly increasing time-pressure, lack of experienced engineers and growing industry needs, designers more and more often use computers to support the design process.

© MIDEM Society

Computers and dedicated software have been used to aid the circuit designers since the introduction of SPICE [1]. Soon, designers started to use various mathematical methods to optimize circuit parameters to reach or even overcome the desired performance (e.g., [2], [3], [4], [5]). However, the optimization of the parameters alone is often not enough to meet the required objectives. In that case, a designer needs to rearrange the topology, which means that the parameters have to be optimized again. The recent advances in the field of analog circuit computer-aided design have to do with the combined automatic parameter optimization and the topology calculation of a desired circuit [6].

Majority of the topology search methods use some kind of evolutionary computation, and some early examples of the approach are IDAC [7], OASYS [8], OPASYN [9] and DARWIN [10]. Those early approaches were based on a random selection of topology parts from a predefined library. Consequently, the topology structure was fixed in advance, which seriously limited the size of the solution space. However, the invention of genetic programming (GP) by Koza et al. [11] has mainly removed this limitation and opened door for the first serious attempts in automated topology design. GP is an idea of automated development of a computer program using an evolutionary algorithm. Each program is presented by a tree-like structure, where branches and leaves represent various computer instructions. Koza already proposed this method for automated analog circuit synthesis, where a computer program was built using instructions for setting up a circuit topology [12]. One of the main problems of GP is so-called *bloat*, a phenomenon of an uncontrolled growing of a program tree. Lohn and Colombano [13] proposed a linearization of the circuit-building instructions, which inherited both advantages and disadvantages of standard GP. A binary or *switching* rectangular topology matrix representation was proposed by Györök [14]. His proposal, however, did not include further reproduction mechanisms and was not designed for fast checks of a single terminal connectivity. Gan et al. [15] suggested an undirected weighted graph representation where graph vertices represent component nodes, while component types and values are represented by branch weights. The idea results in a relatively efficient, lightweight circuit representation, but is limited to basic passive two-terminal components.

All the above-mentioned issues mainly stem from an inappropriate representation of a circuit topology. It is therefore vital for the successful computerizing of the analog circuit synthesis to have a suitable topology representation, which is the main focus of our paper.

The structure of this paper is as follows. In the following section we discuss the main idea behind of our analog circuit representation and its basic properties. We also present algorithms that allow conversion to a SPICE netlist. Later, in Section 3 we describe the algorithm used to alter and evolve the circuit topology in such a way that a solution fits the high-level requirements given at the beginning. In Section 4, we show that our approach is indeed successful in synthetizing an analog circuit from scratch.

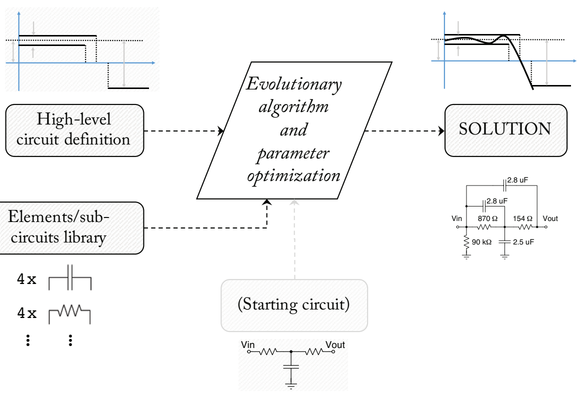

High-level circuit definition
Evolutionary algorithm and parameter optimization
SOLUTION
Elements/sub-circuits library
4x
4x
(Starting circuit)
Vin
Vout
2.8 uF
2.8 uF
870 Ω
154 Ω
90 kΩ
2.5 uF

Figure 1: The main idea behind our analog circuit topology synthesis tool.

2 Circuit representation technique

Figure 1 summarizes the concept of our approach. Central to the synthesis is an evolutionary algorithm that searches for an optimal topology, augmented with an additional parameter optimization method. The algorithm builds the population of circuits using the elements and sub-circuits from the library according to the rules that we describe in this section. During the evolution and optimization process, a specified high-level circuit definition serves as a cost function, which the algorithm tries to minimize. The whole process can be arbitrarily biased with a starting circuit.

There are several requirements that we have to consider in order to obtain a circuit representation suitable to be used in the above described process. The representation should…

- … lend itself to computationally inexpensive modification of topology;
- … be able to prevent uncontrolled growing of a circuit;
- … provide a large search space;
- … allow simple detection of forbidden or unwanted connections.

2.1 The Topology matrix

Probably the most obvious way of decoding a circuit topology is an upper-triangular square binary matrix as shown in Figure 2. The matrix has one row and column assigned to each one of the terminals of all of the elements from the library as well as all the possible main terminals such as GND, V_{in}, V_{out} and similar. The size of the matrix does not change during the evolution. Rather, an element is connected or disconnected from the circuit by setting the corresponding matrix elements to one or zero. Specifically, in order to connect two terminals together, we put a logical one to a place where a row representing the first terminal intersects the column representing the second terminal. By definition, every terminal is connected to itself. That is why the matrix has all ones on the principal diagonal. That way, we can form any possible topology using the elements from the library.

It is obvious that an element is excluded from the final circuit when none of the rows or columns belonging to that element contain any ones (except for the diagonal elements, which connect each terminal to itself). Notice, however, that there are other cases that also exclude an element from the circuit. For example, an element is also excluded when only one of its terminals is connected elsewhere or all of its terminals are short-connected together.

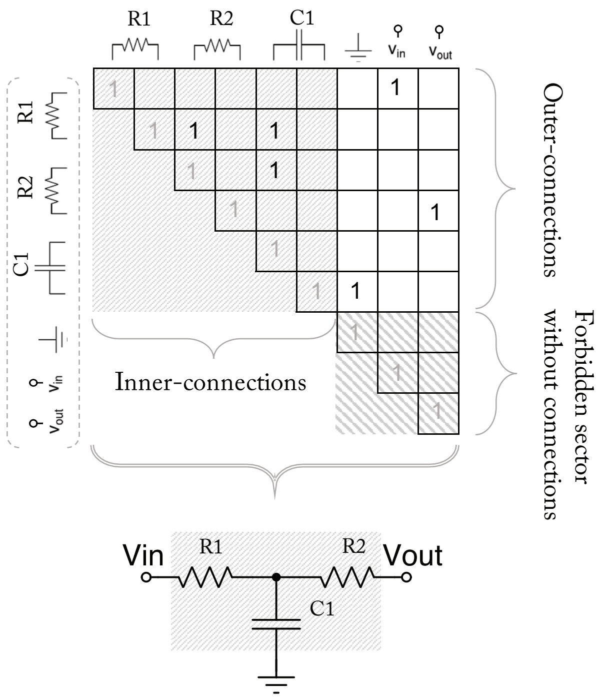


Figure 2: An example of a topology matrix, its main sectors, and the actual circuit that the matrix encodes.

Notice that the topology matrix contains several sectors. The first one is a so-called Inner-connections sector, where all connections between elements and sub-circuits are defined. The second one, the Outer-connections sector, contains all the connections to the outside world. It is easy to detect certain forbidden connections in this sector. Namely, there should only exist a single logical one in each row; otherwise, some outer terminals would be connected together, which is nonsense from the design point of view. There is one more sector (the Forbidden sector), within which no connections are allowed. The reason is the same as before—the outer terminals should not connect to each other. It is very important that we are able to detect some of these nonsense situations easily even before we start a computationally expensive circuit simulation.

2.2 Redundant connections

Consider the Inner-connections sector of the topology matrix depicted in Figure 2. It turns out that exactly the same topology can be represented by four different encoding patterns as shown in Figure 3. Namely, as soon

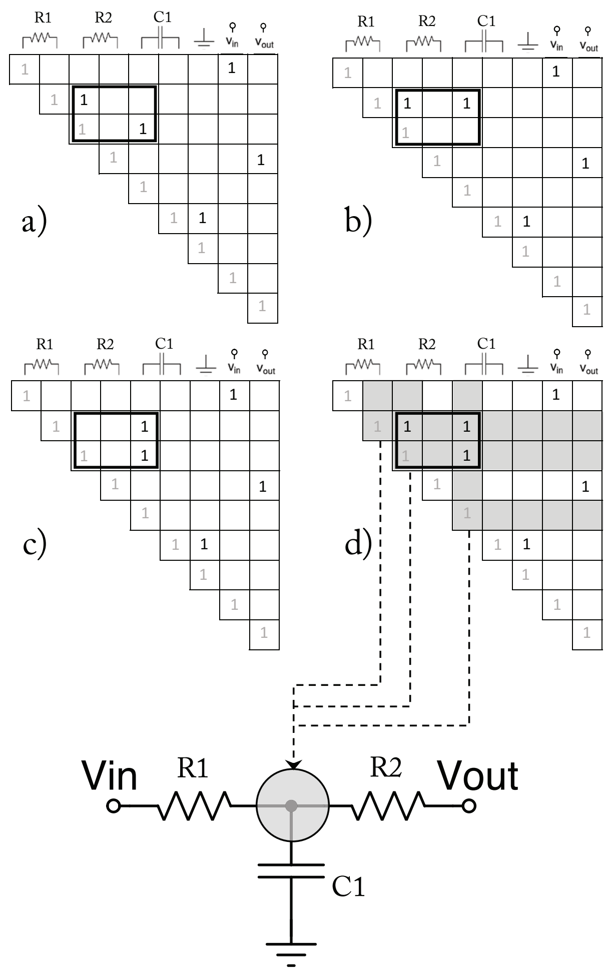


Figure 3: Four different Inner-connections parts of the topology matrix that represent the same T-type circuit.

as we connect two different terminals to a third terminal, we automatically connect the first two terminals together as well. We can observe a similar phenomenon in the genetic code of living organisms, referred to as *degenerate genetic code* [16]. Code degeneracy is important in preserving genotypic diversity as different genotypes (matrix encodings in our case) can represent the same phenotype (a resulting circuit in our case). It is however crucial to have all the connections (even the redundant ones) encoded in the matrix when it comes to creating a netlist to be used by a simulation software like SPICE. By creating a fully redundant matrix, terminals are identified with all joint nodes. As seen on Figure 3 d), all logical fields marked grey belong to a joint node between R1, R2 and C1. The first step of building a SPICE netlist from a topology matrix is therefore filling the matrix with all the redundant connections.

The basic idea behind the procedure of filling the matrix with all the redundant connections is to find all the incomplete rectangles (formed by exactly three logical ones in any three of their four vertices) and fill the remaining vertex with a logical one as well. The algorithm that implements this reads as follows (see also Figure 4):

Repeat

(check up and right, insert diagonally) Scans the matrix along its diagonal from left to right and looks for a missing logical one in the direction (*x*, -*y*). This finds all the rectangles with one existing vertex placed verti-

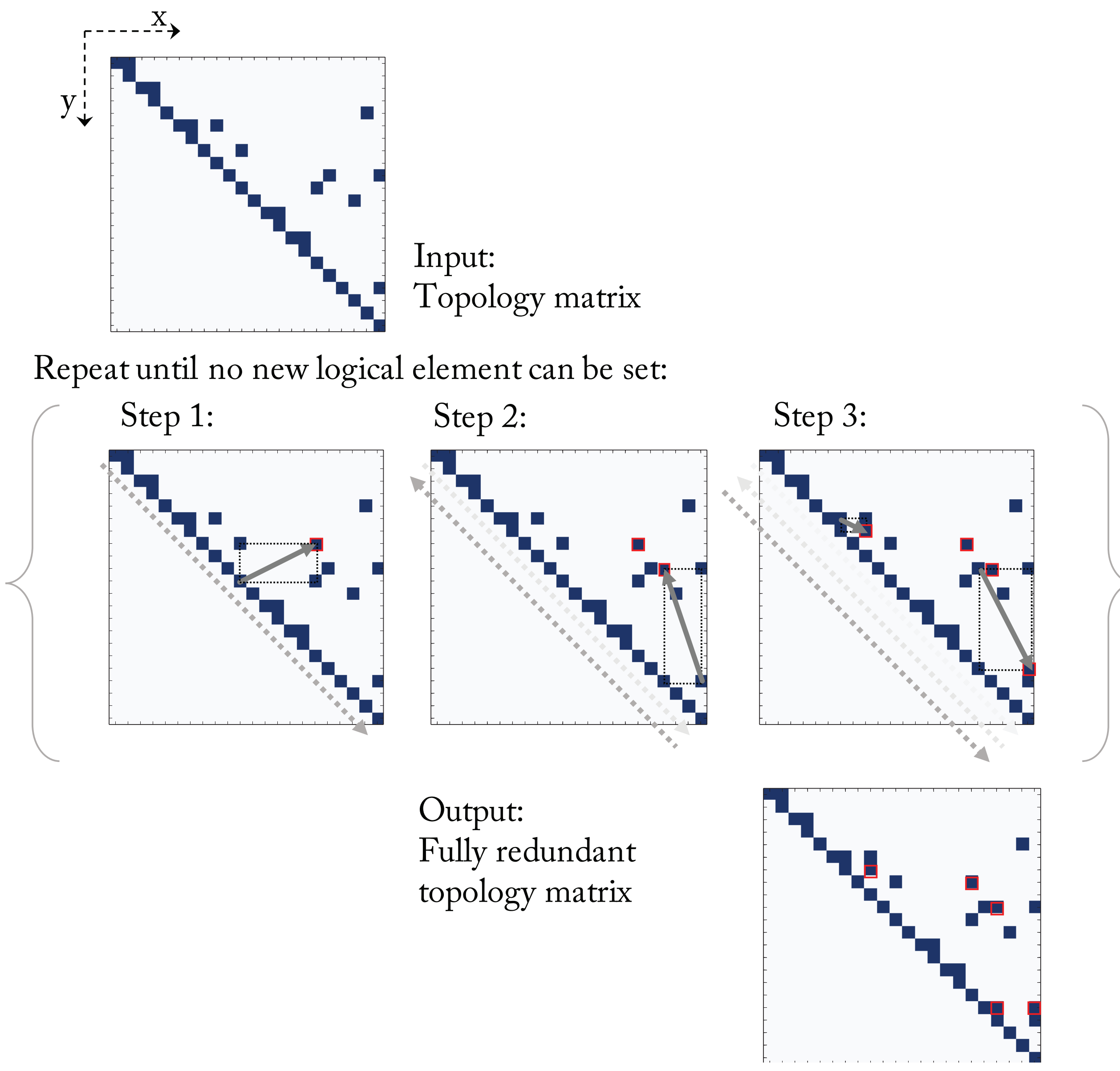


Figure 4: A procedure of finding all the redundant logical ones to build a full topology matrix.

cally and the other horizontally from a certain diagonal element.

(check right and diagonally, insert up) Scans the matrix along its diagonal from right to left and looks for a missing element in the direction *-y*. It finds all the rectangles with two existing vertices in the same column.

(check up and diagonally, insert right) Scans the matrix along its diagonal from left to right and looks for a missing element in the direction *x*. It finds all the rectangles with two existing vertices in the same row.

Until no new logical one was inserted

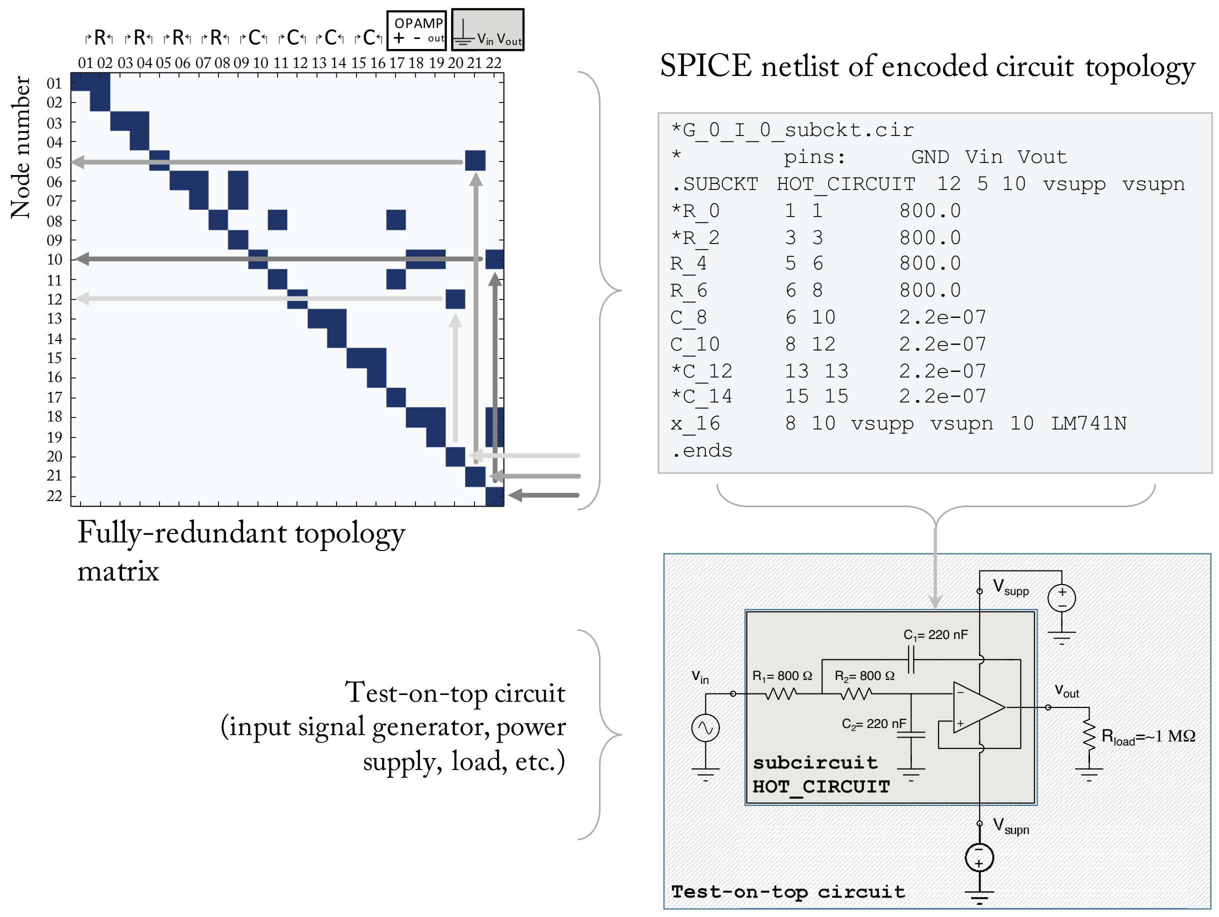
2.3 Parameter vector

Apart from the circuit topology, we also need to encode the numerical parameters such as resistances, capacitances, or transistor gate widths and lengths, in order to fully describe a circuit. We simply store those parameter values in a plain one-dimensional real vector, which leaves us with a complete genotype of a circuit, represented by a topology matrix and parameter vector.

2.4 Matrix-to-netlist conversion

Since we will analyze the circuit using numerical SPICE models, we need to translate the topology matrix together with the parameter vector into a SPICE netlist. The netlist has a simple syntax as shown on the right of Figure 5. Each line starts with the name of an element, the first character of which defines the element or sub-circuit type. The number that follows is simply the number of the element if there are more of the same type. Following the element name, there are the numbers of the nodes in the circuit to which the element is connected. At the end of each line there is usually a numerical parameter of the element or a model name.

Once we have calculated a fully-redundant topology matrix, there is not much work left to do to build a netlist. We demonstrate the whole procedure on the case shown in Figure 5. Let us first identify the node number of terminal V_{out}. The terminal is represented by the diagonal logical one in the bottom right corner of the matrix, pointed to by the darkest gray arrow. From this point, we search for the topmost logical one within the same column. The row index of this logical one represents the node number to be used in the netlist. We



```
*G_0_I_0_subckt.cir
*          pins:     GND  Vin  Vout
.SUBCKT  HOT_CIRCUIT  12  5  10  vsupp  vsupn
*R_0      1  1      800.0
*R_2      3  3      800.0
R_4       5  6      800.0
R_6       6  8      800.0
C_8       6  10     2.2e-07
C_10      8  12     2.2e-07
*C_12     13  13    2.2e-07
*C_14     15  15    2.2e-07
x_16      8  10  vsupp  vsupn  10  LM741N
.ends
```

Figure 5: The conversion from a topology matrix to a netlist.

repeat the same procedure for each and every terminal contained in the matrix. In Figure 5, there are two additional arrows indicating the node number identification for terminals V_{in} and ground although all the terminals follow the same procedure.

It could happen that a diagonal logical one is the only non-zero logical element in the column. In that case, the node number assigned to the terminal is simply the row number of that diagonal logical one (i.e., the left terminals of R_0, R_2, C_{12} and C_{14}). The matrix, netlist, and topology in Figure 5 represent a Sallen-Key active LP filter. Notice that not all available elements are used in the resulting topology. We can see from both the matrix and the netlist that four of the nine available building blocks have their terminals connected together. This automatically means they are excluded from the topology. In the netlist, we commented out the excluded elements using an asterisk (*) to save Spice some unnecessary computation.

Notice that the resulting circuit is coded as a sub-circuit in the netlist. During the evaluation process, this sub-circuit is encapsulated in a special test circuit providing the necessary power supplies, input signals, loads, and measurement points as seen in the bottom right of Figure 5.

3 Search algorithm

Up to this point, we have explained how we encode an individual circuit (the genotype) and how we build a netlist suitable for its simulation (the phenotype). We are now ready for developing an evolutionary algorithm that will evolve a circuit based on a specific fitness specification. The algorithm is similar to the one that we used in [17].

Evolution is a process that allows a biological population to adapt to a given environment by means of change in the heritable characteristics of individuals [18]. The favorable changes mean more chance for an individual of surviving in a particular environment. That way, the population becomes better and better adapted to given conditions. This simple and robust procedure is often used as a means of global optimization [19], and we use it in our work as well. For the purposes of this research, we adapted the three basic evolutionary operators: selection (survival of the fittest), crossover (reproduction, also called recombination), and mutation.

3.1 Selection

The first step of an evolutionary algorithm is usually selection of the fittest individuals that will take part in crossover and/or mutation operations, thus producing offspring. One of the standard methods of selecting best individuals is so-called *tournament selection*. The idea is first to chose a few individuals from the populations at random to be part of a tournament. The winner of a tournament (the individual with the best fitness) is chosen to participate in crossover or mutation. We can easily adjust selection pressure by changing the tournament size. Weak individuals have a greater chance to be selected when the tournament size is smaller.

After we have obtained two winning individuals, we decide between crossover and mutation as shown in Figure 6. The decision is made randomly, based on a given probability. It is not unlikely that, during crossover, we exchange two identical parts of genetic material, which results in two offspring identical to their parents. It turned out that it is beneficial to the algorithm if we discard such offspring and repeat the genetic operation before even evaluating the circuits.

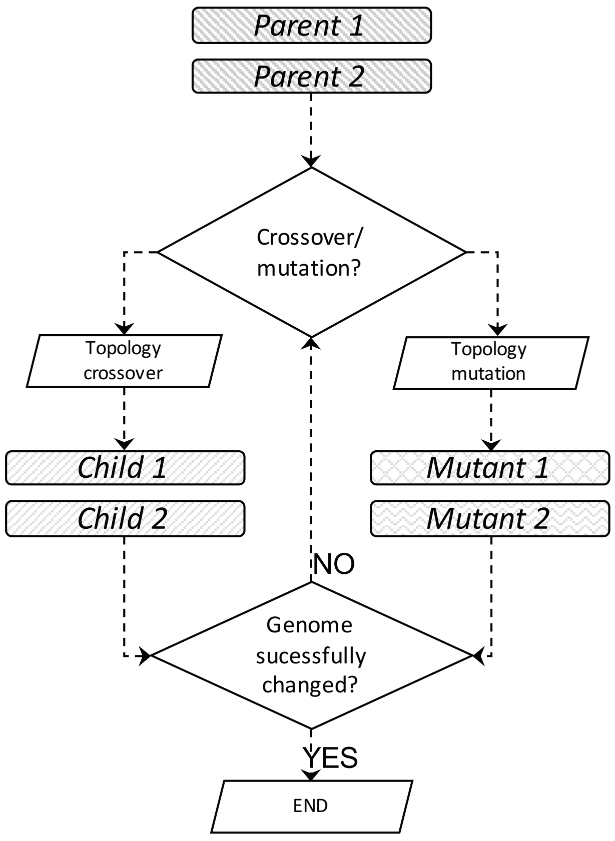


Figure 6: Deciding between crossover and mutation.

Note that tournament selection chooses the best individual from a randomly created subset of individuals. Because of the random selection it might happen that the fittest individuals are not selected at all and therefore could not proceed into the next generation. To prevent this kind of loss, we employ additional elitist selection to ensure that a certain number of the fittest individuals proceed to the next generation even though they have not been selected during any of tournaments.

3.2 Crossover

The basic idea of our crossover technique is closely connected with the encoding type of an upper-triangular matrix. Recall that each logical one on a matrix diagonal corresponds either to a pin of an element or an outer connection (see Figure 2). Each logical one on the right of a particular diagonal element connects the pin to the corresponding pin on its right (see the row of the elements above the matrix in Figure 2). Similarly, each logical one above a particular diagonal element connects the pin to the corresponding pin on its left. As soon as we delete all logical ones from both, the row and column intersecting the diagonal element in question, we remove every information about how that particular pin is connected with the rest of the circuit. Our crossover operator exchanges information about the connections of any number of pins between one and four where the number of exchanged pins is randomly selected. Figure 7 shows examples where one ($N = 1$) and three ($N = 3$) pins are exchanged. The parent on the right is deliberately shown as a full upper-triangular matrix to better illustrate the effect of crossover.

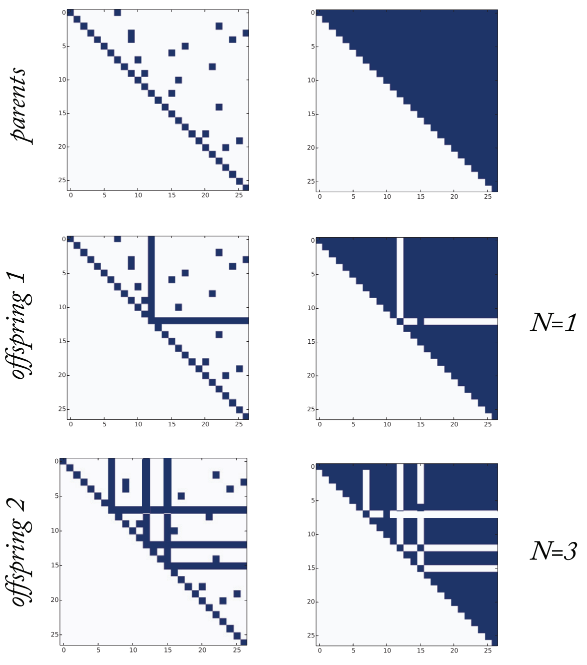


Figure 7: Topology matrix crossover examples exchanging information about one (N = 1) and three (N = 3) pin connections.

3.3 Mutation

Mutation is a random modification of genotype of a selected individual. Our implementation of a mutation operator randomly changes circuit connections in three different ways. It either removes, moves, or adds a logical one to a connection matrix. In case when a mutation operator is selected to be carried out upon the selected individual, one of these three mutation variants is performed based on an evenly distributed random choice.

3.4 Parameter vector optimization

Apart from the circuit topology, the circuit parameters have to be optimized during the evolution as well. We use the PSADE global optimization algorithm [2] to perform this task. PSADE is a hybrid method combining simulated annealing and differential evolution. The method was proven successful on a class of circuit optimization problems, so we use it to alter and additionally optimize the evolving circuit numerical parameters. Parameter optimization is however computationally expensive and optimizing each and every circuit in the generation would make the process unwieldy. It turned out that applying parameter optimization every 10th generation on three randomly chosen circuits (from the 10 best ones in the current population) is quite beneficial to the evolutionary process.

3.5 Circuit evaluation

One of the most important aspect of every evolutionary process (and indeed any optimization) is evaluation of the performance (a.k.a. fitness) of the members of the population. There are few general guidelines as how to do this and a designer mainly has to rely on his or her experience. The goal of our research was to synthetize a passive analog high-pass filter, with –3 dB starting pass band frequency of 8 kHz and the deepest possible damping in stop-band but not higher than –40 dB. We selected four main performance criteria: ripple, damping, f_{pass}, and gain as illustrated in Figure 8. Filter optimization penalty functions are usually designed with a fixed frequency domain structure [3] (i.e., the frequency ranges defining the ripple, dumping, and gain measurements are fixed during the optimization). When evaluating the frequency response, the real damping (or slope) is measured correctly only when f_{pass} is matched to wanted frequency (Figure 8 top). Some evaluated filters might have a proper shape overall, but at wrong frequency. This does not necessary mean that damping is wrong but rather that f_{pass} is off.

Frequency-fixed fitness detection works well for parameter optimization with a fixed topology. In our evolution procedure topology changes, but parameters are fixed until the PSADE triggers. With filters, mainly the topology (the order and type) defines the shape of frequency response, and parameters define bands [20]. This is why we allow f_{pass} to be off during the evolution, but measure other properties correctly (Figure 8 bottom). Doing so, we do not a-priori discriminate circuits, whose f_{pass} is off, but have other qualities.

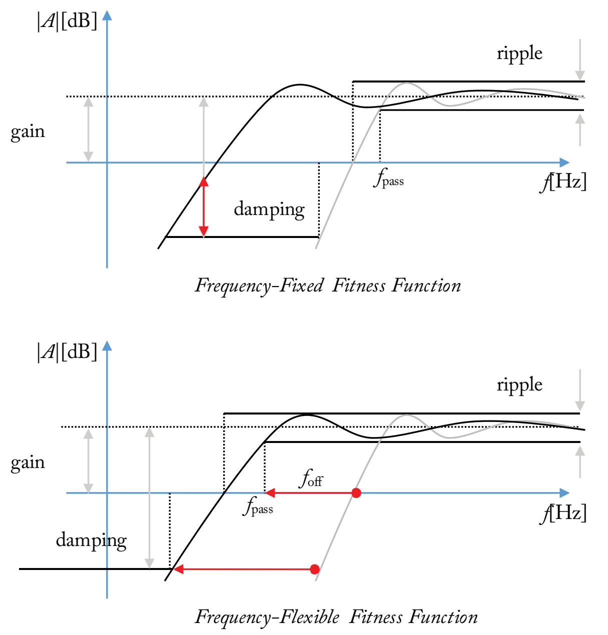


Figure 8: Frequency-Fixed versus Frequency-Flexible fitness function.

We calculate the overall fitness of the circuit using the following cost function:

$$r = \begin{cases} ripple - 0.5\,dB, ripple > 0.5\,dB \\ 0, ripple \le 0.5\,dB \end{cases} \tag{1}$$

$$d = \begin{cases} 40\,dB - damping\ , damping < 40\,dB \\ 0, damping \ge 40\,dB \end{cases} \tag{2}$$

$$f_{off} = \left|\log_{10} 8\,kHz - \log_{10} f_{pass}\right| \tag{3}$$

$$g = \left|0\,dB - gain\right| \tag{4}$$

$$\text{cost} = w_1 r + w_2 d + w_3 f_{\text{off}} + w_4 g \tag{5}$$

where r is ripple larger than 0.5 dB in the pass band, damping is d, smaller than -40 dB in stop band, f_{off} is a difference between f_{pass} and 8 kHz and g is the gain objective. After a number of initial experiments we empirically set the weights to be $w_1 = 1$, $w_2 = 20$, $w_3 = 7$, and $w_4 = 10$. In addition, we weight every individual resulting in an unsuccessful measurement or simulation with factor of $10^3 * N_{nosucess}$, and we weight every individual with a forbidden short-circuit detected already in binary topology matrix with $2 * 10^4 * N_{SC}$, where N_{SC} is a number of detected short-circuits and $N_{nosucess}$ is a number of unsuccessful measurements and analyses.

All measurements were made using the PyOPUS Python library for circuit optimization, which enables simultaneous circuit evaluation on multiple processing cores [21]. Simulations were executed using HSpice.

3.6 The evolutionary algorithm

Figure 9 summarizes the complete evolutionary algorithm used in our research. As the first step, we create an initial random population of topology matrices and parameter vectors. This is done simply by creating topology matrices with evenly distributed logical ones through the whole matrix. Before entering the main optimization loop, we evaluate the initial population and sort the individuals according to their fitness. After performing the genetic operations of selection, crossover and mutation, we evaluate the newly generated individuals. If at least one of them fits the design criteria, we stop the procedure. Otherwise, if the generation number is divisible by ten, we randomly select three of the best ten individuals and run the PSADE optimization algorithm on their parameter vectors.

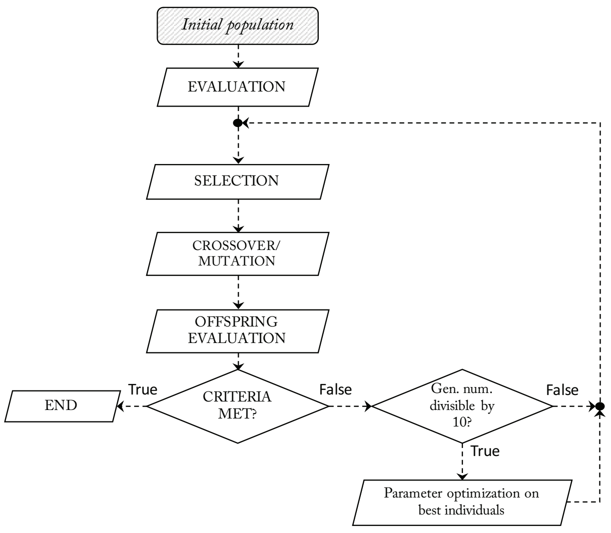


Figure 9: The evolutionary search procedure.

4 Results

In this section, we show the results of two separate runs of our evolutionary algorithm using identical algorithm parameters but with different fitness functions. We included six resistors and six capacitors in the element library to be used by the algorithm. The initial parameters were randomly chosen for every circuit in the initial population ranging from 1 kΩ to 10 MΩ for the resistors and 1 pF to 10 µF for the capacitors. The same values were also used as the constraints for the PSADE parameter optimization. The population size in both runs was set to 400 individuals. The parameters are summarized in Table 1.

Table 1: Evolution parameters.

Population size	400
Tournament size	3
Elite size	8
Crossover probability	0.8
Mutation Probability	0.2

In the first run, we wanted to evolve a high-pass filter with at least –40 dB damping in the stop-band, at least one decade below f_{pass}. The evolution produced a solution after only 430 generations, which took about an hour using a cluster of 10 Core i5 Linux machines. We can observe the resulting circuit in Figure 10 (1) and its frequency response on Figure 11 (1). The resulting RC filter is comprised of three capacitors and four resistors. Its frequency response shows a low (almost zero) ripple in pass band, 0 dB gain and –40 dB/decade slope. There is a return point from –50 dB towards –40 dB at 0.9 kHz.

In the second case, we required steeper damping of –60 dB. In this case, the evolution reached the maximum limit of 2000 generations, which took approximately five hours on the same hardware. The resulting circuit (the circuit in Figure 10 (2) with the values in brackets) met most criteria except for f_{pass}, which settled at 1 kHz instead of 8 kHz. The reason for this failure, however, was not the evolutionary algorithm itself but rather the internal limit on the maximum number of iterations of the PSADE parameter optimization algorithm, which was set to 10^5. This internal limit was set in order to keep each parameter optimization run reasonably short during the topology evolution. After we have run the additional PSADE optimization on the final topology, the starting frequency of the pass-band moved to the desired value (cf. the plots of the second run in Figure 11). It took PSADE additional $5 \cdot 10^6$ iterations to fine-tune the circuit parameters. The final circuit is comprised of five capacitors and four resistors, which form an RC filter with a similar frequency response as in the first case, except with better damping (Figure 10 (2) and Figure 11 (2) – "fine-tuned"). Similarly to the first case, there is a return point from -61 dB towards -55 dB, which slightly violates the damping criterion. This problem could be solved simply by increasing the weight factor assigned to damping in the cost function.

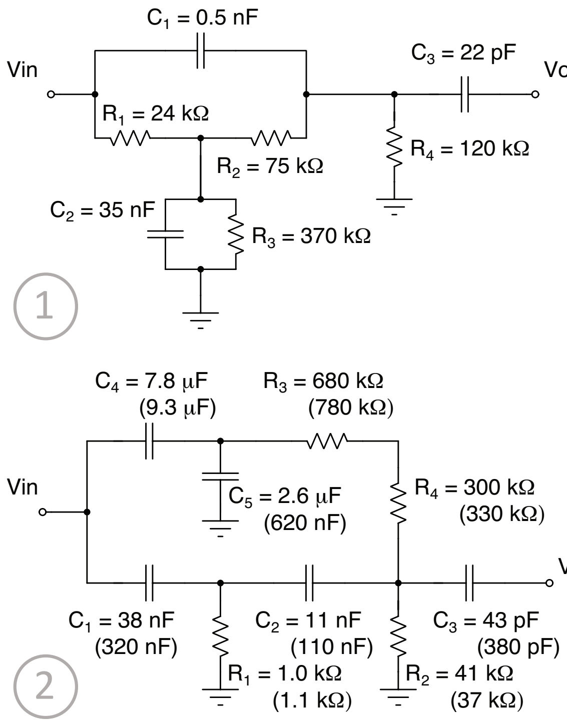


Figure 10: Automatically evolved filter topologies and their numerical parameters when requiring dumping of –40 dB (1) and –60 dB (2). With second circuit, parameters given in brackets are the raw algorithm solution and fine-tuned ones are given above.

Note that both topologies shown in Figure 10 are the raw output of the algorithm. An analog human designer will still see some obvious (topological) redundancies like, for example, the serially connected resistors R_3 and R_4 in the second filter.

4.1 A Comparison to other existing approaches

In this subsection, we compare our approach to other known analog circuit topology representations for evolutionary algorithms found in the literature. A brief glance at Table 2 reveals that our matrix representation technique surpasses the competitive approaches in several categories.

Representations used by Koza [12] and Lohn [13] suffer quite seriously from bloat that manifests itself in many redundant circuit branches, which makes it difficult to control the evolution. We have eliminated this problem because a connection matrix cannot change its size during the evolution. Furthermore, we limit a number and the types of allowed components by specifying a pre-defined component library to be used by the algo-

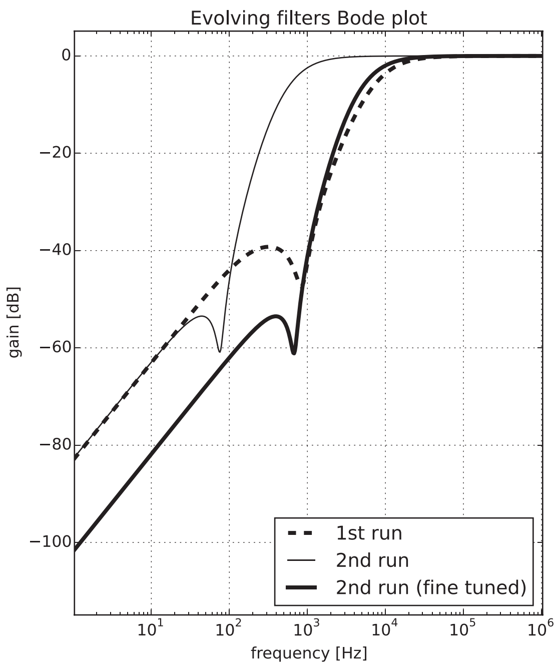


Figure 11: Frequency responses of automatically evolved both topology and parameters for two high-pass filters. For the second run, additional parameter fine-tuning was carried out.

rithm. The only redundancies that emerge in our case are some parallel/serial repetitions of same-type components.

With Koza [12], Lohn [13], and Gan [15], the basic building blocks are limited to two-pole components. Although the methods allow the usage of transistors, one of their three terminals should always be fixed beforehand to one of the outer connections. However, with our matrix representation it is possible to use building blocks having an arbitrary number of terminals, which vastly increases the circuit search space.

The approach proposed by Kruiskamp [10] is limited to combine 24 predefined topologies which is hardly practical for real-life problems. With our representation, there is no limit on the type and size of the evolved topology other than the one imposed by the size of a connection matrix and a pre-defined component library.

Koza [12], Lohn [13] and Györök [14] do not suggest any routine for short-circuit checks directly on the circuit representation level. Our approach incorporates efficient checking of a connection matrix. Configurations resulting in a short-circuited topology are excluded from further unnecessary and potentially expensive computations.

The approach by Kruiskamp [10] requires quite some information about the desired circuit topology structure to be input by the practitioner in advance. Many other methods, on the other hand, demand very little or even no such information. This way, the evolution is able to come up with a completely new topology for a certain task. Our method is flexible in this aspect because it allows a practitioner to enter an arbitrary amount of prior knowledge about the circuit by constructing an appropriate component library. By adding different sub-circuits to the library, or injecting known topologies into the initial generation of the evolutionary search, he or she can freely control the amount of entered knowledge.

Implementation of genetic programming can be quite an arduous task, involving genetic tree definition, tree-to-netlist conversion, and other complex mechanisms.

Table 2: A comparison to other existing circuit topology representation techniques.

	Bloat-safe?	Final topology size	Number of sub-circuit terminals	Search space size	Built-in topology check	Prior-knowledge required	Implementation complexity	Reproduction mechanisms complexity
Kruiskamp [10]	yes	limited	arbitrary	limited	yes	high	low	low
Koza [12]	no	unlimited	two	enormous	no	low	high	high
Lohn [13]	no	unlimited	two	enormous	no	low	high	low
Györök [14]	yes	limited	arbitrary	controllable	no	low	low	/
Gan [15]	yes	limited	two	controllable	yes	low	low	low
This work	yes	controllable	arbitrary	controllable	yes	controllable	low	low

It is also very important that our matrix representation can be implemented quite easily. Furthermore, unlike different genetic trees reproduction operators, our reproduction mechanisms are straightforward to implement and work natively with the upper-triangular connection matrix. Györök [14], as seen in Table 2, does not propose any reproduction mechanism other than a Monte Carlo method.

5 Conclusions

We developed an analog circuit representation technique for automated topology synthesis in the form of an upper-triangular binary matrix. The representation prevents bloat during the evolution so that the circuit cannot grow over the limits. Nevertheless, the implementation still enables a search over quite a large solution space whose size can be controlled by the element/sub-circuit library. We observed the redundancy phenomenon in the matrix-to-netlist conversion, which is important for maintaining the genetic diversity of a population of circuits but is problematic from the netlist generation point of view. We proposed a procedure of generating a fully-redundant matrix that lends itself easily to generation of a SPICE netlist. Based on the proposed topology representation, we developed the crossover and mutation genetic operators and an evolutionary algorithm suitable for evolving an arbitrary circuit based on a high-level statement about its required properties. We demonstrated the suitability of the approach with an evolution of a passive high-pass filter. We believe that the results of this research can easily be extended to synthetizing more complex passive and even active circuits, which will be a focus of our future research.

6 References

1. L. W. Nagel and D. O. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," 1973.
2. J. Olenšek, T. Tuma, J. Puhan and Á. Bűrmen, "A New Asynchronous Parallel Global Optimization Method Based on Simulated Annealing and Differential Evolution," *Applied Soft Computing,* vol. 11, pp. 1481-1489, 2011.
3. J. Puhan, T. Tuma and I. Fajfar, "Optimisation methods in SPICE: a comparison," in *Proceedings of European Conference on Circuit Theory and Design (ECCTD)*, 1999.
4. U. M. Garcia-Palomares, F. J. Gonzalez-Castaño and J. C. Burguillo-Rial, "A Combined Global & Local Search (CGLS) Approach to Global Optimization," *Journal of Global Optimization,* vol. 34, pp. 409-426, 2006.
5. H. Schmidt and G. Thierauf, "A combined heuristic optimization technique," *Advances in Engineering Software,* vol. 36, pp. 11-19, 2005.
6. S. Ebrahim Sorkhabi and L. Zhang, "Automated topology synthesis of analog and RF integrated circuits: A survey," *INTEGRATION, the VLSI journal,* vol. 56, pp. 128-138, 2017.
7. M. G. R. Degrauwe, O. Nys, E. Dijkstra, J. Rijmenants, S. Bitz, B. L. A. G. Goffart, E. A. Vittoz, S. Cserveny, C. Meixenberger, G. van der Stappen and H. J. Oguey, "IDAC: an interactive design tool for analog CMOS circuits," *IEEE Journal of Solid-State Circuits,* vol. 22, pp. 1106-1116, Dec 1987.
8. R. Harjani, R. A. Rutenbar and L. R. Carley, "OASYS: a framework for analog circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 8, pp. 1247-1266, Dec 1989.
9. H. Y. Koh, C. H. Sequin and P. R. Gray, "OPASYN: a compiler for CMOS operational amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 9, pp. 113-125, Feb 1990.
10. W. Kruiskamp and D. Leenaerts, "Darwin: Analogue circuit synthesis based on genetic algorithms," *International Journal of Circuit Theory and Applications,* vol. 23, pp. 285-296, 1995.
11. J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge, MA: MIT Press, 1992.
12. J. R. Koza, I. F. H. Bennett, D. Andre, M. A. Keane and F. Dunlap, "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *Trans. Evol. Comp,* vol. 1, pp. 109-128, Jul 1997.
13. J. D. Lohn and S. P. Colombano, "A circuit representation technique for automated circuit design," *IEEE Transactions on Evolutionary Computation,* vol. 3, pp. 205-219, Sep 1999.
14. G. Györök, "Crossbar network for automatic analog circuit synthesis," in *2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2014.
15. Z. Gan, Z. Yang, T. Shang, T. Yu and M. Jiang, "Automated synthesis of passive analog filters using graph representation," *Expert Systems with Applications,* vol. 37, no. 3, pp. 1887-1898, 2010.
16. K. Baumgardner and G. Elseth, Principles of Modern Genetics, West Publishing Company, 1995.
17. Ž. Rojec, Á. Bűrmen and I. Fajfar, «An evolution-driven analog circuit topology synthesis,» in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.
18. C. Darwin, The Origin of Species, P. F. Collier & Son, 1909.

19. D. E. Goldberg and J. H. Holland, "Genetic Algorithms and Machine Learning," *Machine Learning,* vol. 3, pp. 95-99, Oct 1988.

20. R. Schaumann, H. Xiao and V. V. Mac, Design of Analog Filters 2nd Edition, New York, NY, USA: Oxford University Press, Inc., 2009.

21. A. Bűrmen, J. Puhan, J. Olenšek, G. Cijan and T. Tuma, "PyOPUS - Simulation, Optimization, and Design," EDA Laboratory, Faculty of Electrical Engineering, University of Ljubljana, 2016.

Arrived: 10. 11. 2017
Accepted: 06. 03. 2018