INTEGRATION the VLSI journal 59 (2017) 218–232

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Co-AGSA: An efficient self-adaptive approach for constrained optimization of analog IC based on the shrinking circles technique

Maryam Dehbashian, Mohammad Maymandi-Nejad*

Department of Electrical Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

ARTICLE INFO

Keywords:
Advanced gravitational search algorithm (AGSA)
Shrinking circles technique
Constraint handling
Co-evolutionary
Circuit sizing tool

ABSTRACT

This paper aims to take a step forward to enhance the performance of the optimization kernel of electronic design automation (EDA) tools by coping with the existing challenges in the analog circuit sizing problems. For this purpose, a novel co-evolutionary-based optimization approach, called Co-AGSA, is proposed. In the Co-AGSA, a self-adaptive penalty technique based on the concept of the co-evolution model is incorporated into a powerful optimization algorithm, named advanced gravitational search algorithm (AGSA), to efficiently solve more realistic constrained optimization problems. The performance of the Co-AGSA approach is first evaluated by solving three constrained engineering design problems. Then, the optimization capability of the Co-AGSA-based IC sizing tool is validated using three different case studies, i.e., a two-stage op-amp, a folded-cascode op-amp and a two-stage telescopic cascode amplifier, to show the applicability of the proposed approach. The results demonstrate that the Co-AGSA gives better performance compared to other approaches in terms of efficiency, accuracy and robustness.

1. Introduction

The design of analog circuits in comparison with digital circuits is more difficult, more time consuming and requires highly skilled designers due to more complexity of analog circuits. Thus, to overcome all these difficulties and speed up the design process, Computer Aided Design (CAD) and Electronic Design Automation (EDA) tools were developed. A complete overview of the available circuit sizing tools for the design automation of analog circuits and state-of-the-art sizing optimization techniques has been presented in [1].

In recent years, with the increasing progress in the development of CAD software (e.g., CADENCE® [2], Synopsys® [3] and Mentor Graphics® [4]) and EDA tools (e.g., MunEDA® [5], SOLIDO® [6], Silvaco® [7] and AIDA [8]), the analog IC designers are gradually using them in some steps of the manual design flow. Since the manual design of analog circuits is a tedious and cumbersome task, the use of foregoing tools definitely increases the productivity and the quality of final designs and reduces the cost and the time-to-market. However, designers only employ them as an intelligent assistant and not an expert manager in the design process.

To the best of authors' knowledge, one of the main reasons behind the limited use of EDA tools by the IC designers is the high need for the constant supervision of the designer in all phases of the design process that makes the design flow time consuming. Moreover, the designer may not be satisfied with the solution suggested by one of these EDA tools because each tool provides a solution considering the trade-off between three key factors of accuracy, robustness, and run time. Based on the above reasons and due to the fact that there is no benchmark circuit to compare the performance of EDA tools, no one can claim that the solution provided by one of these tools is the best possible solution.

Improving the performance of the EDA tools for achieving a mature solution in analog IC sizing has become an active research field during the recent years, for instance, in [9–13]. However, there are many challenges in this field. In this paper, we will address two of these challenges that we are currently facing for analog circuit sizing [14]: 1) the need for a more efficient constraint-handling technique and 2) the need for a more powerful optimization kernel. In the following, these two challenges are explained in detail.

Generally, the metaheuristic optimization algorithms (e.g., evolutionary algorithms or swarm intelligent algorithms) are used in the optimization kernel of circuit sizing tools. It should be noted that the optimization kernel is actually a computational engine by which design variables are updated in an iterative process until they achieve an optimum equilibrium point [10]. Despite the fact that the analog circuit sizing problems are highly constrained, the metaheuristic algorithms are not inherently able to handle the constraints. Hence, to deal with constrained problems, there is no alternative to equip optimization algorithms with constraint handling techniques.

* Corresponding author.
E-mail address: maymandi@um.ac.ir (M. Maymandi-Nejad).

http://dx.doi.org/10.1016/j.vlsi.2017.06.003
Received 29 November 2016; Received in revised form 5 May 2017; Accepted 4 June 2017
Available online 17 July 2017
0167-9260/ © 2017 Elsevier B.V. All rights reserved.

The *static penalty function* is the most widely used technique for dealing with constrained optimization problems; this is due to its simplicity and easy implementation that have made it as a relatively reliable method [15]. For minimization problems, the penalty function penalizes the infeasible solutions by adding a certain value to the objective function as an amount proportional to the constraint violation. Thereby, this technique transforms the original constrained optimization problem into an unconstrained one. The determination of an appropriate penalty factor is quite problem dependent and thus it is often a very difficult and time-consuming task. If a high value is selected for the penalty factor, a premature convergence to a local optimum may occur. In contrast, if a too small penalty factor is selected, the search process may be led to outside the feasible region resulting in infeasible solutions [16].

According to [16], most of the current circuit sizing methods employ static penalty functions to handle the constraints. However, since the static penalty function technique suffers from the sensitivity of the penalty factors, it is not effective enough. Unfortunately, many state-of-the-art constrained optimization methods have not been introduced into the EDA tools yet, and hence, to address this challenge, advanced constrained optimization methods need to be applied for circuit sizing tools. On the other hand, according to the significant advances in the IC manufacturing industry, the circuit specifications have become more severe while circuit sizes have been highly scaled-down. Therefore, another challenge to size high-performance analog circuits with tough specifications is the necessity of having a sufficiently powerful optimization kernel for EDA tools to handle more stringent specifications in addition to enhancing the optimization ability [14]. By far, different optimization kernels are provided for EDA tools, among them, we can mention the kernels based on the following optimization algorithms: a modified GA in [17], PSO in [18], ACO in [19], SA in [20], GSA in [21], NSGA-II in [11], and NSGA-II, MOPSO and MOSA in [10].

To cope with the first challenge, the constraint-handling techniques based on the co-evolutionary model can be good alternatives. Since the determination of proper penalty factors is a tough work, Coello [22], as a pioneering work, introduced a self-adaptive penalty approach based on the concept of co-evolution and incorporated it into a genetic algorithm (GA) to solve constrained optimization problems. In this approach, the optimal values of penalty factors are automatically determined during the optimization process in which two populations are generated in parallel in such a way that one population seeks to find the feasible solutions while the second one adapts the associated penalty factors. Likewise, a co-evolutionary particle swarm optimization (CPSO) approach with some modifications on Coello's co-evolution model was presented by [23,24]. In these works, two interactive swarms were evolved with PSO to simultaneously find the feasible solutions and suitable penalty factors. In an analogous way with CPSO, the differential evolution approach based on co-evolution model named CDE was also introduced in [25]. Notably, however, all co-evolutionary-based approaches do not need to define any extra parameters in optimal determination of the penalty factors except the initialization of their associated optimization algorithms. This feature demonstrates the merits of the co-evolutionary model as a self-tuning approach with the least user intervention in the constraint-handling techniques.

To cope with the second challenge (i.e., the need for a more powerful optimization kernel), a novel optimization algorithm named advanced gravitational search algorithm (AGSA) [26] has been recommended as a powerful kernel to enhance the performance of the automated analog IC sizing tools. The main reason for this is that according to the results provided by the pioneer reference [26], the obvious capabilities of AGSA in the optimization of a set of standard benchmark functions demonstrate that AGSA is a good choice to be used as an optimization kernel in the circuit sizing tools. In fact, AGSA is an upgraded version of gravitational search algorithm (GSA) based on an innovative technique named *shrinking circles*. This technique has been devised to balance the exploration and exploitation capabilities when the optimization algorithm is converging to a possible optimum point.

According to the solutions mentioned above, this paper proposes the incorporation of the co-evolution model into the AGSA in order to provide a powerful approach, which we call it Co-AGSA, in solving constrained optimization problems. Moreover, the Co-AGSA approach is used as the optimization kernel of an EDA tool to efficiently size high-performance analog circuits.

The rest of this paper is organized as follows. Section 2 provides a brief review of GSA and AGSA. In this section, the shrinking circles technique and the way it can be incorporated into AGSA are also explained. Section 3 describes the procedure of the proposed Co-AGSA approach and presents a heuristic method for the reduction of computational time. Section 4 evaluates the performance of Co-AGSA in comparison with other methods by solving three realistic engineering problems. Section 5 introduces a novel circuit sizing tool based on Co-AGSA and validates its performance using three different case studies. Finally, Section 6 concludes the paper with some relevant remarks.

2. Brief review of GSA and AGSA

2.1. Gravitational search algorithm

The GSA is one of the widely used algorithms of the swarm intelligence family that is inspired by the *Newton's laws of gravity and motion* [27]. In the search space of GSA, there is a collection of N agents (candidate solutions) in which each agent has a certain mass, so that the performance of each agent is evaluated by the value of its own fitness function. Moreover, each individual agent attracts every other agents using the mutual gravitational force that is directly proportional to the product of the corresponding mass values and inversely proportional to the square of the distance between them. According to the *Newton's law of universal gravitation*, the force of gravity moves all the agents towards the agent with heavier mass. Thus, by the lapse of time, the heaviest mass attracts the other agents gradually until the convergence criterion is met. In this case, the heaviest mass is considered as the optimum solution. For the sake of clarity, the optimization steps of GSA are outlined below.

- ***Step 1: Initialization***

  At the beginning of the algorithm, it is assumed that there are *N* masses in the search space of the problem. The positions of the N masses are initialized randomly as below:

$$X_i = (x_i^1, \ldots, x_i^d, \ldots, x_i^n) for\, i = 1, 2, \ldots, N. \tag{1}$$

  where, x_i^d represents the position of *i*th mass in the *d*th dimension.

- ***Step 2: Fitness evaluation***

  According to the objective function, the fitness values of all masses at iteration *t* are calculated. Then, for a minimization problem, the best and worst fitness values are defined as below:

$$best(t) = \min_{j \in \{1,\ldots,N\}} fit_j(t) \tag{2}$$

$$worst(t) = \max_{j \in \{1,\ldots,N\}} fit_j(t) \tag{3}$$

  where, $fit_j(t)$ represents the fitness value of the mass *i* at iteration *t*.

- ***Step 3: Normalization***

  The normalized mass is calculated at iteration *t* by the following equations:

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \tag{4}$$