Towards Automated Worst-Case Analysis of Circuits: Selecting Initial Values for Global Optimization

Kristóf Horváth, Balázs Bank, György Orosz
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: {hkristof, bank, orosz}@mit.bme.hu

***Abstract*—Worst-case circuit analysis is a mandatory practice in hardware verification and validation. To this end, several methods, including extreme value analysis (EVA) and Monte Carlo analysis are commonly used, however, each has its own limitations. Numerical optimization-based methods have the potential to be generally usable, but have the tendency to get stuck in a local minimum, which can be mitigated using carefully chosen initial values. In this paper we propose methods for automated initial value selection for black-box circuit models. The methods are demonstrated to work on several standard test functions, which is a first step in building an automated worst-case circuit analysis tool.**

***Index Terms*—worst-case circuit analysis, numerical optimization, initial values**

I. Introduction

Worst-case analysis (WCA) is an important step in hardware design verification [1]. Because the parameters of the circuit components can vary in production, certain circuit properties (e.g. amplifier gain, attenuation, power dissipation, etc.) should be analysed to make sure that parameter variation will not push the circuit outside its limits.

For this purpose, various methods have been developed over the years. Among them, extreme value analysis (EVA) is a popular tool in the industry. EVA is based on the assumption that the extreme values of circuit properties can be found on the boundaries of parameter values, therefore by evaluating all extreme-value combinations it is possible to find those which result in the highest deviation from nominal circuit properties. The drawback of EVA is that it requires 2^N function evaluation where N is the number of parameters. In addition, for some circuits the assumption of having the extreme value at the boundaries is not valid (see Figure 2), and thus EVA leads to incorrect results.

Monte-Carlo-analysis is another traditional method, where simulations are used to calculate a probability distribution estimator for the circuit property in question. It assumes that the circuit property is an approximately linear function of the component values, which might be a good approximation for many, but not for all parameters. The downsides of this method include a large computational demand as well as the fact that the results are not exact [2].

A more general approach to worst-case analysis is to perform it as an optimization task [3]. After all, the goal to WCA is to find the extreme values of a circuit property, which can be represented as a mathematical function. Tolerances of the component parameters can be incorporated into the optimization problem as boundaries to the search space.

Traditional optimization methods (e.g. gradient descent, direct methods, etc.) assume cost functions that have only one local minimum in the search space. These algorithms operate from a starting point, and make incremental steps that converge to a local optimum. In some circuit analysis problems, however, the function describing the circuit property in question can have multiple local optima, thus global optimization strategies are necessary. One approach is to sample the function and use clustering to select candidate points to be used as initial values and start a traditional local optimization algorithm from all of them [4], [5]. This way there will be at least one point from where the local optimizer can converge to the global optimum.

An additional difficulty in circuit analysis is that functions rarely appear in closed-form, rather than as a result of a Spice-based circuit simulation. Therefore, we only consider black-box functions for optimization targets, whose derivatives can not be evaluated.

In this paper we present methods for searching appropriate initial values for global optimization of black-box functions.

II. Example circuit

Consider the circuit in Figure 1. The objective of the analysis is the maximum power dissipation of transistor Q2. The tolerances for resistors are 5% of their nominal value, the transistors are used at temperatures between 0°C and 40°C and other parameters and tolerances came from the datasheet. The supply voltage, denoted by VCC, can be between 6 and 12 Volts, and the input voltage, marked by Uin, is allowed to be between 0 and 5.5 Volts. The number of parameters in this example is 14.

Using physical considerations, we have deduced that the power dissipation of Q2 is largely dependent on the input voltage of the circuit. The relationship is plotted in Figure 2 with all other parameters being fixed at the results of a preliminary worst-case analysis. It should be noted that there are two (local) maxima in this graph; at around 4 V and around 1 V, therefore global optimization is necessary.

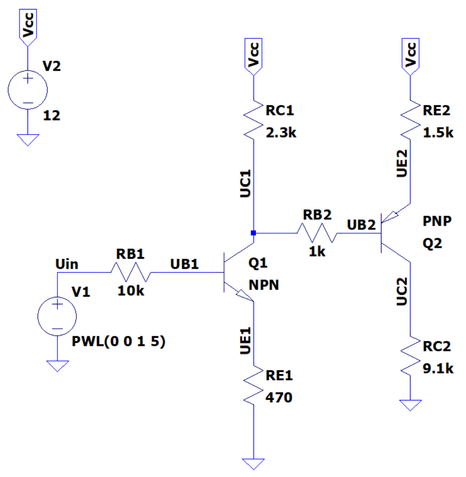

Fig. 1. A simple transistor-based circuit example. The property in question is the power dissipation of transistor Q2.

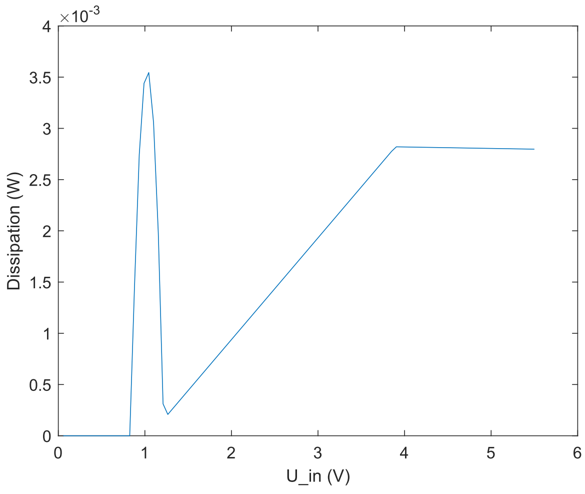

Fig. 2. Dissipation of the Q2 transistor in the example. Here, we only plotted the dissipation in terms of the input voltage, with the other parameters being fixed.

III. Methods for determining a set of initial values

Usually, initial values are chosen based on nominal values [3], or results of a pen-and-paper-based analysis. Another approach is to start the optimization from a random initial value. When the optimization is performed from several random initial values, it is likely that the optimization algorithm will converge to the global optimum from at least one set of initial values. The downside of using a large number of initial values is the excessive amount of redundant computations.

Our approach of determining a set of initial values is based on function sampling. The core idea has been used for empirical estimation of function shape and properties [6], [7] as well as by global optimization methods [4], [5]. The latter sample the function in a large number of random points and select only a few of them. Ideally, only one point is selected from the region of attraction around each local minimum, because selecting more than one will lead to unnecessary calculations in the subsequent local optimization step.

Obviously, an important condition for this approach is the proper sampling of the function of interest: for example, if the sample point set does not contain any points from the vicinity of a certain local minimum, then sampling-based methods can not find that local minimum.

Algorithm 1 Barrier search (BS) algorithm

```
1:  Generate random sampling points: $\{\mathbf{x}_i\}$ for $i = 1 \ldots M$
2:  Evaluate function in sampling points: $y_i = f(\mathbf{x}_i)$ for $\forall \mathbf{x}_i$
3:  Make an ordered list: $f(\mathbf{x}_{o_1}) \leq f(\mathbf{x}_{o_2}) \leq \ldots \leq f(x_{o_M})$
4:  Add first element to output set: $D = \{\mathbf{x}_{o_1}\}$ where $\mathbf{x}_{o_1} = \arg\min_{\mathbf{x}_i}(f(\mathbf{x}_i))$
5:  for all $\mathbf{x}_{o_k}$, $k = 2 \ldots M$ do
6:      for all $\mathbf{x}_{d_i} \in D$ do
7:          Define midpoint as: $\mathbf{m}_{i,k} = (\mathbf{x}_{d_i} + \mathbf{x}_{o_k})/2$
8:          if $f(\mathbf{x}_{d_i}) < f(\mathbf{m}_{i,k})$ and $f(\mathbf{x}_{o_k}) < f(\mathbf{m}_{i,k})$ then
9:              $D := D \cup \{\mathbf{x}_{o_k}\}$
10:         else if $f(\mathbf{m}_{i,k}) < f(\mathbf{x}_{d_i})$ then
11:             Replace $\mathbf{x}_{d_i}$ in $D$ with $\mathbf{m}_{i,k}$
12:             Restart inner loop
13:         end if
14:     end for
15: end for
```

IV. Proposed algorithms for initial value selection

Our intention in constructing our algorithms was to keep them simple, yet at the same time use the least amount of function evaluations at searching suitable initial values for global optimization.

As a first step, all algorithms generate an arbitrary number of random sampling points, $\mathbf{x}_i$, and evaluate the objective function at these points: $y_i = f(\mathbf{x}_i)$. The output of each algorithm is a set of initial values, which we denote by $D = \{\mathbf{x}_{d_1}, \mathbf{x}_{d_2}, \ldots, \mathbf{x}_{d_K}\}$, where $d_{1 \ldots K}$ denote the indices selected as initial values from set $\{\mathbf{x}_i\}$.

Barrier search algorithm (BS) is based on the fact that two local minima should be separated by a barrier, i.e. there should be an area between two local minima where the function value is higher than any of the two minima. In practice, the condition is checked only at the midpoint between two testpoints. The pseudocode for BS algorithm can be found in Algorithm 1.

Convex definition check (CDC) algorithm is based on the necessary condition for convexity: in convex areas, any line segments connecting two points on the graph of the function lies above the graph between the two points. In practice, this condition is checked only at the midpoint between two testpoints. The pseudocode for CDC algorithm can be found in Algorithm 2.

Both the BS and CDC algorithms perform a simple preliminary optimization too: if the function value at the midpoint

Algorithm 2 Convex definition check (CDC) algorithm

```
1: Generate random sampling points: $\{\mathbf{x}_i\}$ for $i = 1 \ldots M$
2: Evaluate function in sampling points: $y_i = f(\mathbf{x}_i)$ for $\forall \mathbf{x}_i$
3: Make an ordered list: $f(\mathbf{x}_{o_1}) \leq f(\mathbf{x}_{o_2}) \leq \ldots \leq f(x_{o_M})$
4: Add first element to output set: $D = \{\mathbf{x}_{o_1}\}$ where $\mathbf{x}_{o_1} = \arg\min_{\mathbf{x}_i}(f(\mathbf{x}_i))$
5: for all $\mathbf{x}_{o_k}$, $k = 2 \ldots M$ do
6:     for all $\mathbf{x}_{d_i} \in D$ do
7:         Define midpoint as: $\mathbf{m}_{i,k} = (\mathbf{x}_{d_i} + \mathbf{x}_{o_k})/2$
8:         if $f(\mathbf{m}_{i,k}) > (f(\mathbf{x}_{d_i}) + f(\mathbf{x}_{o_k}))/2$ then
9:             $D := D \cup \{\mathbf{x}_{o_k}\}$
10:        else if $f(\mathbf{m}_{i,k}) < f(\mathbf{x}_{d_i})$ then
11:            Replace $\mathbf{x}_{d_i}$ in $D$ with $\mathbf{m}_{i,k}$
12:            Restart inner loop
13:        end if
14:    end for
15: end for
```

Algorithm 3 Local minimum definition (LMD) algorithm

```
1: Generate random sampling points: $\{\mathbf{x}_i\}$ for $i = 1 \ldots M$
2: Evaluate function in sampling points: $y_i = f(\mathbf{x}_i)$ for $\forall \mathbf{x}_i$
3: Make an ordered list: $f(\mathbf{x}_{o_1}) \leq f(\mathbf{x}_{o_2}) \leq \ldots \leq f(x_{o_M})$
4: Add first element to output set: $D = \{\mathbf{x}_{o_1}\}$ where $\mathbf{x}_{o_1} = \arg\min_{\mathbf{x}_i}(f(\mathbf{x}_i))$
5: for all $\mathbf{x}_{o_k}$, $k = 2 \ldots M$ do
6:     Find closest L points: $\|\mathbf{x}_{o_k} - \mathbf{x}_{c_1}\|_2 \leq \|\mathbf{x}_{o_k} - \mathbf{x}_{c_2}\|_2 \leq \cdots \leq \|\mathbf{x}_{o_k} - \mathbf{x}_{c_L}\|_2$
7:     if $f(\mathbf{x}_{o_k}) < f(\mathbf{x}_{c_i})$ for $\forall i = 1 \ldots L$ then
8:         $D := D \cup \{\mathbf{x}_{o_k}\}$
9:     end if
10: end for
```

($m_{i,k}$) is lower than the function value at the point in the output set (x_{d_i}), then the latter is replaced by the midpoint.

The third algorithm (Local minimum definition, LMD) is inspired by the definition of local minimum. For each sample point, the closest N points are considered: if the tested point has the lowest function value out of the closest L sampling points then it is assumed to be a local minimum. The pseudocode for LMD can be found in Algorithm 3. Note that LMD does not need additional function evaluations besides the initial sampling points.

V. Performance evaluation

In order to compare the proposed initial value search algorithms, we have tested them on some standard 2-dimensional test functions [8]: sphere, inverse sphere, Rosenbrock, Styblinski-Tang, Goldstein-Price, Booth, Matyas, Himmelblau, Three-hump camel, and McCormick functions. The reason for choosing these functions is that their properties are well-known, and the functions themselves are well-behaved and closely resemble typical physical systems that are analysed in WCA problems.

In our tests, we have sampled the functions in 1000 random points and used these samples for all tested algorithms. We have found that considering the closest 10 points in LMD algorithm leads to the lowest amount of redundant initial values. To measure performance, the number of initial values and the total number of function evaluations were used. The tests were performed on 30 different random point sets and the numbers of initial values returned by each algorithm were averaged. For finding the closest local optimum from each initial point, we have used Matlab's `fmincon` interior-point method as local optimizer.

Figure 3 shows the ratio between the average number of initial values returned by the algorithms and the number of real local minima of the functions. Additionally, the performances on the example circuit in Figure 1 are also shown. In all cases, the local optimization converged to either of the local minima, so a ratio of 1 means that the algorithm managed to find the region of attraction around each local minimum. Values higher than 1 mean that the algorithm returned more initial values than necessary, which lead to unnecessary calculations in the subsequent local optimization step. The BS algorithm could not find all of the local minima of Goldstein-Price function. This is caused by the fact that the function is relatively flat around its local minima and so there is a significant chance that the algorithm can not find a barrier between sampling points.

A box plot containing the required number of function evaluations can be found in Figure 4. It can be seen that the CD algorithm that is based on the definition of convexity requires more function evaluations than the other two in more than 50% of the cases. This is caused by the large number of initial points, as it returns all sampling points in a concave area of the function.

Probably the most surprising is that the LMD algorithm requires the least amount of function evaluations. This is because despite returning the highest number of initial points in the majority of the cases, it only needs to evaluate the function at the sampling points, which in our case is 1000 points. In later experiments we have found that this advantage diminishes as the number of dimensions increases.

VI. Multidimensional example

We have tested our algorithms on a scalable multidimensional function too:

$$f(\mathbf{x}) = -\sum_{i=1}^{N} \cos(2\pi x_i), \quad \forall x_i \in [0;1], \tag{1}$$

TABLE I
RATIO OF MISSED REGIONS OF ATTRACTION AND THE RATIO OF DUPLICATE POINTS IN REGIONS OF ATTRACTION FOR THE MULTIDIMENSIONAL FUNCTION EXAMPLE.

	Missed regions of attraction			Duplicate points in output		
Dim.	BS	CDC	LMD	BS	CDC	LMD
2	0%	0%	0%	0%	68.8%	24.5%
3	0%	0%	0.4%	0%	82%	34.5%
4	0%	0%	1%	0%	85.5%	40.3%
5	3.6%	0.1%	3.9%	1.2%	82.1%	34.9%
6	19.7%	0.2%	16.5%	3.1%	74%	22.2%
7	38.8%	4.7%	39.5%	4.8%	60.9%	12.4%

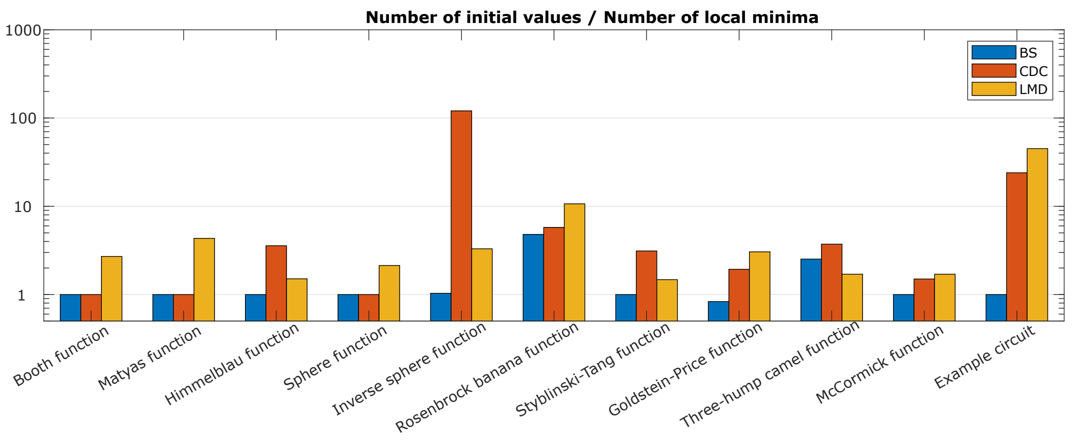

Fig. 3. Ratio between the number of initial values the algorithms found and the number of local minima of each function. Additionally, the analysis of the example circuit is shown for reference.

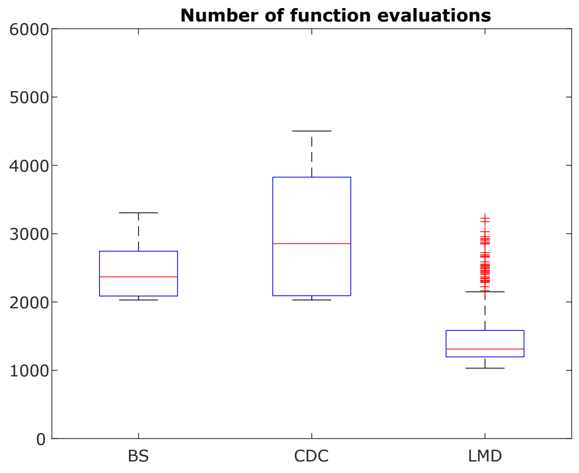

Fig. 4. Total number of function evaluations from initial value search to found optimum. The values contain the local optimum search too.

where $\mathbf{x}$ is an N-dimensional vector and x_i is the ith element of $\mathbf{x}$. This function has 2^N local minima at the corners of the search space.

Table I shows the ratio of missed local minima and the ratio of duplicate points per local minima. It can be seen that at most 4-dimensional problems, the BS and CDC algorithms did not miss any local minima in the search space. Over 5-dimensional functions, the algorithms started omitting regions of attraction. Out of three, the CDC algorithm missed the least number of local minima. In our experiments we have found that if the example function has at most 4 dimensions, the BS algorithm returns exactly the same number of initial values as the number of real local minima, while the other two algorithms always have multiple initial points in the same attraction regions. This problem is the most severe for the CDC algorithm: for example, even at the two-dimensional case, CDC returns more than three times as much duplicate points as real local minima. The LMD algorithm also produces a large number of duplicates, even below 5-dimensional problems too.

Based on our experiments, we suggest to use the BS algorithm for searching for initial values in numerical optimization based worst-case circuit analysis tasks.

VII. Conclusion and Further Research

Numerical optimization is a powerful tool in the worst-case analysis of complex circuits. The algorithms shown in this paper provide initial value sets suitable for global optimization tasks.

Further research include evaluating and comparing the performance of additional global optimization methods (e.g. GLOBAL [5] and memetic algorithms [4]).

VIII. Acknowledgement

Project no. 2019-1.3.1-KK-2019-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme.

References

[1] W. Smith, "Worst case circuit analysis-an overview (electronic parts/circuits tolerance analysis)," in *Proceedings of 1996 Annual Reliability and Maintainability Symposium*, 1996, pp. 326–334.

[2] A. Singhee and R. A. Rutenbar, "Why quasi-monte carlo is better than monte carlo or latin hypercube sampling for statistical circuit analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 11, pp. 1763–1776, 2010.

[3] A. Lokanathan and J. B. Brockman, "Efficient worst case analysis of integrated circuits," in *Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*. IEEE, 1995, pp. 237–240.

[4] F. Schoen and L. Tigli, "Efficient large scale global optimization through clustering-based population methods," *Computers & Operations Research*, vol. 127, p. 105165, 2021.

[5] B. Bánhelyi, T. Csendes, B. Lévai, L. Pál, and D. Zombori, *The GLOBAL Optimization Algorithm: Newly Updated with Java Implementation and Parallelization*. Springer, 2018.

[6] J. W. Chinneck, "Analyzing mathematical programs using mprobe," *Annals of Operations Research*, vol. 104, no. 1, pp. 33–48, 2001.

[7] ——, "Discovering the characteristics of mathematical programs via sampling," *Optimization Methods and Software*, vol. 17, no. 2, pp. 319–352, 2002.

[8] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," *arXiv preprint arXiv:1308.4008*, 2013.