DC Operating Point Analysis using Evolutionary Computing

D. A. Crutchley and M. Zwolinski

Abstract – This paper discusses and evaluates a new approach to operating point analysis based on evolutionary computing (EC). EC can find multiple solutions to a problem by using a parallel search through a population. At the operating point(s) of a circuit the overall error has a minimum value. Therefore, we use an Evolutionary Algorithm (EA) to search the solution space to find these minima. Various evolutionary algorithms are described. Several such algorithms have been implemented in a full circuit analysis tool. The performance and accuracy of the algorithms are compared to Newton-Raphson (NR). Evolutionary algorithms are shown to be robust and to have an accuracy comparable to that of NR. The development of a hybrid algorithm is also discussed.

I. Introduction

The first task in simulating the behaviour of a circuit is to find the quiescent or DC operating point. The DC operating point is used as the starting point for transient analysis (circuit response in the time domain) [1] and for AC analysis. Traditionally, the operating point is found using the *Newton-Raphson* Method (NR). This method has three potential disadvantages. The first is that at the start of each iteration we must re-compute the Jacobian matrix. The second disadvantage is that the solution can diverge or fail to converge by oscillating between several potential solutions. Finally, for circuits with more than one possible solution, the initial guess can influence the final solution and hence finding multiple global solutions is generally difficult.

In this paper we discuss various aspects of *Evolutionary Computing* (EC), in particular, *Evolution Strategies* (ESs) and *Differential Evolution* (DE), and how these techniques can be applied to DC circuit analysis [2]. As will be seen, EC has certain advantages over NR. The main benefits are improved convergence and the ability to find multiple solutions. These can be attributed to the parallel nature of EC algorithms; i.e. a search through a population of solutions rather than a sequential search for an individual solution, as in NR.

At the end of this paper we will discuss of the results obtained from a SPICE-like evolutionary circuit simulator that implements versions of the basic Evolutionary Algorithms.

First, we will define the notation used throughout this work. In the Newton-Raphson algorithm we represent the candidate solutions, e.g. the vector of node voltages and/or branch currents, as a real-valued *candidate vector* $\mathbf{x}^k = \left(x_1^k, ..., x_n^k\right)^{\mathrm{T}}$, at the k^{th} iteration. In general, the aim is to find a set of *n* variables $\mathbf{x}^* = \left(x_1^*, ..., x_n^*\right)^{\mathrm{T}}$, such that, for some *objective function*, *f*, we have $f(\mathbf{x}^*) = 0$. In the case of evolutionary algorithms, the candidate vector has a subscript, *i*, that denotes the i^{th} member of the population and so the candidate vector is written as $\mathbf{x}_i^k = \left(x_{i,1}^k, ..., x_{i,n}^k\right)^{\mathrm{T}}$. In the case of circuit analysis, the objective function is a vector, $\mathbf{f}(\mathbf{x}) \in \Re^n$, representing the characteristic equations of the nonlinear circuit components. In EAs the objective function, *f*, represents the *fitness* of a particular candidate vector.

D. A. Crutchley was and M. Zwolinski is with the School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom, E-mail: mz@ecs.soton.ac.uk

II. Conventional Circuit Analysis

A. The Newton-Raphson Method

The most common technique for nonlinear circuit analysis is the Newton-Raphson method coupled with LU factorization. We aim to solve the set of nonlinear equations $\mathbf{f}(\mathbf{x}^k) = \mathbf{0}$, which is the vector formed from the devices' characteristic equations. By formulating the linearized matrix vector equation (1), using element stamps, we can find the root(s) of $\mathbf{f}(\mathbf{x}^k) = \mathbf{0}$

$$\mathbf{G}^k \mathbf{x}^{k+1} = \mathbf{I}^k . \tag{1}$$

Eq. 1 finds $\mathbf{x}^{k+1}$; the node voltage vector at the $(k+1)^{st}$ iteration. The matrix $\mathbf{G}^k$ is the Jacobian and the RHS vector, $\mathbf{I}^k$, is the vector of excitations. Both are initially set to zero and are updated using element stamps [3], which allow us to update the matrix and RHS automatically with contributions from each device's characteristic equation. By repeatedly solving equation (1) and using the solution to formulate the matrix and excitation vector for the next iteration, the solution vector should converge to an accurate representation of the state of the circuit [1], [2], [4].

This process is computationally intensive. The matrix equation needs to be set up and solved once per Newton

iteration. Building the Jacobian matrix requires the evaluation of the partial derivatives of the device equations. Typically, evaluating the device equations and derivatives, along with the associated matrix operations, requires about 70% or more of the total CPU time [5], [6].

B. Convergence

Newton-Raphson's sensitivity to the initial values in the solution vector used to start the analysis is especially noticeable when we are dealing with nonlinear circuit equations that have multiple solutions.

The main problem with the selection of the initial values is that one can never be sure of the radius of convergence for a particular problem and so picking an initial solution that is outside this radius can lead to divergence or, if there are multiple solutions, could lead to finding a solution other than that being sought. There are several techniques that can be used to help convergence, but these techniques do not all work for every problem.

III. EVOLUTIONARY COMPUTING

There are many different types of evolutionary algorithms, for example: evolution strategies; genetic algorithms (GAs); differential evolution; evolutionary programming and genetic programming, [7], [8], [9], [10], [11], [12]. They all use similar techniques (operators to mimic sexual reproduction and mutation; selection processes), but they differ in the way that they represent a candidate vector. For our problem of DC analysis, a candidate vector would have a genotype that is a vector of binary values for the integer and fractional parts of the node voltage values and a phenotype that is a vector of real valued data for the node voltage values. Genetic algorithms, [12], manipulate the binary data using bit mutation and binary string crossover schemes and then map the resulting data into phenotypic form. On the other hand, the evolutionary strategies and differential evolution based methods operate at the phenotypic level.

We aim to manipulate a population and thus perform a parallel search through the solution space instead of performing a sequential search for solutions, as with NR. For each member of the *population*, $P^k = \{\mathbf{x}_1^k, \ \ldots \ , \mathbf{x}_N^k\}$, at the k^{th} generation, we aim to minimize *n* objectives (nodal equations) for each individual. *N* is the size of the population. Here $\mathbf{x}_i^k$ represents the i^{th} candidate vector of node voltages and/or branch currents, for $i=1,2,...,N$, at the k^{th} generation. Usually, we set an upper limit on the total number of generations for which the algorithm can run; we call this K_{MAX}. We then form the *objective vector*, denoted by $\mathbf{y}_i^k = \left(y_{i,1}^k,...,y_{i,n}^k\right)^{\mathrm{T}}$, which will contain the objectives that we wish to minimize. In the case of nodal analysis, $\mathbf{x}_i^k$ contains only node voltages so we can use these in the evaluation of the device equations. The resulting device currents can then be used to form the KCL equation for each node in the circuit. Hence, we define y_j as the absolute value of the net current flowing at node *j*, which is the node's KCL equation. Thus the objective vectors are in one-to-one correspondence with the candidate vectors.

During the optimization process we minimize the y_j values, hence as a candidate vector reaches optimality the y_j values in the corresponding $\mathbf{y}_i$ objective vector will tend towards zero. In other words, the net current flowing at each node should be zero for a perfect solution. We minimize the y_j values by minimizing the *fitness* of the solutions. Hence, by keeping candidate vectors with better fitness values we will steadily be reducing the values in the objective vectors over successive generations.

A. Evolution Strategies

Evolution strategies are probabilistic, heuristic, direct search optimization techniques, invented independently in 1965 by Rechenberg [9] and Schwefel [10]. There is generally no crossover or inversion in ES (or not in the same sense as in GAs [12]). Sometimes, however, it can be beneficial to have a crossover-like operator. When this is the case we use *discrete recombination*, in which we recombine two parents by discretely swapping randomly selected vector components.

We use a population divided into a parent and an offspring pool. At the end of a generation the best individuals from the union of the parent and offspring pools are taken as the parents for the next generation.

In general, ESs use only a mutation operator but the self-adaptive scheme (ESA) also uses a recombination operator.

B. Tournament Selection Evolutionary Algorithm

The EAs described use *truncation selection* [11]. Tournament selection is an alternative selection mechanism. At the start of each generation, each member of the population is compared pair-wise with randomly selected, distinct members of the population and gains a tally point if it is fitter than that member. When a parent is required, we randomly pick a parent from the population and randomly accept or reject that choice using a biased coin toss compared with the tally total. Here, we apply Tournament Selection to the ES parent pool described in the previous section.

C. Differential Evolution

Storn and Price described an evolutionary algorithm that is self-adaptive, simple and yet very powerful called

Differential Evolution (DE) [7]. Several DE schemes have been proposed by Storn [13]; two schemes will be discussed here: DE1 and DE2 [7]. In DE1, for each candidate vector $\mathbf{x}_i^k$, in the population, we generate an intermediate vector $\mathbf{v}_i^k$ as follows:

$$\mathbf{v}_i^k = \mathbf{x}_{r_1}^k + \tau \cdot (\mathbf{x}_{r_2}^k - \mathbf{x}_{r_3}^k) . \quad (2)$$

In Eq. 2, τ is a positive real-valued user-set scale factor and r_1, r_2 and r_3 are randomly selected, mutually distinct integers in the range $[1, N]$. The intermediate vector $\mathbf{v}_i^k$ is used with $\mathbf{x}_i^k$ in a crossover procedure to generate a new offspring $\tilde{\mathbf{x}}_i^{k+1}$. If $\tilde{\mathbf{x}}_i^{k+1}$ is fitter than $\mathbf{x}_i^k$ then $\mathbf{x}_i^k = \tilde{\mathbf{x}}_i^{k+1}$ and we discard $\mathbf{x}_i^k$, else we keep $\mathbf{x}_i^k$. This means that we no longer require an offspring pool.

DE2 is identical to DE1 except for the generation of the intermediate vector $\mathbf{v}_i^k$. This time an additional difference vector is used as shown in Eq. 3:

$$\mathbf{v}_i^k = \mathbf{x}_i^k + \tau' \cdot (\mathbf{x}_{best}^k - \mathbf{x}_i^k) + \tau \cdot (\mathbf{x}_{r_1}^k - \mathbf{x}_{r_2}^k) . \quad (3)$$

Note that this time we only need two random integers r_1 and r_2, and τ and τ' are positive user-set scale factors.

D. A Hybrid Evolutionary Algorithm

The main idea behind the *Differential Evolution initialized Newton-Raphson method* (DENR) [14] is that after a certain number of generations an EA ceases to make dramatic changes to the population and instead refines the existing population until convergence. This suggests that at the point where this change in gradient occurs, one should switch to some other faster technique to finish the optimization. Hence, we use the differential evolution algorithm DE1 coupled with Newton's method. The diagram in Fig. 1 illustrates this technique.

At this point in the simulation, a filtering algorithm is used on the population to remove potential duplicate solutions. Now each of the remaining candidate solutions is used, in turn, as a starting point for NR.

The benefit of this algorithm is that it exploits the speed of NR but tries to avoid the initial condition problem of NR by using DE1 to provide multiple and good starting points for NR. As will be seen later this algorithm is both fast and accurate.

IV. DC Analysis using Evolutionary Computing

A. Benchmark Circuits

Several benchmark circuits were used to evaluate the performance of all of the EAs. SPICE level 3 MOS models were used unless otherwise noted. Three benchmark circuits are discussed here:

- An Inverting Schmitt Trigger. The Schmitt Trigger is made up of five p-type and five n-type MOSFETs. In DC analysis, there are three possible solutions, including a metastable state.
- A multi-state circuit with nine operating points, consisting of 4 bipolar transistors and 14 resistors. This circuit is taken from Chua and Ushida's work [15].
- A positive edge-triggered D-latch with inputs D=1 and C=0. This circuit has multiple operating points. It consists of 7 CMOS inverters and 4 transmission gates and has a total of 22 transistors.

B. Experimental Results and Discussion

The performance of each algorithm with each of the circuits is shown in Tables 1 - 3. The algorithm is stated in the first column of each table. In the second column, the number of solutions found by each algorithm automatically is stated as an integer. If multiple solutions could be found by changing settings, this is stated as an expression (e.g. 1+1 means 2 solutions were found by restarting the algorithm). The third column shows the number of generations (or for NR, the number of iterations). In the case of DENR, the second figure gives the total number of NR iterations. The fourth column shows the accuracy relative to NR. Finally the CPU time in milliseconds is given. Again, if multiple runs were needed to find multiple solutions, this is stated as a sum.

In the case of the multi-stable bipolar circuit some of the NR results are bracketed from the others. This is because the starting points required for these solutions proved to be harder to find than for the first 5 solutions.

DE1, DE2 and ES are often the best evolutionary algorithms for finding multiple solutions automatically. ESA is the least good at finding multiple solutions, even when restarted and parameter settings are altered. By far the best evolutionary algorithm for multiple solutions is the hybrid DENR.

NR is always the fastest, in terms of CPU time (but note that manual intervention is needed to find multiple solutions). For all the circuits, DENR is always the fastest of the evolutionary algorithms in terms of CPU time. ESA is consistently the slowest (apart from for the Schmitt trigger circuit, where it only found one solution). The best-performing evolutionary algorithm's (DENR) CPU time is, however, between about 5 and 20 times that of NR.

The accuracy of the evolutionary algorithms is very similar. DENR followed by TSEA are the most accurate in most cases and the DE algorithm is usually accurate too.

The CPU time does not necessarily represent the total effort required to find a solution. This is particularly true when multiple solutions exist and are sought. It can therefore be argued that the best algorithms in terms of accuracy, speed and the ability to find multiple solutions and to analyze problem circuits, such as the Schmitt trigger and the other multi-stable circuits, are DENR or DE1.

V. CONCLUSIONS

The use of evolutionary algorithms for nonlinear operating point analysis of MOS circuits has been demonstrated. DE and the other EAs are globally convergent, whereas NR is only locally convergent. NR requires manual intervention to find all the solutions to a circuit; it has been shown that DE and the other EAs can find multiple solutions in a single pass.

All the EAs here are, in general, slower than NR. This can be attributed to two factors. Firstly, a significant amount of sorting of populations has to be done. This accounts for the majority of the CPU time taken. Secondly the device equations have to be evaluated many times in each generation (once for each member of the population). The hybrid algorithm is very competitive, however, when compared to NR for the circuits tested here.

REFERENCES

[1] Litovski, V. and Zwolinski, M., *VLSI: Circuit Simulation and Optimization*, Chapman and Hall, London, 1997.

[2] Zwolinski, M., Crutchley, D.A. and Yang, Z.R., "Evolutionary computing for operating point analysis of nonlinear circuits", *Proceedings of ICSES 2000*, Poland, Oct 17th-20th 2000.

[3] Ho, C.W., Ruehli, A.E. and Brennan, P.A., "The modified nodal approach to network analysis", *IEEE Trans. on Circuits and Systems*, vol. CAS-22, no. 6, June 1975, pp. 504-509.

[4] Calahan, D.A., *Computer Aided Network Design*, Revised Edition, McGraw-Hill, New York, 1972.

[5] Cox, P.F., Burch, R.G., Hocevar, D.E., Yang, P. and Epler, B.D., "Direct circuit simulation algorithms for parallel processing", *IEEE Trans. on Computer-Aided Design*, Vol. CAD-10, No.6, June 1991, pp.714-725.

[6] Johnson, T.A. and Zukowski, D.J. (1991), "Waveform-relaxation-based circuit simulation on the Victor V256 parallel processor", *IBM J. Res. Develop.* Vol.35, No.5/6, Sept/Nov.

[7] Storn, R. and Price, K., "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces", *Technical Report* TR-95-012, ICSI, Berkeley, 1995.

[8] Fogel, L.J., "Autonomous automata", *Industrial Research*, Vol. 4, 1962, pp 14-19.

[9] Rechenberg, I., "Cybernetic solution path of an experimental problem", *Royal Aircraft Establishment, Library Translation* No. 1122, August 1965.

[10] Schwefel, H.-P., "Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik", Diploma Thesis, Technical University of Berlin, 1965.

[11] Fogel, D.B., *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*, 2nd Ed., IEEE Press, NY., 2000.

[12] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[13] Storn, R., "On the usage of differential evolution for function optimization", *Technical Report*, ICSI, Berkeley, 1996.

[14] Crutchley, D.A. and Zwolinski, M., "Using evolutionary and hybrid algorithms for DC operating point analysis of nonlinear circuits", *Proc. 2002 Congress on Evolutionary Computation (CEC'02)*, Hawaii, USA, May 12th-17th 2002.

[15] Chua, L.O. and Ushida, A., "A switching parameter algorithm for finding multiple solutions of nonlinear resistive circuits", *International Journal Circuit Theory and Applications*, Vol. 4, 1976, pp. 215-239

[16] CircuitSim90 Benchmark Circuits, North Carolina State University, CAD Benchmarking Laboratory, http://www.cbl.ncsu.edu/CBL_Docs/csim90.html

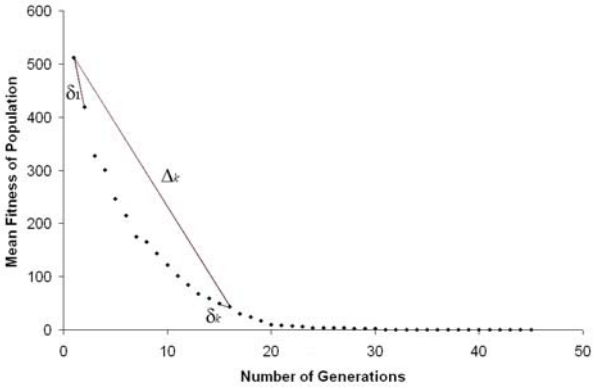

Fig. 1 Criteria for switching from EC to NR.

TABLE 1
RESULTS FOR SCHMITT TRIGGER

Alg.	No. Sols	No. Gens/Its	Mean Err	CPU (ms)
NR	1+1	144+13	~	11+2
DE1	2	271	2.06×10^{-1}	1210
DE2	2	481	1.88×10^{-1}	1980
ES	2	107	6.64×10^{-1}	830
ESA	1	29	2.01×10^{-1}	490
TSEA	1+1	28+31	1.04×10^{-1}	270+330
DENR	2	29+407	1.18×10^{-1}	50

TABLE 2
RESULTS FOR MULTI-STATE BJT CIRCUIT

Alg.	No. Sols	No. Gens/Its	Mean Err	CPU (ms)
NR	1+1+1+ 1+1(+1+ 1+1)	13+9+11+ 12+12(+13+ 18+15)	~	5+4+5+5+5 (+5+6+5)
DE1	5	2214	3.48×10^{-1}	6709
DE2	3	4669	2.97×10^{-1}	13980
ES	2	1899	4.85×10^{-1}	5958
ESA	0	~	~	~
TSEA	1+1	336+295	1.52×10^{-1}	1025+980
DENR	4+3+1	9+1806, 9+2106, 17+474	2.50×10^{-5}	105+150+ 160

TABLE 3
RESULTS FOR POSITIVE EDGE TRIGGERED D-LATCH

Alg.	No. Sols	No. Gens/Its	Mean Err	CPU (ms)
NR	1+1+1	8+186+186	~	5+20+19
DE1	3	241	1.25×10^{-2}	1629
DE2	3	471	1.59×10^{-2}	3047
ES	2	85	2.10×10^{-3}	1109
ESA	1	22	1.15×10^{-1}	760
TSEA	1+1+1	10+9+12	8.17×10^{-3}	270+240+ 340
DENR	3	46+2493	7.43×10^{-3}	690