Original scientific paper

Informacije MIDEM
Journal of Microelectronics, Electronic Components and Materials
Vol. 46, No. 2(2016), 57 – 64

Population Ranking Based Differential evolution with Simulated Annealing for Circuit Optimization

Jernej Olenšek, Árpád Bűrmen

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

Abstract: Finding the values of circuit parameters for which the resulting circuit satisfies the design requirements can be formulated as an optimization problem. This problem is often solved using global optimization algorithms that provide some guarantee the resulting solution is the best possible one provided that the algorithm is given sufficient time. Unfortunately, these algorithms are slow and require many circuit evaluations. One of the algorithms proposed in our past research is PSADE that combines the favorable properties of simulated annealing and differential evolution and was shown to be a fast and reliable tool for solving circuit optimization problems. To make PSADE faster we replace the Metropolis criterion for accepting a trial point with one that is based on population ranking. The proposed algorithm retains its highly parallel nature. We tested the algorithm on a set of mathematical test functions and on a real-world circuit optimization problem. The results on the analog circuit sizing case show that the modified algorithm is more efficient and reliable than PSADE and some other global optimization methods.

Keywords: differential evolution; simulated annealing; population ranking; global optimization;, circuit design

Optimizacija vezij z diferencialno evolucijo in simuliranim ohlajanjem na osnovi rangiranja populacije

Izvleček: Iskanje vrednosti parametrov, pri katerih vezje zadosti načrtovalskim zahtevam, lahko predstavimo kot optimizacijski problem, ki ga pogosto rešujemo z globalnimi optimizacijskimi postopki. Ti nam zagotavljajo, da bomo našli najboljšo možno rešitev pod pogojem, da ima postopek na razpolago dovolj časa. Na žalost tovrstni postopki zahtevajo veliko simulacij vezja. Eden od postopkov, ki smo jih razvili, je postopek PSADE, ki združuje ugodne lastnosti diferencialne evolucije in simuliranega ohlajanja in se je izkazal kot hiter in zanesljiv pri optimizaciji vezij. Da bi postopek pospešili, smo zamenjali Metropolisov kriterij za sprejem točk s kriterijem, ki temelji na rangu točke znotraj populacije. Dobljeni postopek lahko zelo učinkovito paraleliziramo, saj obdrži vse ugodne lastnosti postopka PSADE. Preizkusili smo ga na naboru matematičnih funkcij in na praktičnem primeru iz načrtovalske prakse. Rezultati kažejo, da je pri načrtovanju vezij predlagani postopek bolj učinkovit in zanesljiv kot nekatere druge znane globalne optimizacijske metode

Ključne besede: diferencialna evolucija; simulirano ohlajanje; rangiranje populacije; globalna optimizacija; načrtovanje vezij

* Corresponding Author's e-mail: jernej.olensek@fe.uni-lj.si

1 Introduction

Choosing the values (parameters) of circuit components (also referred to as circuit sizing) with the goal of satisfying the design requirements can be formulated as an optimization problem by introducing a cost function (CF) that reflects the quality of the circuit in a real number. Finding the best performing circuit reduces to finding the minimum of a CF. Unfortunately real-world CFs have many local minima. Finding the best local minimum is a computationally hard problem that is addressed with global optimization algorithms.

Many global optimization algorithms were devised in the past. Some of the most successful mimic the evolution (e.g. [1, 2]) and behavior (e.g. [3, 4]) of living beings and physical processes (e.g. [5, 6]). Simulated annealing (SA) ([7, 8]) was one of the first global optimization algorithms drawing its inspiration from the process of cooling a metal. Due to its nature this algorithm is capable of finding a global minimum, albeit with a large number of CF evaluations.

Due to the so-called "no free lunch" theorems [9] the research in the area of global optimization started to focus on hybrid algorithms (e.g. [10, 11]). In our past research we hybridized SA with differential evolution (DE) [12] which resulted in the parallel simulated annealing and differential evolution algorithm (PSADE) [14] that exhibited good performance on mathematical test functions, as well as, real world circuit design problems. Due to the SA component of PSADE it can mathematically be proven that the algorithm converges to a global minimizer given a sufficiently large number of CF evaluations. PSADE is highly parallelizable which makes it possible to significantly speed up the optimization when multiple processors are available.

PSADE has the same drawbacks as other heuristic optimization algorithms. Most notable is the possibility of premature convergence to a local minimizer. The main cause of this drawback is the Metropolis acceptance criterion of SA which often rejects trial points that would otherwise lead the algorithm away from a local minimizer. Furthermore, the Metropolis acceptance criterion is based on an artificial parameter (temperature) that is based on the CF value. Because the CF value can differ to a great extent between similar optimization problems the acceptance criterion can be misguided into rejecting promising points.

To counter this drawback and simplify the algorithm we propose a different approach for accepting points based on the solution ranking within the population. The rank of the point is determined by sorting the points according to their CF value. We also assign every individual in the population a separate position. In turn, every position is assigned a set of parameters for SA and DE. Parameters corresponding to higher positions have a greater probability to be used in the process of constructing and evaluating a trial point. The algorithm tries to align the positions of individuals within the population with their ranks. This alignment is occasionally broken to increase the probability of accepting an inferior solution which may lead the algorithm away from a local minimum. We deem the proposed algorithm DESAPR (DE and SA with Population Ranking).

The paper is divided as follows. Section 2 outlines the proposed approach. The asynchronous parallel version of the algorithm is the subject of section 3. The optimization results for mathematical test function are given in section 4 while section 5 presents the results obtained on a real-world circuit optimization problem. Section 6 concludes the paper.

Notation. Vectors are denoted by bold lowercase letters. The *i*–th component of vector $\boldsymbol{a}$ is denoted by a_i. Inequalities are applied to vectors component-wise. The realization of a uniformly distributed random number from interval (0,1) is denoted by *U*(0,1).

2 The proposed method

The problem subject to optimization can mathematically be formulated as

$$\begin{aligned} &\boldsymbol{x}^* = \operatorname{argmin}_{x\in S} f(\boldsymbol{x}) \\ &f : S \to \mathbb{R} \\ &S = \left\{\boldsymbol{x} : \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{L} \le \boldsymbol{x} \le \boldsymbol{U}\right\} \end{aligned} \tag{1}$$

where *f* is the CF, *N* is the number of optimization variables, and ***L*** and ***U*** are vectors of lower and upper bounds imposed on these variables. The outline of the DESAPR is given by Algorithm 1.

Algorithm 1: DESAPR outline.

1. Initialization.
2. Competition.
3. Selection of parameters.
4. Trial point generation.
5. Trial point evaluation.
6. Replacement of a point in the population.
7. Local search.
8. If termination condition is not met, go back to step 2.

The population consists of *M* individuals. It is initialized by dividing each of the *N* parameter ranges given by vectors ***L*** and ***U*** uniformly into *M* subintervals. For every variable the *M* subintervals are assigned randomly to *M* individuals. The value of a variable for an individual is then chosen by randomly selecting a point in the assigned subinterval.

We denote the available positions with $i = 0,1, \ldots, M-1$. The behavior of the algorithm is determined by the weight factor *W*, crossover probability P_X, and random step probability distribution width parameter η. A set of parameter values is assigned to every one of the *M* available positions. The values of parameters assigned to *i*-th position are

$$W_i = W_0 e^{-c_W i}$$
$$P_{X,i} = P_{X,0} e^{-c_{P_X} i}, \tag{2}$$
$$\eta_i = e^i - 1$$

Coefficients c_W and c_{P_X} are computed from the weight factors and crossover probabilities of positions 0 and *M*-1 which are user defined parameters of the algorithm.

$$c_W = (M-1)^{-1} \ln \frac{W_0}{W_{M-1}}$$
$$c_{P_X} = (M-1)^{-1} \ln \frac{P_{X,0}}{P_{X,M-1}} \tag{3}$$

Every individual $\boldsymbol{x}_i$ is assigned to one of the *M* available positions. Let p_i and r_i denote the position and the rank of *i*-th individual. The rank of an individual is determined by ordering the individuals according to the CF value $f_i = f(x_i)$ and assigning ranks from M-1 (for the lowest CF value) to *0* (for the highest CF value). The goal of the competition in step 2 of Algorithm 1 is to align the rank of the individuals in the population with their positions. For this purpose two individuals are randomly selected from the population. Let *i* and *j* denote their indices. They exchange positions if $r_j > r_i$ and $p_j < p_i$. This forces individuals with high rank (low CF value) to move to high positions.

In step 3 of Algorithm 1 an individual is selected randomly with probability $P_{S,i}$.

$$P_{S,i} = \frac{e^{r_i}}{\sum_{i=0}^{M-1} e^{r_i}} \tag{4}$$

Let *k* denote the position of the selected individual. The weight factor (W_k), the crossover probability ($P_{X,k}$), and the range parameter (η_k) values assigned to this position are used in steps 4-7 of Algorithm 1. Individuals with high rank are selected with higher probability. Because higher ranking individuals tend to occupy higher positions, parameter values corresponding to higher positions are often used (but not always).

To simplify the notation the search space is transformed so that the components of vectors (i.e. individuals) lie within [0,1], where 0 and 1 correspond to the lower and the upper bound, respectively.

In step 4 of Algorithm 1 a trial point is generated using a modified DE operator [12] and polynomial mutation [13]. First a parent (***x***) and three additional distinct individuals (***u***, ***v***, and ***w***) are selected. The DE operator is applied component wise with probability $P_{X,k}$ resulting in point ***y'*** with components

$$y_i^{'} = \begin{cases} u_i + W_k(v_i - w_i) & U(0,1) \le P_{X,k} \\ x_i & \text{otherwise} \end{cases}. \tag{5}$$

If ***y'*** violates the bounds the components that are outside the bounds (i.e. $y_i' \notin [0,1]$) are corrected resulting in point ***y*** with components defined as

$$y_i = \begin{cases} y_i^{'} & 0 \le y_i^{'} \le 1 \\ x_i + U(0,1)\cdot(1-x_i) & y_i^{'} > 1 \\ x_i - U(0,1)\cdot x_i & y_i^{'} < 0 \end{cases}. \tag{6}$$

Finally, polynomial mutation is applied to ***y*** component wise. For every component a random number $\alpha_i = U(0,1)$ is generated. The trial point ***z*** is then computed as

$$z_i = y_i + \begin{cases} 1 - \left[2 - 2\alpha_i + (2\alpha_i - 1) y_i^{\eta_k+1}\right]^{1/(\eta_k+1)} & \alpha_i > 0.5 \\ \left[2\alpha_i + (1-2\alpha_i)(1-y_i)^{\eta_k+1}\right]^{1/(\eta_k+1)} - 1 & \text{otherwise} \end{cases} \tag{7}$$

Let f_z denote the CF value corresponding to the trial point ***z***. The population along with the trial point is ordered according to the CF value. Rank 0 is assigned to the point with the highest CF value. Let r_z and r_x denote the ranks of the trial point and the parent, respectively. The trial point is accepted into the population (i.e. replaces the parent point ***x*** in step 6 of Algorithm 1) with probability

$$P = \min\left(1, e^{\frac{(r_z - r_x)k}{M-k}}\right) \tag{8}$$

The acceptance criterion resembles the original Metropolis criterion [7] in the sense that higher ranking trial points are accepted with higher probability. A trial point ranking higher than the parent point is always accepted.

Finally, in step 7 of Algorithm 1 a simple local search strategy is performed if the trial point is accepted, the parent point is the best point in the population, or with some small probability P_L (set to 0.05). Local search uses one of the points in the population as the origin and two additional points for computing a search direction (***d***). All three points are chosen randomly. Two additional points are evaluated along direction ***d*** and a quadratic model of the CF is computed. This model is minimized and the resulting point evaluated. The evaluated point with the lowest cost function value is the result of the local search. The resulting point replaces the parent point if its CF value is lower. The complete details of the local search can be found in [14].

3 Parallel implementation

Suppose one has *m* parallel processors available. Assuming most of the computational time is spent for evaluating the trial point and for local search, these two tasks are outsourced to parallel processors in an asynchronous manner. The main process (master) runs the following algorithm:

```
Algorithm 2: asynchronous parallel optimization
1.  Initialization.
2.  If no processor is idle go to step 5.
3.  Perform steps 2-4 of Algorithm 1 (generate a trial
    point).
        Send it to an idle processor ($p$) for evaluation.
        Remember the parent point ($x$) and the selected
        parameters position ($k$) for that processor.
4.  If there are idle processors left return to step3.
5.  Wait until one of the processors ($p$) finishes its
    task. Collect the results.
6.  If $p$ was performing global search point evaluation.
        Perform step 6 of Algorithm 1 (point replacement)
        using the parameters corresponding to
        position $k$ that was stored for $p$.
        If required, start a local search (step 7 of Algorithm 1)
        on processor $p$.
    If $p$ was performing local search.
        Replace the parent point of $p$ if local search
        found a better point.
7.  If termination condition is not met, go back to
    step 2.
```

Algorithm 2 performs multiple passes of Algorithm 1 in parallel and in this way accelerates the evolution of the population.

4 Performance on mathematical test functions

DESAPR was implemented within the framework of the PyOPUS library [15]. The performance of DESAPR was compared to that of PSADE, DE, SA, and JADE [18] on 13 test functions from [14]. For the sake of comparison, 30 optimization runs were performed for every function. We will later use the presented method for analog circuit sizing, where CF evaluations can take several seconds. Therefore we impose a CF evaluation budget of 100000 function evaluations per run to maintain reasonable optimization run times.

For DESAPR we used fixed parameter values in all experiments: $M=20$, $W_0=0.9$, $W_{M-1}=0.9$, $PX_0=0.9$, $PX_{M-1}=0.1$. We made no attempt to fine tune the parameter values to any specific problem. It is very time consuming especially for real world problems, where every CF evaluation can take considerable amount of time. The values were chosen based on our experience with evolutionary algorithms. High weight factor for DE and low crossover probability tend to maintain population diversity longer, which is desirable since DESAPR uses very small population. Fine tuning the parameters and introducing parameter adaptation or evolution could lead to even better performance for our method and is also subject of our future research.

For the compared methods, the parameters were selected according to guidelines from the authors of the methods. For DE we used DE/rand/bin strategy with population size 100, weight factor 0.5 and crossover probability 0.9. SA used in our experiments uses only two parameters. We set the final temperature and random step range parameter to T_{min}=1e-10, R_{min}=1e-10. For PSADE we set population size to 20, T_{min}=1e-10, R_{min} =1e-10, τ_1 = 0.01 (local step), τ_2 =0.1 (parameter adaptation). For JADE we also followed the author suggestions. We used the version without the archive, as suggested by the authors for problems with low dimensionality (< 30). We used the population size of 100, the learning parameter *c*=0.1 and the percentage of points considered as the best in population *p*=5%.

The results are given in Table 1.

For every function we chose a target CF value f_{target}, that lies in the basin of attraction of the global minimum. Finding this solution means that global search was successful, and any local search procedure can be used to quickly find the exact minimum. Not all runs succeed in reaching f_{target}. We report the success rate and the average number (over successful runs) of CF evaluations (#CF) needed to reach f_{target}. The average final CF error (with respect to the true global minimum) after 100 000 CF evaluations is listed in the *error* column. The best value of #CF and *error* are written in boldface if there exist a statistically significant difference between the best and second best method.

No method was able to reach f_{target} for all functions in all runs. JADE and DESAPR outperformed the other methods so we will focus on them. Considering final CF value, JADE outperformed DESAPR on 8 functions, while DESAPR was better only on 2 functions. On 3 functions there was no statistically significant difference. When

Table 1: Performance comparison results for 30D mathematical test functions. The function value at the global minimum is denoted by *f**. The results of the best performing algorithm are written in boldface if there exist a statistically significant difference to the second best method.

Function	*f**	f_{target}	#CF	Success rate [%]	*error*	Algorithm
f_1	0	10-10	39388	100	$9.74 * 10^{-17}$	DESAPR
Sphere			62347	100	$5.32 * 10^{-12}$	PSADE
			NA	0	$5.20 * 10^{-7}$	DE
			NA	0	$7.09 * 10^{-6}$	SA
			34000	100	**0.0**	JADE
f_2	0	0.1	**14477**	100	$5.19 * 10^{-7}$	DESAPR
Schwefel 2.22			47562	100	$4.91 * 10^{-2}$	PSADE
			58954	100	$1.73 * 10^{-4}$	DE
			78691	76	$4.30 * 10^{-3}$	SA
			16020	100	**0.0**	JADE
f_3	0	15	37693	100	$7.11 * 10^{-3}$	DESAPR
Schwefel 1.2			67121	100	0.811	PSADE
			NA	0	29.01	DE
			61051	100	$7.38 * 10^{-4}$	SA
			26334	100	$\mathbf{4.81 * 10^{-9}}$	JADE
f_4	0	0.1	**35228**	100	$\mathbf{4.07 * 10^{-5}}$	DESAPR
Schwefel 2.21			71209	100	0.073	PSADE
			94510	36	$2.94 * 10^{-1}$	DE
			73522	100	$1.18 * 10^{-3}$	SA
			88247	63	$9.22 * 10^{-2}$	JADE
f_5	0	30	18919	100	16.52	DESAPR
Rosenbrock			36599	100	21.46	PSADE
			57851	100	21.11	DE
			63210	70	92.07	SA
			17418	100	**4.28**	JADE
f_6	0	0	11149	100	0	DESAPR
Step			16151	100	0	PSADE
			42566	100	0	DE
			61547	100	0	SA
			10862	100	0	JADE
f_7	0	0.02	30999	100	$7.84 * 10^{-13}$	DESAPR
Noisy quartic			36211	100	$2.31 * 10^{-3}$	PSADE
			79544	57	$2.58 * 10^{-2}$	DE
			58044	73	$1.41 * 10^{-2}$	SA
			16225	100	$\mathbf{1.96 * 10^{-3}}$	JADE
f_8	-418.982887*30	-12569.45	**22049**	97	3.95	DESAPR
Schwefel 2.26	= -12569.486618		36955	93	7.90	PSADE
			NA	0	$7.59 * 10^{3}$	DE
			NA	0	$7.11 * 10^{2}$	SA
			77722	97	3.95	JADE
f_9	0	0.1	**30697**	100	$\mathbf{6.83 * 10^{-13}}$	DESAPR
Rastrigin			81588	100	$8.19 * 10^{-3}$	PSADE
			NA	0	144.23	DE
			NA	0	6.31	SA
			77644	100	$1.60 * 10^{-4}$	JADE
f_{10}	0	10^{-4}	31255	100	$2.27 * 10^{-7}$	DESAPR
Ackley			55982	100	$1.93 * 10^{-5}$	PSADE
			96542	83	$8.14 * 10^{-4}$	DE
			NA	0	0.71	SA
			27122	100	$\mathbf{4.44 * 10^{-16}}$	JADE
f_{11}	0	10^{-9}	42281	100	$1.37 * 10^{-13}$	DESAPR
Griewank			77545	100	$7.88 * 10^{-9}$	PSADE
			NA	0	$4.13 * 10^{-8}$	DE
			NA	0	$2.10 * 10^{-2}$	SA
			35385	100	$\mathbf{5.55 * 10^{-17}}$	JADE

f_{12} Penalty 1	0	10^{-10}	39953	100	$4.70 * 10^{-16}$	DESAPR
			52650	100	$3.31 * 10^{-16}$	PSADE
			NA	7	$1.71 * 10^{-9}$	DE
			NA	0	$4.94 * 10^{-8}$	SA
			32804	100	$\mathbf{3.77 * 10^{-32}}$	JADE
f_{13} Penalty 2	0	10^{-10}	43895	100	$1.65 * 10^{-14}$	DESAPR
			54261	100	$5.74 * 10^{-15}$	PSADE
			NA	0	$3.59 * 10^{-7}$	DE
			NA	0	$1.01 * 10^{-6}$	SA
			34960	100	$4.39 * 10^{-30}$	JADE

comparing the speed, JADE was faster on 7 functions, while DESAPR was better on 4. JADE displays a good fine tuning capabilities and fast convergence on unimodal and well behaved functions. However on the most difficult f_8 and f_9, that have many local minima, DESAPR was significantly better than JADE, regarding both the speed and the final solution quality. JADE and DESAPR also exhibit similar success rate.

5 Performance on a real-world design problem

We tested DESAPR by sizing a Miller operational trans conductance amplifier (OTA) (Figure 1)[16] across a large number of corner points. The bias current was set to 100μ*A*. The performance measures and the design requirements are listed in Table 2.

Every corner point is a combination of temperature (0°C, 25 °C, and 100 °C), operating voltage (1.7V, 1.8V, and 2.0V), and CMOS corner model (average, worst power, worst speed, worst one, and worst zero). Let $\mathcal{C}$ denote the set of 45 corner points obtained in this manner. Every design requirement must be met for all 45 corner points from set $\mathcal{C}$. The CF is a function of the design parameters ($\boldsymbol{x}_D$). It is constructed as a sum of contributions (CF_i) corresponding to individual performance measures.

$$CF(\boldsymbol{x}_D) = \sum CF_i(\boldsymbol{x}_D) \tag{9}$$

Let $p_i(\boldsymbol{x}_D)$, g_i, and n_i denote the worst value of a performance measure across corners from set $\mathcal{C}_i \subseteq \mathcal{C}$, the corresponding goal, and the corresponding norm, respectively. A contribution of a performance measure to the CF for which an upper bound is imposed (design requirement of the form $p_i(\boldsymbol{x}_D) \le g_i$) is computed as [17]

$$CF_i(\boldsymbol{x}_D) = \begin{cases} \dfrac{p_i(\boldsymbol{x}_D) - g_i}{n_i} & p_i(\boldsymbol{x}_D) \ge g_i \text{(the design requirement is not met)} \\ 10^{-6} \cdot \dfrac{p_i(\boldsymbol{x}_D) - g_i}{n_i} & \text{otherwise} \end{cases} \tag{10}$$

For performance measures with design requirements of the form $p_i(\boldsymbol{x}_D) \ge g_i$ the roles of $p_i(\boldsymbol{x}_D)$ and g_i in equation (10) are exchanged. By default the norm is equal to the goal. If the goal is 0, the norm is set to 1. An exception is the circuit area for which the norm is set to 100μm². Constructing the cost function in this manner penalizes designs that fail to satisfy the design requirements with a positive CF contribution while rewarding designs that exceed the design requirements with a small negative CF contribution.

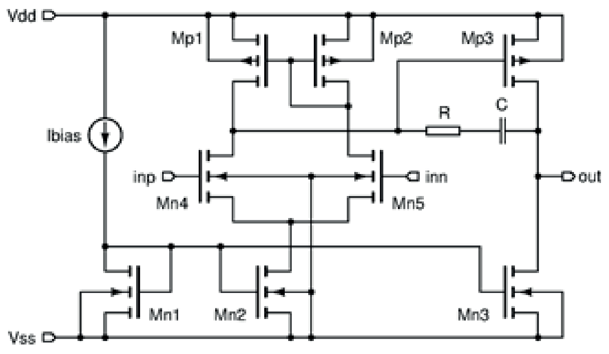

Figure 1: Miller OTA.

The optimizer tries to find the minimum of the CF by tuning the 13 design parameters (11 transistor channel widths and lengths, the resistance of R and the capacitance of C). The optimizer stops as soon as all design requirements are satisfied.

Table 2: Design requirements and circuit analyses from which the performance of the Miller OTA is evaluated.

Performance measure	Goal	Required analyses
Supply current [μA]	≤ 200	op
Vgs overdrive [V]	≥ 0.0	op
Vds overdrive [V]	≥ 0.1	op
Output swing [V]	≥ 1.2	dc
Gain [dB]	≥ 60	ac
UGBW [MHz]	≥ 30	ac
Phase margin [o]	≥ 50	ac
PSRR Vdd [dB]	≥ 65	ac, acvdd

PSRR Vss [dB]	≥ 65	ac, acvss
CMRR [dB]	≥ 90	ac, accom
Settling time (up) [ns]	≤ 100	tran
Settling time (down) [ns]	≤ 100	tran
Overshoot (up) [%]	≤ 10	tran
Overshoot (down) [%]	≤ 10	tran
Slew rate (up) [V/μs]	≥ 10	translew
Slew rate (down) [V/μs]	≥ 10	translew
Circuit area [μm^2]	≤ 1000	-

Because evaluating one point in the design space requires the circuit to be evaluated across all corner points from $\mathcal{C}$ a strategy for reducing the number of circuit evaluations was used. The circuit was optimized in multiple passes where the solution of k–th pass ($\boldsymbol{x}_{D,k}$) was used as the initial point for pass k +1. In the beginning of k–th pass all performance measures were evaluated across all corners from $\mathcal{C}$ at the initial point $\boldsymbol{x}_{D,k-1}$. Let $c_i \in \mathcal{C}$ denote the corner point where the i–th performance measure reached its worst value. If this worst performance did not satisfy the corresponding design requirement corner point c_i was added to $\mathcal{C}_i$. If no corner was added to any of the corner point subsets, no further passes were performed. If the resulting circuit satisfied all design requirements the run was deemed as successful. If, however, a corner point was added to any of the corner point subsets, the CF was minimized using an optimization algorithm starting from $\boldsymbol{x}_{D,k-1}$ which resulted in point $\boldsymbol{x}_{D,k}$. In most cases the final corner subsets contained only a handful of corners where the circuit exhibited its worst performance. Therefore the number of circuit evaluations was much lower compared to the brute force approach where every point in the design space is evaluated across all 45 corners.

Population based algorithms were started from a given initial point $\boldsymbol{x}_{D,k}$ by replacing one member of the initial population with $\boldsymbol{x}_{D,k}$. Optimization was stopped as soon as all design requirements were satisfied across the corresponding corner point sets $\mathcal{C}_i$ or if the number of evaluated circuits exceeded 50000. Four optimization algorithms were tested: differential evolution (DE), PSADE [14], DESAPR and JADE. SA was not included in this test because its performance on mathematical test functions considering the average final CF value was significantly worse than the performance of DE. Due to the stochastic nature of the tested algorithms the circuit was optimized 10 times on a cluster of 100 processors. For every algorithm the final CF value, the run time, and the number of performed circuit analyses was recorded. The minimal, the maximal, and the average results are listed in Table 3.

In terms of the final CF value DE and JADE failed to find a circuit satisfying all design requirements, despite many more CF evaluations. DESAPR and PSADE both succeeded in finding such a circuit in all 10 optimizations. The final CF value found by PSADE was slightly better, although this is not relevant because the optimization was stopped as soon as a circuit satisfying all design requirements was found and there was no real competition between the algorithms in terms of finding the best possible circuit. In terms of computational time DESAPR was on average two times faster than PSADE. The same can be said about the number of circuit analyses.

6 Conclusion

Finding the values of the circuit's design parameters is an important task in analog design automation. Global optimization algorithms are often selected for this task due to their ability to find the best possible circuit. Un-

Table 3: Performance comparison results for the Miller OTA. The best results (CF value and time) are written in boldface.

		CF	Time [s]	# op	# dc	# ac	# acvss	# acvdd	# accom	# tran	# translew
DE	Min	1.76e-02	7.42e+03	597390	132382	999350	221375	241473	261571	528424	264396
	Max	4.97e-01	2.95e+04	2358660	381896	5036254	2105616	2280094	2396099	2029708	1133701
	Avg	8.66e-02	1.96e+04	1453050.4	271884.2	3049903.2	1004943.8	1080426.9	1134624.7	1247288.0	670108.5
PSADE	Min	-3.99e-05	6.59e+02	34722	9180	45739	9180	12955	14976	39943	21951
	Max	-3.50e-05	3.59e+03	180440	42717	365833	112328	95558	119591	259453	124762
	Avg	-3.77e-05	1.92e+03	107708.4	24171.2	172718.2	48279.7	46172.2	55005.7	136731.1	66726.0
DESAPR	Min	-3.96e-05	4.08e+02	19222	5750	31718	6225	8090	8090	24758	13626
	Max	-3.47e-05	1.24e+03	69940	15284	117381	29909	35803	39533	75463	54541
	Avg	-3.70e-05	9.24e+02	46279.4	10833.1	71826.0	19131.2	20380.1	23858.7	53771.4	32792.2
JADE	Min	8.67E-03	2.55E+05	3806920	688720	10603220	5299470	5299470	5299470	5989420	3018020
	Max	5.74E-02	4.08E+05	8380226	1280426	27604226	11204410	11204410	11062510	11054026	5803526
	Avg	2.78E-02	3.06E+05	5.70e+06	9.55e+05	1.77e+07	7.56e+06	7.59e+06	7.51e+06	8.00e+06	4.23e+06

fortunately these algorithms are quite slow. We propose a modification of the PSADE global optimization algorithm that replaces the original Metropolis criterion of simulated annealing with a ranking based criterion. By replacing the acceptance criterion we hope to avoid situations where the algorithm gets caught in the neighborhood of a local minimum. The proposed algorithm (DESAPR) is highly parallelizable. We tested the algorithm on a set of mathematical test functions and on a real-world circuit design problem. The results confirmed, that DESAPR is an efficient and reliable optimizer for analog circuit sizing problem.

7 Acknowledgements

The research was co-funded by the Ministry of Education, Science, and Sport (Ministrstvo za Šolstvo, Znanost in Šport) of the Republic of Slovenia through the programme P2-0246 Algorithms and optimization methods in telecommunications.

8 References

1. JY Sun, QF Zhang, and EPK Tsang. DE/EDA: A new evolutionary algorithm for global optimization. Information Sciences, 169(3-4):249–262, 2005.
2. R Chelouah and P Siarry. A continuous genetic algorithm designed for the global optimization of multimodal functions. Journal of Heuristics, 6(2): 191–213, 2000.
3. Chuen Tse Kuah, Kuan Yew Wong, and Manoj Kumar Tiwari. Knowledge sharing assessment: An Ant Colony System based Data Envelopment Analysis approach. Expert Systems with Applications, 40(8):3137–3144, 2013.
4. Bolun Chen, Ling Chen, and Yixin Chen. Efficient ant colony optimization for image feature selection. Signal Processing, 93(6, SI):1566–1576, 2013.
5. J Kennedy and R Eberhart. Particle swarm optimization. IEEE International Conference on Neural Networks Proceedings, 1-6: 1942–1948, 1995.
6. JJ Liang, AK Qin, PN Suganthan, and S Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation, 10(3):281–295, 2006.
7. S Kirkpatrick, CD Gelatt, and MP Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.
8. D. R. Thompson and G. L. Bilbro. Sample-sort simulated annealing. IEEE Transactions on System, Man, and Cybernetics (B), 35(3):625–632, 2005.
9. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1):67–82, 1997.
10. A. Kaveh and S. Talatahari. Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures. Computers & Structures, 87(5-6):267–283, 2009.
11. Ali Riza Yildiz. A novel hybrid immune algorithm for global optimization in design and manufacturing. Robotics and Computer-Integrated Manufacturing, 25(2):261–270, 2009.
12. R Storn and K Price. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 11(4):341–359, 1997.
13. M Hamdan. A dynamic polynomial mutation for evolutionary multi-objective optimization algorithms. International Journal on Artificial Intelligence Tools, 20(1):209–219, 2011.
14. Jernej Olenšek, Tadej Tuma, Janez Puhan, and Árpád Bűrmen. A new asynchronous parallel global optimization method based on simulated annealing and differential evolution. Applied Soft Computing, 11(1):1481–1489, 2011.
15. "PyOPUS - Circuit Simulation and Optimization", available at http://fides.fe.uni-lj.si/pyopus/, 2015.
16. R.J. Baker, CMOS Circuit Design, Layout, and Simulation, Wiley-IEEE Press, Hoboken (NJ), 2007.
17. Á Bűrmen, D Strle, F Bratkovič, J Puhan, I Fajfar, T Tuma. Automated robust design and optimization of integrated circuits by means of penalty functions. AEU-International journal of electronics and communications 57 (1), 47-56, 2003.
18. Jingqiao Zhang and Arthur C. Sanderson. JADE: Adaptive Differential Evolution with Optional External Archive. IEEE Transactions on evolutionary computation, 13 (5), 945-958, october 2009.

Arrived: 18. 12. 2015
Accepted: 13. 06. 2016