An Evolution-Driven Analog Circuit Topology Synthesis

Žiga Rojec
Faculty of Electrical Engineering
University of Ljubljana
Ljubljana, Slovenia
Email: ziga.rojec@fe.uni-lj.si

Árpád Bűrmen
Faculty of Electrical Engineering
University of Ljubljana
Ljubljana, Slovenia

Iztok Fajfar
Faculty of Electrical Engineering
University of Ljubljana
Ljubljana, Slovenia

***Abstract*—A design of an analog circuit is often a time consuming, iterative procedure, which strongly depends on designer's knowledge and experience. A designer is facing tough requirements for designing within a time budget and consuming as few resources (both human and material) as possible. We designed a system that can assist and speed-up a design process. Based on a high-level statement describing the circuit functionality, the system evolves both a topology and parameters using a bio-inspired evolutionary algorithm. Each circuit within a population is coded as a multidimensional chromosome. Proof of concept is given by a fully-automatic evolution of a 1 KHz low-pass passive filter.**

I. Introduction

A design of an analog circuit is roughly a three-step procedure (Fig. 1) [1]. Initially, a designer collects the information about the detailed specifications on the behavior of the final circuit and eventual physical constraints and limitations given by either the customer or technology. Next, the designer chooses a suitable topology that previously performed well for a similar task. Solution of this step is highly dependable on designer's knowledge and expertise. The final step is most often a parameter tuning; in order to meet all the desired specifications, the designer has to adjust resistances, capacitances, transistor gate widths and lengths, and the like. Often a parameter tuning does not give satisfactory results already in the first trial, so a jump back to the second step (i.e., modifying the topology) is required. The whole design process is an intensive, time-consuming, and iterative procedure, which is why computer assistance and automatization of the process is desired. A suitable computer system could help a designer not only to speed up the whole process but even to get novel design ideas.

The problem of parameter tuning has been studied quite extensively (e.g., [2]–[6]), which has produced many efficient and practically applicable methods for automatic parameter optimization. A practitioner can choose between a number of existent algorithms such as simulated annealing, differential evolution, genetic algorithm, particle swarm optimization or various hybrid methods, to name just a few.

Similarly—in order to further reduce the time and human resources needed for a design process—the problem of circuit topology synthesis has also been broadly investigated. As a result, several design approaches have been proposed. With

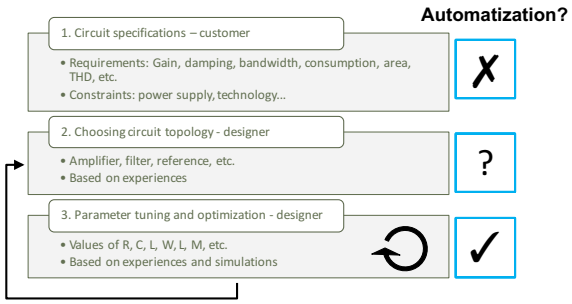


Fig. 1: A three-step design process scheme.

IDAC [7], OASYS [8] and OPASYN [9], the synthesis is based on random selection of topologies from a predifined library. In DARWIN [10], the circuit evolution uses a genetic algorithm, which can choose between 24 predefined circuits. Another system that is capable of evolving analog circuits within predefined high-level topology is MOJITO [13], which performs a search using genetic programming. All mentioned approaches feature a rather limited search space, except for the last one, which is searching over more than 100000 of possible topologies. On the other hand, Koza's WYWIWYG [11] concept offers a search space comprising billions of possible topologies. While using a genetic programming technique to evolve a circuit topology, each individual within a population is represented by a computer program described by a sophisticated tree-like syntax. However, besides its undue complexity, there is another disadvantage of Koza's circuit representation technique, which is the occurrence of a program *bloat* [17], a well known phenomenon in the genetic programming community. The result of the bloat are circuits whose sizes become enormous during the automatic synthesis. Another approach to automatic circuit design is the Cartesian Genetic Programming, invented by J. F. Miller [14]. The approach was proven successful on digital circuit design and is as well immune to bloat. However, CGP encodes a directed graph representation, which is more natural to digital circuits but less to analog ones.

In this paper we propose a novel evolutionary algorithm for automatic circuit topology synthesis augmented with automatic parameter optimization, which is run occasionally during

the evolution process. In Section II we introduce a specialized analog circuit representation in a form of a two-dimensional matrix, which is simple to define and allows both usage of pre-defined sub-circuits as well as individual electrical elements. Unlike the Koza's tree-like representation, our representation is more natural to the circuit topology and also limits the maximum possible size of the circuit thus preventing the bloat. Although the maximum size of the circuit is limited, our search space is still enormous. For example, having eight two-pole elements results in 1.65×10^{19} possible topologies. Using the proposed two-dimensional circuit representation, we then develop a two-dimensional genetic algorithm to effectively search over the space of all possible topologies, which is described in Section III. Finally, in Section IV, we show an example of evolving an analog, passive low-pass filter, demonstrating that our approach can evolve a simple circuit on a personal computer within a matter of minutes.

II. Analog Circuit Representation

In order to be able to automatically modify the circuit topology, we need to be able to represent this topology in the most appropriate way. Having evolutionary terminology in mind, we call such representation a circuit's *chromosome* (a.k.a. *genotype*). Changing a genotype usually results in a modified circuit with a possibly different behavior (also called a *phenotype*). Moreover, the circuit representation has to be such as to enable an easy exchange of the genetic material using appropriate genetic operators.

A. Connection matrix

For the purposes of our work, we represent a circuit in a form of a logical *connection matrix*. Individual discrete elements are placed in a row, where every element pin is represented by a column of the matrix (see Fig. 2). Each one in the matrix represents a connection to another element pin. Note that the corresponding matrix has an upper-triangular form with all the diagonal elements set to one because, by definition, each pin is connected to itself. The matrix is divided in two segments: the left one defines connections between the elements while the right one connects the circuit to the outer world (e.g., v_{in}, v_{out}, and GND, as seen in Fig. 2). Each element can have two or more connection pins, which makes it possible for each element to be either a simple discrete element or an arbitrary complex sub-circuit, as previously proposed in [18]. Using this setting, one can modify the circuit topology simply by changing the positions of ones in a connection matrix (see 4a). Note that it is not necessary that all the elements within the matrix are actually used in the final circuit. One can simply take out elements by disconnecting their pins or even short-circuiting unused pins as seen in Fig. 3.

B. Value vector

Each element is also defined by one or more parameters such as a resistance, capacitance, or the transistor gate width and length. We represent these as components of a so-called *value vector* (see Fig. 3), which are placed in the same order

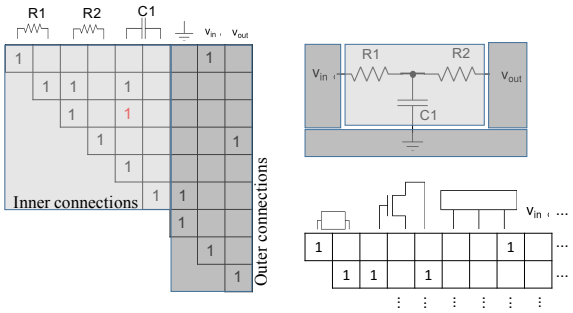


Fig. 2: An example of a matrix representation of a simple T-circuit with the corresponding connection matrix (left), the schematic of the corresponding circuit (top right), and a generalization of the technique (bottom right).

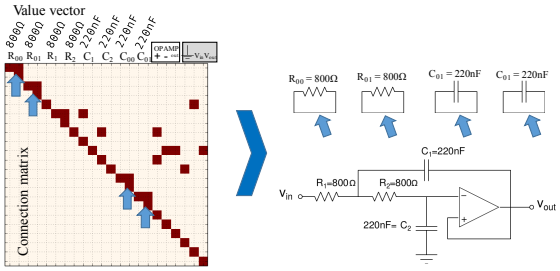


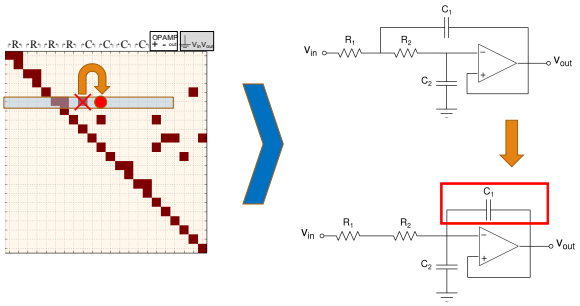
Fig. 3: A full circuit chromosome (*the Connection matrix* and *Value vector* of a Sallen-Key low-pass filter. The arrows are pointing to the excluded elements (right) and to their representations in the chromosome (left).

as are the corresponding elements in the connection matrix. Modifying the values of a value vector of a given topology in order to achieve the best results is equivalent to circuit parameter optimization.

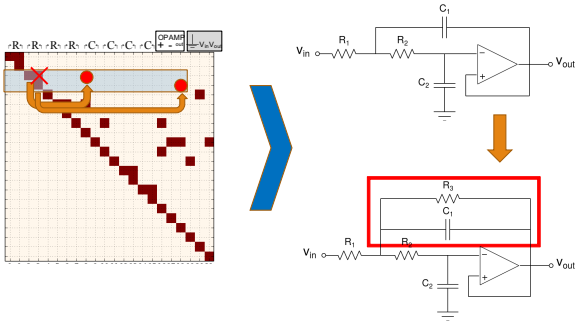
III. Synthesis Algorithm

As in any evolutionary algorithm, we start the process by creating an initial population of randomly created individuals. After the initial population has been evaluated and sorted by their fitness values, we select the best individuals for creating offspring. Using various reproduction techniques (see below), we get new individuals (see Fig. 5).

An offspring is generated through two main reproduction mechanisms—crossover and mutation. Based on given mating probability ($mating\_prob$), the two chosen individuals exchange genes either of connection matrix or value vector. Which of those two events will occur is statistically defined by the parameter $topologyChange\_prob$. Moreover, when changing a connection matrix (either in mating or mutation), $innerConnsChange\_prob$ will define whether a reproduction mechanism will work on left of right part of connection matrix (recall Fig. 2). Mutation probability is always $1 - mating\_prob$. Also when mutating, two individuals are chosen and two children are created, each of the two is a mutation of its parent.



(a) The position of the capacitor C_1 is changed after moving a connection in a matrix.



(b) An inclusion of the previously hidden resistor R_3 into the evolving circuit.

Fig. 4: Two cases of possible topology changes caused by a chromosome modification for the Sallen-Key low-pas filter example.

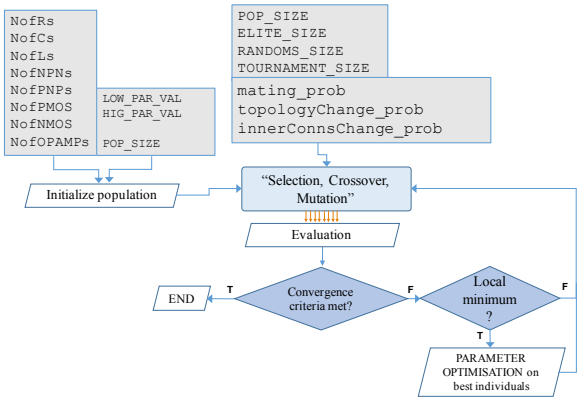


Fig. 5: An evolutionary algorithm flow chart with the listings of the input parameters.

It can happen that after the reproduction phase we get some identical individuals. Experiments have shown, having duplicates in a population slows-down the convergence significantly. Because we want to maintain the population as diverse as possible, we remove all the duplicates at the end of each iteration. We search the population for duplicates by comparing hash values of the population members. In order to maintain a fixed population size, we add randomly generated new individuals to replace the removed ones. After a new population has been completed, we evaluate the unevaluated

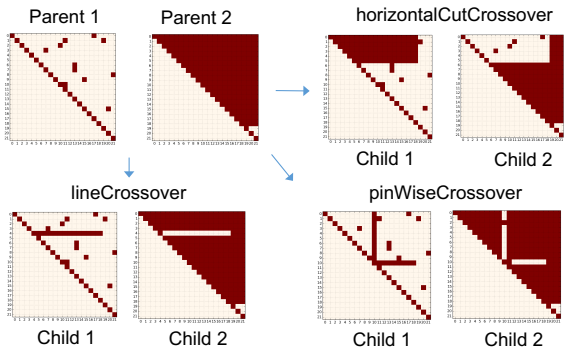


Fig. 6: Mating possibilities.

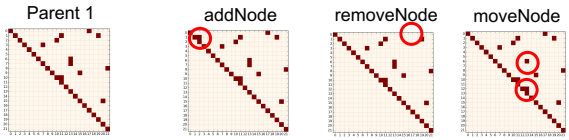


Fig. 7: Possible mutations of Parent 1.

individuals, choose the best ones and repeat the reproduction step.

After every generation, we check whether the stopping criterion of the procedure is reached. If at least one of the individuals meets all the specifications or the number of generations exceeds the specified maximum, we stop the algorithm.

A. Crossover and mutation

Because the circuit topology is represented by a two-dimensional chromosome, we need two-dimensional genetic operators similar to those previously proposed for two-dimensional genetic algorithms [19], [20]. In this section we present some solutions that yielded good results when used with our circuit representation.

1) Connection matrix: Two parents can exchange genetic material by exchanging parts of their connection matrices (see Fig. 6). One of the simplest solutions is to exchange rows of two logical matrices (*lineCrossover*). Note that we only exchange parts of matrices where the inner connections are defined while the connections to the outside of the circuit stay intact. A second alternative is to exchange parts of matrices with multiple inner-connections (*horizontalCutCrossover*). The third approach is to use *pinWiseCrossover*, which exchanges every connection to one element pin. Mutating an individual is easier. Plausible possibilities are adding a connection, removing a connection, and moving a random connection as shown in Fig. 7.

2) Value vector: A crossover of value vectors is carried out through some well-known crossover techniques such as one-point, two-point, or interpolation crossover [21]. We can mutate a value vector by simply choosing a random gene and set it to a random value (within predefined margins).

B. Cost function

In order to evaluate individuals and compare them between each other we propose a *cost function* that returns a higher value for circuits that behave bad, and returns a lower value for circuits that work better. During the evolution, a circuit topology may be changed in such a manner that it becomes useless (e.g., if there is a short-circuit between any of the outer-connections of the circuit). A cost function must penalize such an event, since it does not work beneficially for the genetic pool. Similarly, a cost function has to penalize failed measurements when circuit evaluation fails or returns a non-physical value (e.g., an infinite gain). If no forbidden short-circuits are found and all the measurements are successfully evaluated, then a *circuit score* S is calculated. Usually one of the requirements is to evolve the cheapest working solution. In case of circuits, designing a topology with less elements is desired. That is why we add a small punishment to the score, which depends on the number of the used elements. As a result, the evolution algorithm will tend to produce solutions with less elements. In summary, this is how our cost function looks:

$$C = \begin{cases} W_3 * e^{N_{sc}} & , if N_{sc} > 0 \\ W_2 * e^{N_{fail}} & , if N_{fail} > 0 \\ W_1 * (S + N_{elm} * S * 1\%) & , if N_{elm} > 0, \end{cases} \quad (1)$$

where the numbers N_{sc}, N_{fail}, N_{elm} represent the number of short circuits in outer connections, number of failed measurements, and the number of used elements, respectively. Since a short circuit is a greater failure than a failed measurement, and both are a lot worse than a fully operational circuit, a clear distinction between these is necessary. That is why we chose the values of the weights in our experiment to be $W_3 >> W_2 >> W_1$, in particular, we selected $W_3 = 10^4$, $W_2 = 10^3$, and $W_1 = 1$.

C. Selection

We select individuals to participate in reproduction using tournaments (TOURNAMENT_SIZE). Every individual gets a pair among others from the mating pool. We also define the size of the elite (ELITE_SIZE), which is automatically taken from the previous generation and copied into the next one. In every generation there is also a part of random individuals defined by RANDOMS_SIZE that improve population diversity.

D. Local minimum detection

Global search algorithms often get stuck in a local minimum. We can infer that this has happened from the observation that the cost function value remains relatively high across several generations. Should this happen, we trigger a parameter optimization on several best individuals using the PSADE hybrid optimization method [2].

IV. Synthesis example

In this section we describe an example of a circuit evolution using a case of an analog, passive low-pass filter with the cutoff frequency of 1 kHz, a signal damping of more than 20 dB, a 0 dB gain and ripple of less than 0.5 dB. The used input parameters are listed in Table I. The lower and upper bounds of the parameter values are presented in Table II. The goal of the algorithm is to construct the filter using at most nine discrete passive elements: three resistors, three capacitors, and three inductors. The pinWiseCrossover method was used for a crossover, while for a mutation, one of the add-, remove-, and moveNode methods was randomly chosen using a uniform selection probability.

TABLE I: The input parameters for a low-pass filter evolution.

NofRs	3
NofCs	3
NofLs	3
NofOPAMPs	0
POP_SIZE	200
ELITE_SIZE	4
RANDOMS_SIZE	40
TOURNAMENT_SIZE	3
mating_prob	0.4
topologyChange_prob	0.6
innerConnsChange_prob	0.5

TABLE II: The upper and lower bounds for the parameter values.

	R_1, R_2, R_3	C_1, C_2, C_3	L_1, L_2, L_3
Lower	10Ω	$2.5nF$	$2.5\mu H$
Upper	$500k\Omega$	$2.8\mu F$	$2.8mH$

A. Score function

In order to evaluate the performance of each working filter, we measured three parameters: *ripple*, *gain* and *damping*. The score function was computed as:

$$S = w_1 * ripple + w_2 * damping + w_3 * gain, \quad (2)$$

where $w_1 = 10$, $w_2 = 10$, and $w_3 = 1$. The weights were chosen empirically after the first few algorithm runs in which we experienced a high ripple and slow slope-off. Setting the ripple and damping weights to higher values attributed significantly to the convergence. For the core circuit simulation the SpiceOpus package [22] was used.

B. Results

We ran the algorithm on a Core i5 Linux machine, evaluating four individuals at a time using parallelization tools provided by [23]. After 480 generations (approx. 30 minutes), the best individual fulfilled all the requirements and the evolution was stopped (see Fig. 8). The resulting circuit is shown in Fig. 9.

The evolution resulted in a topology comprising all the available resistors and capacitors (i.e., three of each). The inductors, however, were discarded during the evolution altogether. An analog circuit designer might notice that the resistor R_0 is not critical for the performance of this low-pass filter

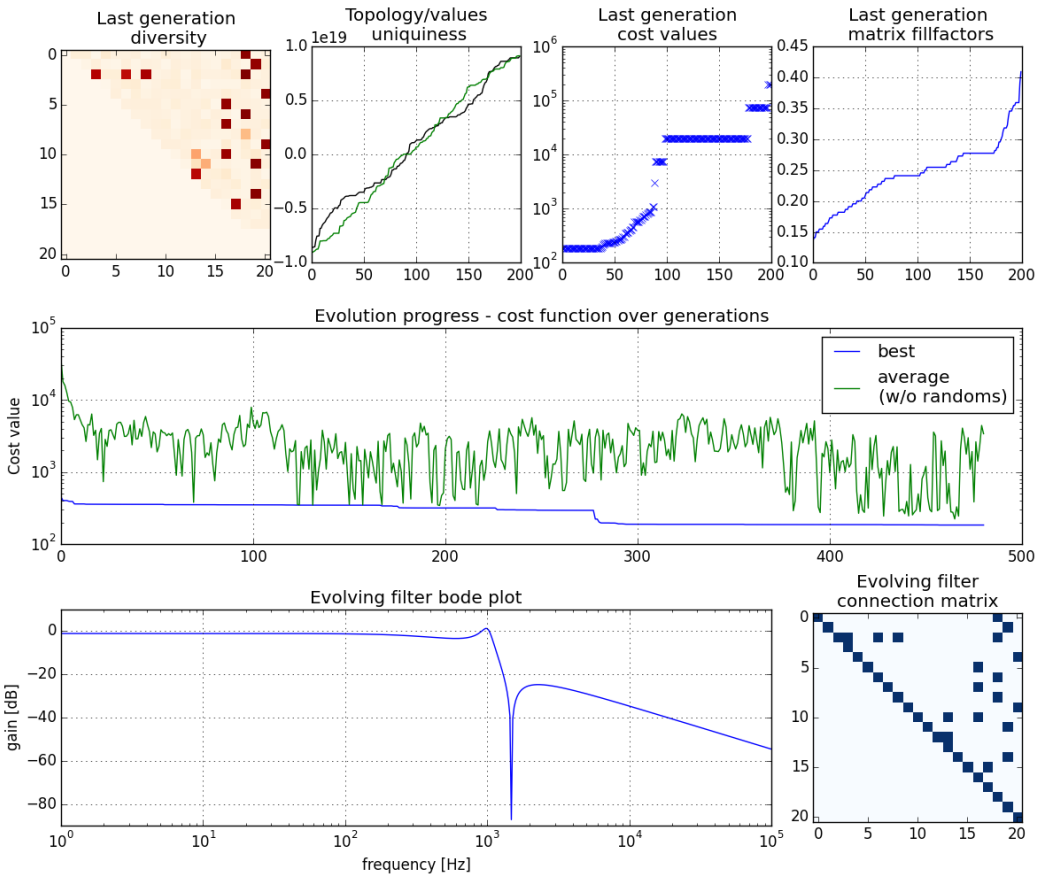


Fig. 8: The progress of the evolution and the properties of the last generation. Top left: the diversity of the connection matrices of the last generation (the darker the color, the more individuals have this particular connection). Top middle left: the hash functions of all the connection matrices (black) and value vectors (green). Top middle right: the cost function results of each individual in the last generation. Top right: the density of the connection matrix for each individual (higher value means more ones in the matrix). Middle: the evolution of the cost function value (best and average). Bottom left: the best filter Bode plot. Bottom right: the best filter connection matrix.

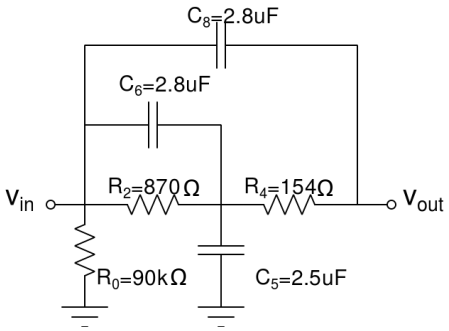


Fig. 9: The evolved passive LP-filter after 480 generations.

although it decreases input resistance (a higher input resistance is better). After running the algorithm further, we could as well observe omission of R_0 since, having two circuits delivering the same performance, the cost function value will be lower for the circuit with fewer elements.

C. Search commentary

One can observe the population diversity in top four plots of Fig. 8.

1) Connection matrices: Diversity of the connection matrices (in top left of Fig. 8) is the highest for the initial population, where the connections are uniformly randomized. After some generations, the algorithm finds some possible solutions and focuses on search around them. That is why at the end of the evolution, the connection matrix diversity plot shows the highest degree of similarity exactly where the ones are placed in the matrix of the final solution (in bottom right). One should regard this diversity plot as the sum of all the matrices from the population—the darker points correspond to the connections that are used in many matrices. Note that the diagonal connections are omitted from this plot for better image contrast.

2) Topology and value uniqueness: In order to inspect and visualize possible repetitions of the same genetic material in the population we translate the circuit chromosome into a

unique 64-bit integer. We do that separately for the connection matrix and the value vector of an individual using a hashing algorithm (see top middle left of Fig. 8).

3) Cost values: We can observe that several individuals share the same cost function value (in top middle right of Fig. 8). This is especially true for values above 10^4, which correspond to the circuits with a poor performance and forbidden connections (e.g., short-circuits). We can hardly distinguish such circuits from each other from the phenotypic point of view, but we still keep some of them in the genetic pool in order to maintain genetic diversity.

4) Connection matrix fill-factor: If the cost function is poorly defined, the algorithm tends to evolve a circuit with every element connected to ground. Because we wanted to detect such a scenario, we defined a so-called *fill-factor*, which is a number between one and zero which shows the percentage of ones contained in the connection matrix. One can observe in top left if Fig. 8, that all the fill-factors are less than 0.5.

V. Conclusion

We proposed a procedure for analog circuit synthesis using the principles of evolutionary algorithms. A circuit topology was represented using a connection matrix capable of encoding all the possible connections between a fixed set of elements that can be used for circuit evolution. Such a representation technique allows a designer to use both simple elements as well as arbitrary complex sub-circuits as basic design blocks. We proposed various crossover and mutation methods for a two-dimensional connection matrix that result in changes to the circuit topology. An experiment of a successful fully automatic evolution of an analog passive low-pass filter on a personal computer was presented. We are currently working on a synthesis of more complex circuits, both active and integrated.

References

[1] G. G. E. Gielen, R. A. Rutenbar, *Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits*. Proceedings of the IEEE, Vol. 88, No. 12, December 2000.

[2] J. Olenšek, T. Tuma, J. Puhan, Á. Bűrmen, *A new asynchronous parallel global optimization method based on simulated annealing and differential evolution*. Applied Soft Computing, Januar 2011.

[3] P. Preux, E. G. Talbi, *Towards hybrid evolutionary algorithms*. International Transactions in Operational Research 6, p.557–570, 1999.

[4] U. M. Garcia-Palomares, F. J. Gonzales-Casta, *A combined global and local search CGLS approach to global optimization*. Journal of Global Optimization 34, p. 409–526, 2006.

[5] H. Schmidt, G. Thierauf, *A combined heuristic optimization technique*. Advances in Engineering Software 36, p. 11–19, 2005.

[6] Y. T. Kao, Z. E., *A hybrid genetic algorithm and particle swarm optimization for multimodal functions*. Applied Soft Computing 8 (2), p. 849–857, 2008.

[7] Degrauwe, M. et al., *IDAC: An Interactive Design Tool for Analog Integrated Circuits*. IEEE Journal of Solid State Circuits, 1987.

[8] R. Harjani, R. A. Rutenbar, L. R. Carley, *OASYS: A Framework for Analog Circuit Synthesis*. IEEE Transactions on Computer Aided Design, 1989.

[9] H. Y. Koh, C. H. Sequin; P. R. Gray, *OPASYN: A Compliler for CMOS Operational amplifiers*. IEEE Transactions on Computer Aided Design, 1990.

[10] W. Kruiskamp in D. Leenaerts, *DARWIN: Analogue Circuit Syhesis Based on Genetic Algorithms*. International Journal of Circuit Theory and Applications, Vol. 23, 285-296, 1995.

[11] J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, *Automated WYWIWYG Design of Both the Topology and Component Values of Electrical Circuits Using Genetic Programming*. Proc. Firts Annual Conf. Genetic Programming. Stanford, CA: Stanford Univ., July 28-31, 1996.

[12] J. D. Lohn in S. P. Colombano, *A Circuit Representation Technique for Automated Circuit Design*. IEEE Trans. Evol. Comput., Vol. 3, No. 3, September 1999.

[13] T. McConaghy, P. Palmers, M. Steyaert in G.G.E. Gielen, *Trustworthy Genetic Programming-Based Synthesis of Analog Circuit Topologies Using Hierarchical Domain-Specific Building Blocks*. IEEE Transactions On Evolutionary Computation, Vol. 15, No. 4, August 2011.

[14] Miller J.F., Job D., Vassilev V.K., *Principles in the Evolutionary Design of Digital Circuits - Part I*. Genetic Programming and Evolvable Machines, 1, 8-35, 2000.

[15] J. R. Koza, F. H. Bennet, D. Andre, M. A. Keane in F. Dunlap, *Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming*. IEEE Trans. Evol. Comput., Vol. 1, No. 2, July 1997.

[16] T. Sripramong, C. Toumazou, *Concept and Techniques of Automatic Analogue Circuit Design using Genetic Programming on Affordable Computer System*. 1998.

[17] R. Poli, W. B. Langdon, N. F. McPhee, *A Field Guide to Genetic Programming*. Department of Computing and Electronic Systems, University of Essex - UK, p. 101 - 108, 2008.

[18] G. Györök, *Crossbar Network for Automatic Analog Circuit Synthesis*. SAMI 2014, IEEE 12th Syposium on Applied Machine Intelligence and Informatics, 2014.

[19] C. A. Anderson, K. F. Jones, J. Ryan, *A Two-Dimensional Genetic Algorithm for the Ising Problem*. Complex Systems, Vol. 5, p. 327-333, 1991.

[20] Ming-Wen Tsai, Tzung-Pei Hong, Woo-Tsong Lin, *A two-dimensional Genetic Algorythm and Its Application to Aircraft Scheduling Problem*. Hindawi Publishing Corporation, Mathematical Problems in Engineering, Vol. 2015, Article ID 906305, 2015.

[21] M. Mitchell, *An Introduction to Genetic Algorithms*. A Bradford Book The MIT Press, p. 128-135, 1999.

[22] Á. Bűrmen, J. Puhan, J. Olenšek, I.Fajfar, T. Tuma, *SPICE OPUS – A SPICE engine for OPtimization UtilitieS.*. Available: http://www.spiceopus.si/, EDA Laboratory, Faculty of Electrical Engineering, University of Ljubljana, 2009.

[23] Á. Bűrmen, J. Puhan, J. Olenšek, G. Cijan, T. Tuma, *PyOPUS - Simulation, Optimization, and Design*. Available: http://fides.fe.uni-lj.si/pyopus/index.html, EDA Laboratory, Faculty of Electrical Engineering, University of Ljubljana, 2016.