Chapter 2
Fundamentals of Optimization Techniques in Analog IC Sizing

Chapters 2, 3, 4 concentrate on high-performance analog integrated circuit sizing under nominal conditions. This is the foundation of advanced topics in the following chapters of this book, i.e., variation-aware analog IC sizing and electromagnetic (EM)-simulation based mm-wave integrated circuit and antenna synthesis. This chapter defines the problem, introduces some widely used evolutionary algorithms and basic constraint handling approaches.

This chapter is organized as follows. Section 2.1 introduces the problem. Section 2.2 reviews existing approaches to analog circuit sizing. Section 2.3 introduces the general implementation of an EA and provides a detailed introduction to the differential evolution (DE) algorithm, which is used as the typical search mechanism throughout this book. Section 2.4 introduces two basic but widely used constraint handling techniques. Section 2.5 introduces two widely used multi-objective evolutionary algorithms (MOEAs). Two examples are shown in Sect. 2.6. Section 2.7 provides the summary of this chapter.

2.1 Analog IC Sizing: Introduction and Problem Definition

Nowadays, VLSI technology progresses towards the integration of mixed analog-digital circuits as a complete system-on-a-chip. Although the analog part is a small fraction of the entire circuit, it is much more difficult to design due to the complex and knowledge-intensive nature of analog circuits. Without an automated synthesis or sizing methodology, analog circuit design suffers from long design times, high design complexity, high cost and requires highly skilled designers. Consequently, automated synthesis methodologies for analog circuits have received much attention. The analog circuit design procedure consists of topological-level design and parameter-level design (also called circuit sizing) [1, 2]. This book concentrates on the latter, aiming at parameter selection and optimization to improve the performances for a given circuit topology. We assume that the designer provides the circuit topology.

There are two main purposes of an analog IC sizing system: first, to replace exploration of tedious and ad-hoc manual trade-offs by automatic design of parameters; secondly, to solve problems that are hard to design by hand. Accuracy, ease of use, generality, robustness, and acceptable run-time are necessary for a circuit synthesis solution to gain acceptance [3]. Other than those requirements, the ability to deal with complex problems, closely meeting the designer's requirements even for highly constrained problems, and the ability to achieve highly optimized results are significant objectives of a high-performance analog IC sizing system. Many parameter-level design strategies, methods, and tools have been published in recent years [1–22], and some have even reached commercialization. We will review them in Sect. 2.2.

Most analog circuit sizing problems can naturally be expressed as a single- or multi-objective constrained optimization problem. We first consider single-objective cases. The problem can be defined as the minimization[1] of an objective, e.g., power consumption, usually subject to some constraints, e.g., DC gain larger than a certain value. Mathematically, this can be formulated as follows:

$$\begin{aligned}&\text{minimize}_x\, f(x)\\&\text{s.t. } g(x) \geq 0\\&h(x) = 0\\&x \in [X_L, X_H]\end{aligned} \tag{2.1}$$

In this equation, the objective function $f(x)$ is the performance function to be minimized. $h(x)$ are the equality constraints. In analog circuit design, the equality constraints mainly refer to Kirchhoff's current law (KCL) and Kirchhoff's voltage law (KVL) equations. Vector x corresponds to the design variables, and X_L and X_H are their lower and upper bounds, respectively. The vector $g(x) \geq 0$ corresponds to user-defined inequality constraints. An example is provided as follows: given the example circuit topology shown in Fig. 2.1, the sizing problem can be defined as (2.2). The objective of the sizing system is to determine the sizing and biasing of all devices (transistors, capacitors, etc) such that the power consumption is the smallest possible while satisfying all the constraints mentioned in (2.2).

$$\begin{aligned}&\text{minimize power}\\&\text{s.t. DC gain} \geq 80\,\text{dB}\\&\text{GBW} \geq 2\,\text{MHz}\\&\text{Phase Margin} \geq 50^\circ\\&\text{Slew Rate} \geq 1.5\,\text{V}/\mu\text{s}\end{aligned} \tag{2.2}$$

For multi-objective analog circuit sizing, we simultaneously optimize more than one performance and get the approximate Pareto-optimal front. By multi-objective sizing, the trade-offs and sensitivity analysis between the different objectives can be

[1] The maximization of a design objective can easily be transformed into a minimization problem by just inverting its sign.

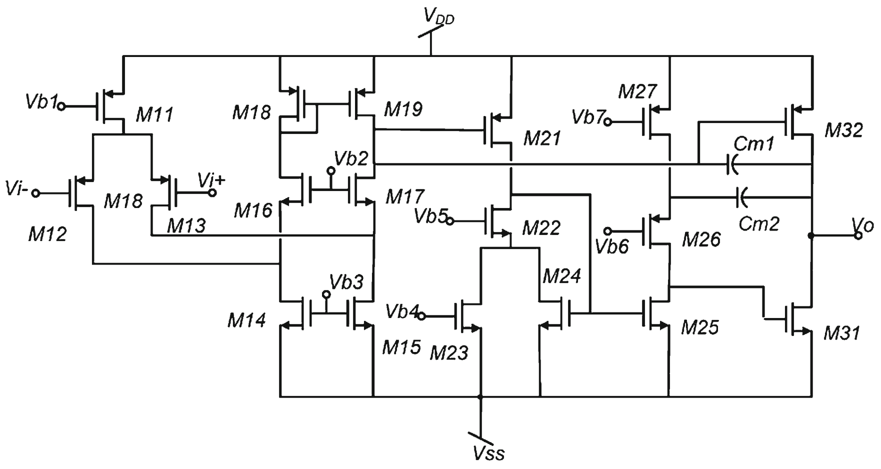


Fig. 2.1 A CMOS three-stage amplifier

explored. In multi-objective sizing, performances can either be set as objectives or constraints. An objective means that the designer is interested in its trade-offs with all other objectives in the whole performance space, which contains much information; while a constraint usually means that the performance needs to be larger than or smaller than a certain value, and no trade-off information is considered. Usually, only a part of the important performances are considered as objectives, and the others can be set as constraints. The reason is that the designer is often not interested in the trade-offs among all performances. Moreover, there are typically other design constraints, such as restricting the transistors to operate in the saturation region, which are essentially defined as functional constraints. An example is shown in (2.3).

$$\begin{aligned}
&\text{minimize power}\\
&\text{minimize area}\\
&\text{s.t. DC gain} \geq 80\,\text{dB}\\
&\text{GBW} \geq 2\,\text{MHz}\\
&\text{Phase Margin} \geq 50^\circ\\
&\text{Slew Rate} \geq 1.5\,\text{V}/\mu\text{s}
\end{aligned} \tag{2.3}$$

2.2 Review of Analog IC Sizing Approaches

Analog integrated circuit sizing can be carried out by the following two different approaches: knowledge-based and optimization-based [23]. The basic idea of knowledge-based synthesis is to formulate design equations in such a way that given the performance characteristics the design parameters can be directly calculated

[4–6]. In these tools, the quality of the solutions in terms of both accuracy and robustness are often not acceptable in complex circuits and modern technologies since the very concept of knowledge-based sizing forces the design equations to be simple [4]. Other drawbacks are the large preparatory time/effort required to develop design plans or equations, the difficulty in using them in a different technology, and the limitation to a limited set of circuit topologies [5].

In optimization-based synthesis, the problem is translated into a function minimization problem, which can be solved through numerical methods, such as (2.2). This kind of method is widely accepted [23]. Essentially, it is based on the introduction of a performance evaluator within an iterative optimization loop [23]. The system is called equation-based when the performance evaluator is based on equations capturing the behavior of a circuit topology [7–10]. However, creating the equations often consumes much more time than manually designing the circuit. In addition, the simplifications required in the closed-form analytical equations cause low accuracy and incompleteness. On the contrary, simulation-based methods do not rely on analytical equations but on SPICE-like simulations to evaluate the circuit performances in the optimization process (the values of $f(x)$ and $g(x)$ in (2.1) are derived by the electrical simulator), which results in superior accuracy, generality, and ease of use [12–14]. Therefore, the simulation-based optimization method is the main focus in this book. Through the link between an available circuit simulator (e.g., HSPICE [24]) and the environment of programming the optimization algorithm (e.g., MATLAB, C++ environment), the candidate parameter values are transmitted from the optimization system to the simulation engine, and the circuit performances obtained by the electrical simulator are returned to the optimization system. The penalty to pay is a relatively long computation time (compared to other methods), although, due to the fast speed of the circuit simulation software and computers nowadays, the computational time of a full sizing can be kept within very acceptable limits, normally from minutes to tens of minutes for a circuit up to tens of transistors.

Techniques for analog circuit optimization that have appeared in literature can broadly be classified into two main categories: deterministic optimization algorithms and stochastic search algorithms (evolutionary computation algorithms, simulated annealing, etc). The traditional deterministic optimization methods include Newton methods, Levenberg-Marquardt method, etc. These techniques are available in some commercial electrical simulators [24]. The drawbacks of deterministic optimization algorithms are mainly the following three aspects: (1) they require a good starting point; (2) an unsatisfactory local minimum may be reached in many cases; (3) they often require continuity and differentiability of the objective function. Some researchers have tried to address these difficulties, such as [15], where a method to determine the initial point is presented. Another approach is the application of geometric programming methods, which guarantee the convergence to a global minimum [10]. However, they require a special formulation of the design equations, which make them share many of the disadvantages of equation-based methods. Research efforts on stochastic search algorithms, especially evolutionary computation (EC) algorithms (genetic algorithms, differential evolution, genetic programming, etc) have begun to appear in literature in recent years [1, 16–22]. Due to the ability and efficiency to find a satisfactory solution in a reasonable CPU time, genetic algorithm (GA) has been employed as optimization routines for analog circuits in both industry and academia. For problems with practical design constraints, most of the reported approaches use the penalty function method to handle the constraints [7–11, 23].

2.3 Implementation of Evolutionary Algorithms

2.3.1 Overview of the Implementation of an EA

In Chap. 1, the basic concepts of evolutionary computation have been introduced using the example of the genetic algorithm. This subsection concentrates on the implementation of an EA. An EA program often follows the following procedure:

Step 1: Initialize the population of individuals by random generation.
Step 2: Evaluate the fitness of each individual in the initial population.
Step 3: Evolution process until the termination condition is met:

- **Step 3.1:** Select the parent individuals for reproduction.
- **Step 3.2:** Generate offsprings (child population) based on the parent individuals through crossover and mutation.
- **Step 3.3:** Evaluate the fitness of each individual in the child population.
- **Step 3.4:** Update the population by replacing less fitting individuals by individuals with good fitness.

Different EAs can differ from the representation of solutions to the search operators. For the representation of solutions, strings of numbers (e.g., binary), real numbers and computer program structure can be used, and the typical corresponding EAs are canonical genetic algorithm, evolution strategy and genetic programming, respectively. In terms of the search operators, there are a number of mutation, crossover and selection operators. Note that appropriate mutation, crossover and selection operators must be combined to achieve a good search performance.

The targeted problems in this book, the design automation of analog and RF IC, are numerical optimization problems. In the following, we will introduce the differential evolution (DE) algorithm [25, 26] as an example of the implementation of EA. DE is based on the vector differences and is especially suitable for numerical optimization problems. For many continuous global optimization problems, DE is often the first choice and is used as the search mechanism throughout this book. Besides DE, other promising CI approaches, real-coded genetic algorithm [27, 28], particle swarm optimization [29] and evolution strategy [30, 31] have good potential for the targeted problems, such as [32].

2.3.2 Differential Evolution

In general, the goals in this book are to optimize certain properties of a system by pertinently choosing the system parameters. Moreover, because of using simulation to evaluate the candidate's performances, the problems are often black-box optimization problems. They may have the properties of being nonlinear, non-convex and non-differentiable. As said above, among evolutionary algorithms, DE [25, 26] is recognized as a very effective evolutionary search engine for global optimization over continuous spaces.

Users generally demand that a practical optimization technique fulfills the following requirements [25] (minimization is considered):

(1) Ability to handle non-differentiable (gradients are not available or difficult to be calculated), nonlinear and multimodal cost functions (more than one or even numerous local optima exist).
(2) Parallelizability to cope with computation-intensive cost functions.
(3) Ease of use, i.e., few control variables to steer the minimization. These variables should also be robust and easy to choose.
(4) Good convergence properties, i.e., consistent convergence to the global minimum in consecutive independent trials.

As explained in the following, DE was designed to fulfill all of the above requirements. For the first two requirements, they are common to all EAs, so does DE. In order to satisfy requirement (3), DE borrows the idea from the Nelder and Mead algorithm [33] of employing information from within the vector population to alter the search space. DE's self-organizing scheme takes the difference vector of two randomly chosen population vectors to perturb an existing vector. Extensive testing under various conditions show the high performance of DE for complex benchmark problems [25, 26]. For requirement (4), although theoretical description of the convergence properties are available for many approaches, only extensive testing under various conditions can show whether an optimization method can fulfill its promises. DE scores very well in this regard for complex benchmark problems [25, 26].

In the following, the DE operations are described in detail.

Differential Evolution (DE) is a parallel direct search method which utilizes NP d-dimensional parameter vectors:

$$x_i(t) = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}], \quad i = 1, 2, \ldots, NP \tag{2.4}$$

as a population for each iteration t. NP is the size of the population and does not change during the minimization process. The initial vector population is chosen randomly and should cover the entire parameter space. As a rule, we will assume a uniform probability distribution for all random decisions unless otherwise stated. In case a preliminary solution is available, the initial population might be generated by adding normally distributed random deviations to the nominal solution. DE generates new parameter vectors by adding the weighted difference between two population vectors to a third vector. Let this operation be called mutation. The mutated vector's parameters are then mixed with the parameters of another predetermined vector (the corresponding vector generated in the last iteration), the target vector, to yield the so-called trial vector. Parameter mixing is often referred to as "crossover" in the EC community and will be explained later in more detail. If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the following generation. This last operation is called selection. Each population vector has to serve once as the target vector so that NP competitions take place in one generation (iteration).

More specifically DE's basic strategy can be described as follows:

A. Mutation

For each target vector $x_i(t)$; $i = 1, 2, \ldots, NP$, a mutant vector is generated according to

$$V_i(t+1) = x_{r1}(t) + F(x_{r2}(t) - x_{r3}(t)) \tag{2.5}$$

where random indexes $r1, r2, r3 \in 1, 2, \ldots, NP$, are integer, mutually different numbers and $F > 0$. The randomly chosen integers $r1$, $r2$ and $r3$ are also chosen to be different from the running index i, so that NP must be greater or equal to four to allow for this condition. F is a real and constant number within (0, 2] which controls the amplification of the differential variation $x_{r2}(t) - x_{r3}(t)$. The selection of F is detailed in [26]. For single-objective problems, using F between 0.8–1 is a common choice. For multi-objective optimization using DE as the search engine, this depends on the specific search mechanism, which will be illustrated in the following chapters. Figure 2.2 shows a two-dimensional example that illustrates the different vectors which play a part in the iteration of $V_i(t+1)$.

B. Crossover

In order to increase the diversity of the perturbed parameter vectors, crossover is introduced. To this end, the trial vector:

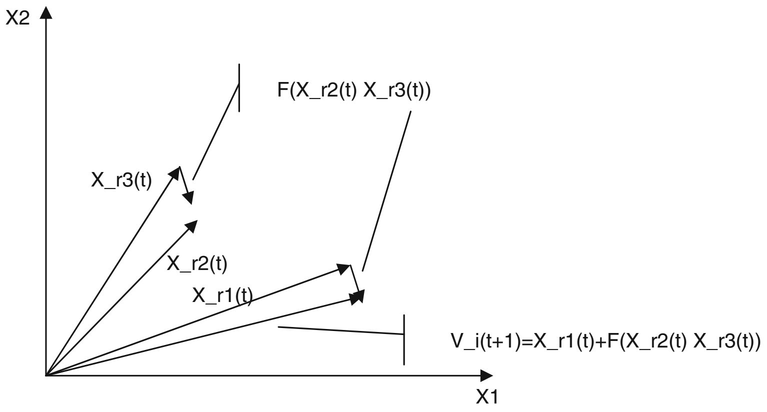
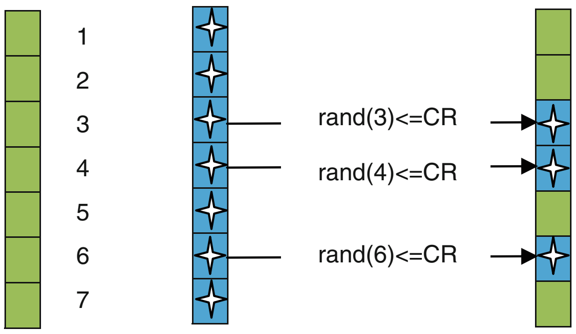


Fig. 2.2 An example of the process for generating $V_i(t+1)$ (two-dimensional)

Fig. 2.3 Illustration of the crossover process for $D = 7$ parameters (from [25])



$$U_i(t+1) = [u_{i,1}(t+1), u_{i,2}(t+1), \ldots, u_{i,d}(t+1)], \quad i = 1, 2, \ldots, NP \tag{2.6}$$

is formed, where

$$u_{i,j}(t+1) = \begin{cases} v_{i,j}(t+1), & \text{if } (rand(i,j) \leq CR) | j = randn(i) \\ x_{i,j}(t), & \text{otherwise} \end{cases} \tag{2.7}$$

In (2.7), $rand(i, j)$ is the j-th evaluation of a uniform random number generator with outcome within [0, 1]. CR is the crossover constant, within [0, 1], which has to be determined by the user. $randn(i)$ is a randomly chosen index $1, 2, \ldots, d$ which ensures that $U_i(t+1)$ gets at least one parameter from $V_i(t+1)$. Figure 2.3 gives an example of the crossover mechanism for 7-dimensional vectors.

C. Selection

To decide whether or not the trial vector $U_i(t+1)$ should become a member of generation $t+1$, it is compared to the target vector $x_i(t)$ using the greedy criterion. If vector $u_i(t+1)$ yields a smaller fitness function value than $x_i(t)$, then $x_i(t)$ is set to $u_i(t+1)$; otherwise, the old value $x_i(t)$ is retained. DE follows a standard evolutionary algorithm flow, which has been described in Chap. 1.

DE variants

It can be seen that mutation (2.5) is the most important operation which controls the search of DE. There are at least six kinds of mutation operators. In this book, we normally use the DE/best/1 strategy [25], which is shown as follows:

$$V_i(t+1) = x_{best}(t) + F(x_{r2}(t) - x_{r3}(t)) \tag{2.8}$$

Compared to (2.5), the base vector is selected to be the current best candidate, so that the best information can be shared among the newly generated population. The advantage is that the convergence can significantly be enhanced, especially for complex problems [26, 34]. If the population size is not too small, the risk of premature convergence is low. The strategy in (2.5) called DE/rand/1. This strategy maintains high diversity of the population, which is also widely used. Nevertheless, its convergence speed is often slower.

2.4 Basics of Constraint Handling Techniques

The constraint handling problem is very important in many real-world optimization problems, including analog circuit sizing. For high-performance (with tough specifications) circuits, the optimization problems are always highly constrained. However, the evolutionary algorithms, serving as the optimization engine, are constraint blind. Hence, constraint handling technique is a hot topic in the EC field. This section will introduce two widely applied constraint handling methods. They are often used in current literatures for analog IC sizing.

2.4.1 Static Penalty Functions

Most of the reported synthesis methods use static penalty functions to handle constraints [23, 35]. In these methods, the constrained optimization problem is transformed into an unconstrained one by minimizing the following function (a minimization problem is considered):

$$f'(x) = f(x) + \sum_{i=1}^{i=n} w_i < g_i(x) > \tag{2.9}$$

where the parameters w_i are the penalty weighting coefficients. "Static" means the penalty weighting coefficients are decided beforehand and are not changed during the optimization process. $\langle g_i(x) \rangle$ returns the absolute value of $g_i(x)$ if its negative, and zero otherwise, considering the constraints of $g_i(x) \geq 0, i = 1, 2, \ldots, n$. $f(x)$ is the objective function.

Clearly, the advantage of the static penalty function-based method is its simplicity. Only the fitness function is properly formulated and all the evolutionary operators do not need any adaptation or revision. On the other hand, the outcome of approaches based on static penalty function techniques is sensitive to the values chosen for the penalty coefficients, but the determination of proper penalty coefficients is a tough work. Small values of the penalty coefficients drive the search outside the feasible region and often produce infeasible solutions, whereas imposing very severe penalties makes it difficult to drive the population to the optimum. Although several penalty strategies have been developed, there has been no general rule for designing penalty coefficients. As the experiments in Chap. 3 will demonstrate, the methodology based on the combination of genetic algorithms and penalty functions cannot successfully solve design problems with many or severe constraints.

2.4.2 Selection-Based Constraint Handling Method

A constraint handling algorithm based on tournament selection for genetic algorithms that has proven to be effective was proposed by [36]. Prior to this, identical separation of feasible and infeasible solutions had been proposed in combination with simulated annealing by analog IC sizing researchers [13, 37]. Given two candidates in the population, there may be, at most, three situations:

(1) Both solutions are feasible;
(2) Both solutions are infeasible;
(3) One solution is feasible, but the other is not.

Accordingly, the selection rules are:

(1) Given two feasible solutions, select the one with the better objective function value;
(2) Given two infeasible solutions, select the solution with the smaller constraint violation;
(3) If one solution is feasible and the other is not, select the feasible solution.

The most important advantage is that this method needs no penalty coefficients, so the problems of penalty function-based algorithms are overcome. The drawback is that some candidates with very good performances may be missing in the search process because of loss of diversity. This deficiency will occur in problems with disconnected feasible regions in which cases the EA may be stuck within one of the feasible regions and never get to explore the others [38]. To illustrate this phenomenon, let us consider the performance space in Fig. 2.4. Assume that the regions in grey represent the feasible regions (they are disconnected) and that the global optimum is in the left one. In the beginning, the evolution may follow a fast way to minimize the total violation of constraints without considering the objective function, causing some decision variable(s) swiftly go(es) into one of the feasible intervals and is / are restricted there. In this example, the population after some iterations may converge in the feasible region on the right, but the global optimal value for the objective function is in the feasible region on the left.

Although these two methods have some drawbacks, they are often workable for analog circuit sizing problems with ordinary constraints. For highly constrained or problems with complex hyper-surfaces, which may appear in high-performance analog circuit sizing, advanced methods are necessary.

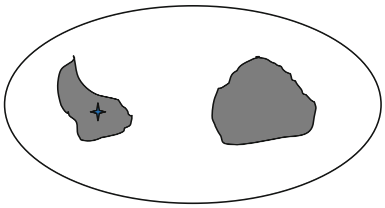


Fig. 2.4 Illustrative solution space

2.5 Multi-objective Analog Circuit Sizing

Multi-objective analog circuit sizing is based on multi-objective evolutionary algorithms (MOEAs). The main difference between multi-objective optimization and single-objective optimization is the fitness assignment. For single-objective optimization, the optimality is determined by a single function value, but for multi-objective optimization, not only the optimality should be determined by multiple objective functions, but also the distribution of the solutions in the approximated Pareto front (PF) is important. MOEAs can be generally classified into non-dominate sorting-based methods and decomposition-based methods. This section will introduce non-dominated sorting genetic algorithm-II (NSGA-II) [39] and multi-objective evolutionary algorithm based on decomposition (MOEA/D) [40], which are widely used in multi-objective sizing of analog circuits.

Before introducing MOEAs, two essential concepts for Pareto-optimal multi-objective optimization are described.

- Dominance
  Considering that there are two objectives $f_1(x)$ and $f_2(x)$, and both of them are for minimization, let x and x' be two candidate solutions for the multi-objective optimization problem. x is said to dominate x' if and only if $f_1(x) \leq f_1(x')$, $f_2(x) \leq f_2(x')$, and at least one of these two inequalities is strict. A solution x^* is Pareto-optimal if there is no other solution that dominates it. The set of all the Pareto-optimal solutions is called the Pareto set (PS) and the image of PS in the objective space (i.e., $f_1 - f_2$ space) is the Pareto front (PF). All the Pareto-optimal solutions are considered as equally good if there is no specific preference.
- Diversity of the approximated PF
  A decision maker often wants to have an approximate PF for gaining more understanding of the problem to make his / her final decision. Note that in most MOEAs, the PF is approximated by a number of points (Pareto-optimal solutions). Therefore, it is desirable that the generated Pareto-optimal points spread evenly in the approximated PF (high diversity), instead of clustered to a/several small part(s) of the PF (low diversity), where no information can be gained from the blanks in the approximated curves or hyper-surfaces.

2.5.1 NSGA-II

NSGA-II uses non-dominated sorting for fitness assignments. As has been said, in multi-objective sizing, not only good convergence is the optimization goal, but also the diversity of the points in the PF is important. Thus, environmental selection, that considers both dominance and distance between individuals, is necessary [41]. The truncation method is the approach for environmental selection very commonly used in recent years for multi-objective optimization problems with 2–3 objectives [41] and is also used in NSGA-II. A typical truncation method proceeds as follows:

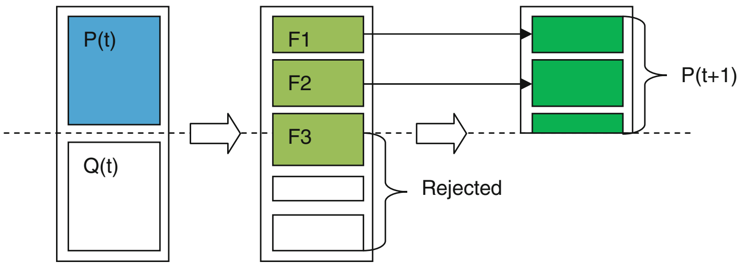


Fig. 2.5 Truncation method used in NSGA-II from [39]

(1) Collect all the candidates that compete for the next generation and rank them on dominance.
(2) If the number of non-dominated solutions exceeds the size of the population, then the non-dominated solutions are ranked by distance. The solutions with ranks larger than the population size are cut off.
(3) If the number of non-dominated solutions is less than the population size, then fill in the available places with dominated solutions according to their quality on dominance and distribution.

Figure 2.5 shows the truncation method, where P means the parent population, Q means the offspring population and F means the number of fronts.

The front number assigned to each individual is ranked by the level of other individuals that dominate it. The pseudocode of the fast non-dominate sorting is in Fig. 2.6 [39]. In order to make the algorithm evolve towards a uniformly spread out PF, crowding distance is used and is considered in the ranking. Crowding distance estimates the density of solutions surrounding a particular solution in the population. It calculates the average distance of two points on either side of this point along each of the objectives. For each objective, the population is firstly sorted. Then, the solutions with smallest and largest function values are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. The crowding-distance value is the sum of the normalized distance values of each objective.

The search engine of NSGA-II is a real coded GA. The crossover operator used is simulated binary crossover (SBX). The procedure of generating two children c_1 and c_2 from two parent solutions p_1 and p_2 is as follows:

1. Create a random number u between 0 and 1.
2. Calculate coefficient β:

$$\beta = \begin{cases} (2u)^{1/(\eta+1)} & \text{if } u \leq 0.5 \\ (\frac{1}{2(1-u)})^{1/(\eta+1)}, & otherwise \end{cases} \tag{2.10}$$

```
fast-non-dominated-sort($P$)
for each $p \in P$
   $S_p = \emptyset$
   $n_p = 0$
   for each $q \in P$
      if $(p \prec q)$ then                If $p$ dominates $q$
         $S_p = S_p \cup \{q\}$            Add $q$ to the set of solutions dominated by $p$
      else if $(q \prec p)$ then
         $n_p = n_p + 1$                   Increment the domination counter of $p$
   if $n_p = 0$ then                       $p$ belongs to the first front
      $p_{\text{rank}} = 1$
      $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
$i = 1$                                    Initialize the front counter
while $\mathcal{F}_i \neq \emptyset$
   $Q = \emptyset$                         Used to store the members of the next front
   for each $p \in \mathcal{F}_i$
      for each $q \in S_p$
         $n_q = n_q - 1$
         if $n_q = 0$ then                 $q$ belongs to the next front
            $q_{\text{rank}} = i + 1$
            $Q = Q \cup \{q\}$
   $i = i + 1$
   $\mathcal{F}_i = Q$
```

Fig. 2.6 Fast non-dominate sorting method in NSGA-II from [39]

3. The children solutions are

$$\begin{aligned} c_1 &= 0.5[(1-\beta)p_1 + (1+\beta)p_2] \\ c_2 &= 0.5[(1+\beta)p_1 + (1-\beta)p_2] \end{aligned} \tag{2.11}$$

where the positive number η is the distribution index of the SBX operator. It can be seen that a small η would generate children solutions far away from the parent solutions, while a large η restricts children solutions to be near the parent solutions.

Essentially, the SBX operator has two properties:

- The difference between the offspring is in proportion to the parent solution.
- Near-parent solutions become mostly offspring rather than solutions distant from parents if η is properly selected.

The mutation operator used in NSGA-II is polynomial mutation. The probability distribution is a polynomial function. The shape of the probability distribution is directly controlled by an external parameter γ_m, and the distribution is not dynamically changed with iterations. If x_i is the value of the ith parameter selected for mutation with a probability p_m and the result of the mutation is the new value y_i obtained by a polynomial probability distribution $Pr(\delta) = 0.5(\gamma_m + 1)(1 - |\delta|)^{\gamma_m}$, then:

$$
\begin{aligned}
& y_i = x_i + (x_i^H - x_i^L)\delta_i \\
& \delta_i = \begin{cases} (2r_i)^{1/(\gamma_m+1)-1} & \text{if } r_i < 0.5 \\ 1 - |2(1-r_i)|^{1/(\gamma_m+1)}, & \text{otherwise} \end{cases}
\end{aligned} \tag{2.12}
$$

where x_i^L and x_i^H are the lower and upper bound of x_i respectively and r_i is a random number in [0, 1].

2.5.2 MOEA/D

MOEA/D is a recently proposed MOEA [40], which outperforms NSGA-II for complex problems and problems with more objectives (e.g., 5). By using a (linear or nonlinear) weighted aggregation method, the approximation of the PF can be decomposed into N single objective optimization subproblems. MOEA/D defines neighbourhood relations among these subproblems based on the distances among their weight vectors. Each subproblem is optimized in MOEA/D by using information mainly from its neighbouring subproblems. Among several decomposition methods, the Tchebycheff approach is the most widely used one. More specifically, the scalar function is as follows:

$$
\begin{aligned}
& \text{minimize} && g^{te}(x|\lambda, z^*) = max_{1\le i\le m}\{\lambda_i|f_i(x) - z_i^*|\} \\
& s.t. && x \in [a,b]^d
\end{aligned} \tag{2.13}
$$

where $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_m\}$ is a weight vector and $\sum_{i=1}^{m} \lambda_i = 1$. $[a,b]^d$ is the solution space and $z^* = \{z_1^*, \ldots, z_m^*\}$ is the reference point. In other words, z^* are the smallest values of each objective function. If N is reasonably large and $\lambda^1, \ldots, \lambda^N$ are properly selected, the optimal solutions to those scalar functions will provide a good approximation to the Pareto Set (PS)/PF. It is worth noting that as z^* is usually unknown before the search, the smallest values of each objective found during the search are often used to substitute them. There are a number of different variants of MOEA/D. The original MOEA/D uses the SBX and polynomial mutation introduced in the previous subsection. MOEA/D-DE was proposed in [42] and high performance is shown.

MOEA/D-DE is described as follows.[2]

Input:

(1) an multi-objective optimization problem (MOP)
(2) a stopping criterion
(3) N: the number of sub-problems

[2] Note that in the algorithmic description for multi-objective optimization, to avoid the confusion of indices in a vector and the indices in a group of vectors, a superscript i indicates the ith individual of a group, and a subscript i indicates the ith element in a vector.

(4) T: the neighborhood size
(5) δ: the probability that parent solutions are selected from the neighborhood
(6) n_r: the maximum number of solutions replaced by a child solution
(7) CR: the crossover rate in DE
(8) $\widehat{F}$: the scaling factor in the DE mutation
(9) p_m: the probability to perform polynomial mutation
(10) λ: the weight vector (the generation method is in [40])

Output:

(1) Approximation to the PF
(2) Approximation to the PS

Procedure:

Step 1: Initialization

Step 1.1: Compute the Euclidean distances between the weight vectors and work out the T closest weight vectors to each weight vector. For $i = 1, \ldots, N$, set $B(i) = \{i_1, \ldots, i_T\}$. $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest vectors to λ^i.
Step 1.2: Randomly generate an initial population $x^1, \ldots, x^N$. Calculate the fitness values of the population.
Step 1.3: Initialize $z = \{z_1, \ldots, z_m\}$, where $z_j = min_{1 \le i \le N} f_j(x^i)$.

Step 2: Update
For $i = 1, \ldots, N$,

Step 2.1: Selection of the mating pool:
Generate a random number "rand" which is uniformly distributed in the range [0,1]. Set

$$P = \begin{cases} B(i), & \text{if } rand < \delta \\ \{1, \ldots, N\}, & \text{otherwise} \end{cases} \tag{2.14}$$

Step 2.2: Reproduction:
Set $r_1 = i$ and randomly select two indexes r_2 and r_3 from P, and generate a new solution $\bar{y}$ by the DE mutation. Then, perform a polynomial mutation on $\bar{y}$ with probability p_m to produce a new solution y.
Step 2.3: Repair:
If an element of y is out of the bound of $[a, b]^d$, its value is reset to be a randomly selected value inside the boundary.
Step 2.4: Update of the reference point:
For $j = 1, \ldots, m$, if $z_j > f_j(y)$, set $z_j = f_j(y)$.
Step 2.5: Update of solutions:
Set $c = 0$ and then do the following:
(1) If $c = n_r$ or P is empty, go to Step 3. Otherwise, randomly pick an index j from P.
(2) If $g^{te}(y|\lambda^j, z) \le g^{te}(x^j|\lambda^j, z)$, then set $x^j = y$, and $c = c + 1$.
(3) Remove j from P and go to 1).

Step 3: Stopping Criterion:
If the stopping criterion (e.g., a certain number of iterations) is satisfied, then stop the algorithm and output $\{x^1, \ldots, x^N\}$ and $\{f(x^1), \ldots, f(x^N)\}$. Otherwise, go to Step 2.

2.6 Analog Circuit Sizing Examples

This section shows the application of the optimization techniques mentioned above. According to the previous sections, we have several effective single- and multi-objective evolutionary algorithms (GA, DE, PSO, ES, NSGA-II, MOEA/D) and two constraint handling methods (static penalty function and selection-based methods). To compose a method for analog circuit sizing, we can combine methods from the two pools. For example, for single-objective analog circuit sizing problem, the method of GA with static penalty function methods can be used. For multi-objective optimization without constraints, MOEA/D-DE can be used. Note that in the framework of MOEA/D, it is difficult to directly cooperate with a constraint handling method. This issue will be further discussed in Chap. 6.

Two examples are provided in the following: the first one is an analog circuit sizing problem based on single-objective constrained optimization, and the other is based on multi-objective constrained optimization.

2.6.1 Folded-Cascode Amplifier

The example circuit selected in this section is a folded-cascode amplifier. The folded-cascode amplifier is shown in Fig. 2.7. The load capacitance *cl* is 5 pF. The technology used is a 0.18 μm CMOS process with 1.8 V power supply. Transistor lengths were allowed to vary between the minimum value allowed by the technological process, 0.18 μm, up to 10 μm. Transistor widths were changed between the minimum technology value, 0.24 μm, up to 1000 μm. The bias current was allowed to vary between 1 μA and 2.5 mA. Appropriate matching relations were imposed: $M1 \equiv M2$, $M3 \equiv M4 \equiv Mbp$, $Mbn \equiv M5$, $M6 \equiv M7$, $M8 \equiv M9$, $M10 \equiv M11$. Therefore, the number of independent design parameters is 13.

2.6.2 Single-Objective Constrained Optimization

For the automated sizing of the above folded-cascode amplifier, the optimization goal is to minimize the power consumption with constraints on DC gain, gain-bandwidth product (GBW), phase margin, output swing, slew rate, area consumption and the functional constraints that all the transistors should be in the saturation region.

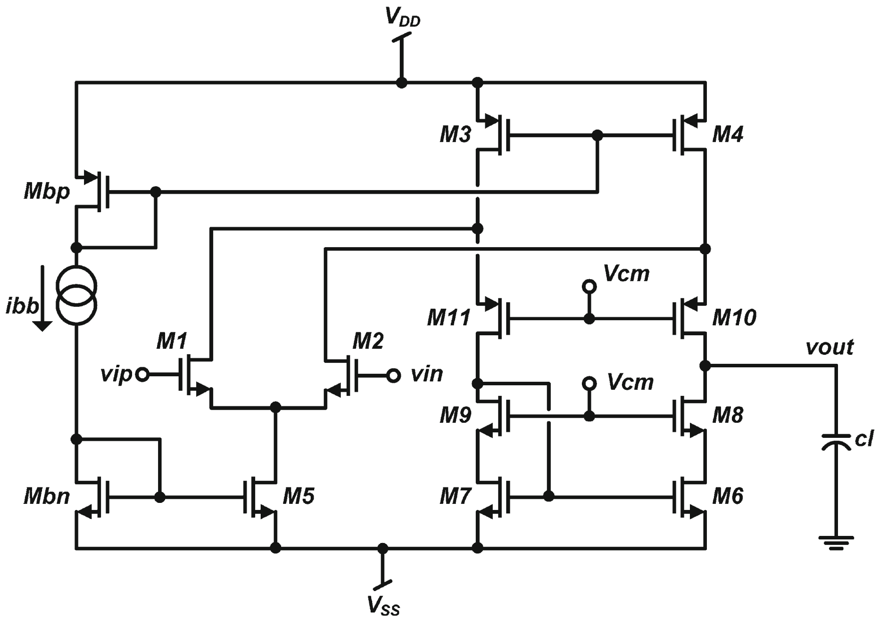


Fig. 2.7 CMOS Folded-cascode amplifier

The following methods are implemented: (1) the genetic algorithm, combined with penalty functions to handle constraints (GAPF); (2) the differential evolution algorithm combined with the same penalty-based method to handle constraints (DEPF); (3) an algorithm that combines the selection-based method for constrained optimization proposed by [36] and differential evolution (denoted as SBDE) [43]. For methods using DE, the DE/best/1 strategy is used. The inputs to the design methodology are a SPICE netlist file containing the topology and user-defined performance simulations. All examples have been run on a workstation with a Dual Xeon Quad processor at 3 GHz with 33 GB of RAM memory and Linux operating system.

Relatively soft performance constraints are used first. The design objective (power minimization) and constraints are shown in Table 2.1 together with the results provided by the three methods, GAPF, DEPF and SBDE. The penalty coefficients in the GAPF and DEPF algorithms were manually improved through five runs. At each new run, the penalty coefficients were updated, trying to increase the relative importance of the constraints not met in the previous run. Table 2.1 shows the best results among the five runs. It can be seen that all the algorithms met the design specifications.

Being a stochastic search process, a statistical study is needed to test the robustness of the different algorithms. Therefore, we executed 20 runs of each algorithm starting from 20 different initializations. For the GAPF and DEPF algorithms a reasonable choice of the penalty parameters was used for this statistical study: all constraints were normalized with respect to the specified value and equal weights were assigned to all of them. Table 2.2 shows the results for the different algorithms. The first two

Table 2.1 Design specifications and results of GAPF, DEPF and SBDE (not severe constraints)

Specifications	Constraints	GAPF	DEPF	SBDE
$DC\ gain$ (dB)	$\geq$55	62.14	55.03	56.87
GBW (MHz)	$\geq$2	2.05	2.12	2.03
$Phase\ margin$ (°)	$\geq$50	88.25	83.58	82.09
$Output\ swing$ (V)	$\geq$1.2	1.31	1.39	1.28
$Slew\ rate$ (V/μs)	$\geq$1	1.13	1.00	1.02
$Area$ (μm^2)	$\leq$225	196.95	150.53	142.16
$M1$	Saturation	met	met	met
$M4$	Saturation	met	met	met
$M5$	Saturation	met	met	met
$M6$	Saturation	met	met	met
$M8$	Saturation	met	met	met
$M10$	Saturation	met	met	met
$Power$ (mW)	objective	0.028	0.024	0.023
$Time$ (s)	N.A.	165	151	163

Table 2.2 Statistical results with the different methods (not severe constraints)

Item	Feasible	Infeasible	Best	Worst	Average
GAPF	20	0	0.0219	0.0473	0.0283
DEPF	20	0	0.0221	0.0282	0.0234
SBDE	20	0	0.0216	0.0305	0.0236

columns show the number of feasible and infeasible solutions found in the 20 runs. It can be seen that all methods were able to find a feasible solution. The following three columns in Table 2.2 show the best, worst and average value of the power consumption obtained in those 20 runs.

A set of more severe design constrains is shown in Table 2.3, together with a typical result provided by all algorithms. It can be observed that neither the GAPF algorithm nor the DEPF algorithm is able to satisfy the design constraints, even though five different sets of penalty coefficients were tried.

2.6.3 Multi-objective Optimization

For the same folded-cascode amplifier, the two optimization objectives are power and area, and the constraints include specifications on DC gain, GBW, phase margin, output swing and the functional constraints of all the transistors should be in the saturation region. This is a constrained multi-objective optimization problem. NSGA-II is used for multi-objective optimization. To handle constraints, the selection-based methods can be used to update the truncation method in NSGA-II. Feasibility (the

Table 2.3 Design specifications and results of GAPF, DEPF and SBDE (severe constraints)

Specifications	Constraints	GAPF	DEPF	SBDE
$DC\ gain$ (dB)	≥ 60	61.89	60	60.06
GBW(MHz)	≥ 80	3.13	51.13	80.05
$Phase\ margin$ (°)	≥ 60	79.59	78.84	74.74
$Output\ swing$ (V)	≥ 1.25	1.39	1.32	1.26
$Slew\ rate$ (V/μs)	≥ 60	1.56	33.97	60.00
$Area$ (μm^2)	≤ 440	330.22	320.20	431.73
$M1$	Saturation	met	met	met
$M4$	Saturation	met	met	met
$M5$	Saturation	met	met	met
$M6$	Saturation	met	met	met
$M8$	Saturation	met	met	met
$M10$	Saturation	met	met	met
$Power$ (mW)	objective	0.03	0.74	1.31
$Time$ (s)	N.A.	173	161	209

Table 2.4 Experimental results with two sets of constraints

Specifications	Constraints	Average	Constraints	Average
$DC\ gain$ (dB)	≥ 60	60.05	≥ 55	55.05
GBW (MHz)	≥ 50	50.03	≥ 40	40.02
$Phase\ margin$ (°)	≥ 60	82.60	≥ 60	85.72
$Output\ swing$ (V)	≥ 1.2	1.25	≥ 1	1.23
$Slew\ rate$ (V/μs)	≥ 30	34.00	≥ 20	22.86
$dm1$	≥ 1	17.64	≥ 1	17.58
$dm3$	≥ 1	1.81	≥ 1	1.67
$dm5$	≥ 1	3.52	≥ 1	4.24
$dm7$	≥ 1	2.42	≥ 1	2.23
$dm9$	≥ 1	1.56	≥ 1	1.79
$dm11$	≥ 1	7.56	≥ 1	8.28
$Time\ (s)$	N.A.	1706	X	1672

sum of the violation of constraints) is assigned superiority, and the truncation method in NSGA-II is only used for feasible individuals.

Two experiments have been performed with the severe constraints indicated in the second column of Table 2.4 and a second experiment with the not severe constraints from the fourth column. One typical solution is plotted for them, respectively. The power versus area Pareto fronts are shown in Fig. 2.8. The high specifications (severe constraints) and low specifications (not severe constraints) and the average values of the 200 individuals in the typical solution on each set of specifications are shown in Table 2.4.

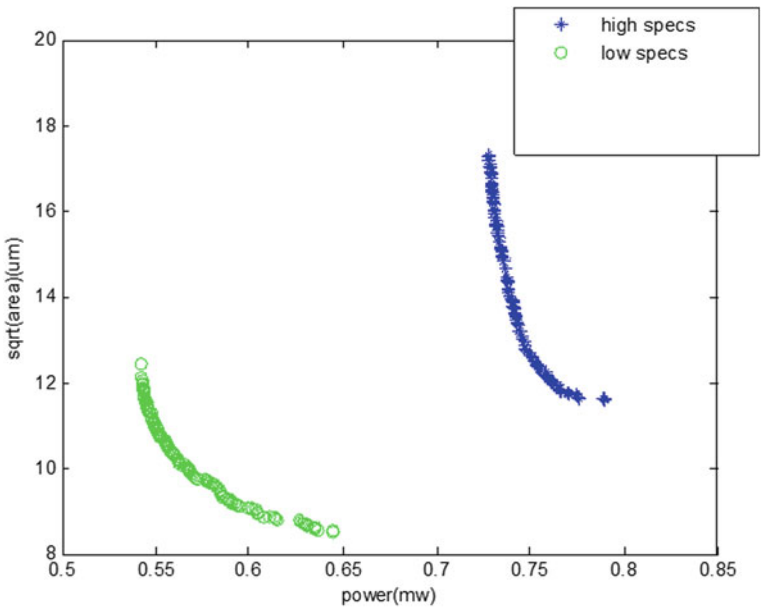


Fig. 2.8 Two typical Pareto fronts obtained for the two sets of constraints

2.7 Summary

This chapter has introduced the basics of automated sizing of analog circuits. It has tried to answer the following two questions: (1) How to use the evolutionary computation-based optimization techniques to solve practical problems in electronic design automation field; (2) How to design a workable analog circuit sizing method. The analog IC sizing problem is formulated and is connected with evolutionary computation techniques. After reviewing the previously published automatic analog circuit sizing approaches, the widely used and promising evolutionary algorithms for the targeted problem have been reviewed, among which, the differential evolution algorithm and its implementation were paid special attention. Constraint handling techniques and multi-objective evolutionary algorithms are also important in analog IC sizing and other real-world problems. Hence, two constraint handling techniques have been introduced. They are easy to implement and are effective to many problems, especially the selection-based methods. Two multi-objective optimization methods have then been introduced. NSGA-II is the most widely used method in multi-objective analog IC sizing and MOEA/D is a new state-of-the-art multi-objective optimization method, which is gradually becoming popular in the analog IC sizing field. At last, two practical examples have been shown to illustrate how to design an analog circuit sizing approach.

References

1. Nam D, Seo Y, Park L, Park C, Kim B (2001) Parameter optimization of an on-chip voltage reference circuit using evolutionary programming. IEEE Trans Evol Comput 5(4):414–421
2. Martens E, Gielen G (2008) Classification of analog synthesis tools based on their architecture selection mechanisms. Integr VLSI J 41(2):238–252
3. Krasnicki M, Phelps R, Hellums J, McClung M, Rutenbar R, Carley L (2001) ASF: a practical simulation-based methodology for the synthesis of custom analog circuits. In: Proceedings of IEEE/ACM international conference on computer aided design, pp 350–357
4. Degrauwe M, Nys O, Dijkstra E, Rijmenants J, Bitz S, Goffart L, Vittoz E, Cserveny S, Meixenberger C, Van Der Stappen G et al (1987) IDAC: an interactive design tool for analog CMOS circuits. IEEE J Solid-State Circuits 22(6):1106–1116
5. Harjani R, Rutenbar R, Carley L (1989) OASYS: a framework for analog circuit synthesis. IEEE Trans Comput Aided Des Integr Circuits Syst 8(12):1247–1266
6. Makris C, Toumazou C (1995) Analog IC design automation. ii. automated circuit correction by qualitative reasoning. IEEE Trans Comput Aided Des Integr Circuits Syst 14(2):239–254
7. Ochotta E, Rutenbar R, Carley L (1996) Synthesis of high-performance analog circuits in ASTRX/OBLX. IEEE Trans Comput Aided Des Integr Circuits Syst 15(3):273–294
8. Gielen G, Walscharts H, Sansen W (1990) Analog circuit design optimization based on symbolic simulation and simulated annealing. IEEE J Solid-State Circuits 25(3):707–713
9. Maulik P, Carley L, Allstot D (1993) Sizing of cell-level analog circuits using constrained optimization techniques. IEEE J Solid-State Circuits 28(3):233–241
10. Boyd S, Lee T et al (2001) Optimal design of a CMOS op-amp via geometric programming. IEEE Trans Comput Aided Des Integr Circuits Syst 20(1):1–21
11. Nye W, Riley D, Sangiovanni-Vincentelli A, Tits A (1988) DELIGHT. SPICE: an optimization-based system for the design of integrated circuits. IEEE Trans Comput Aided Des Integr Circuits Syst 7(4):501–519
12. Phelps R, Krasnicki M, Rutenbar R, Carley L, Hellums J (2000) Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search. IEEE Trans Comput Aided Des Integr Circuits Syst 19(6):703–717
13. Medeiro F, Rodríguez-Macías R, Fernández F, Domínguez-Castro R, Huertas J, Rodríguez-Vázquez A (1994) Global design of analog cells using statistical optimization techniques. Analog Integr Circuits Signal Process 6(3):179–195
14. Castro-López R, Fernández F, Guerra-Vinuesa O (2006) Re-use based methodologies and tools in the design of analog and mixed-signal integrated circuits. Springer, Berlin
15. Stehr G, Pronath M, Schenkel F, Graeb H, Antreich K (2003) Initial sizing of analog integrated circuits by centering within topology-given implicit specification. In: Proceedings of the IEEE/ACM international conference on computer-aided design, pp 241–246
16. Balkir S, Dundar G, Alpaydin G (2004) Evolution based synthesis of analog integrated circuits and systems. In: Proceedings of NASA/DoD conference on evolvable hardware, pp 26–29
17. Takemura K, Koide T, Mattausch H, Tsuji T (2004) Analog-circuit-component optimization with genetic algorithm. In: Proceedings of the 47th midwest symposium on circuits and systems, vol 1, pp 489–492
18. Barros M, Neves G, Guilherme J, Horta N (2005) An evolutionary optimization kernel with adaptive parameters applied to analog circuit design. In: Proceedings of international symposium on signals, circuits and systems, vol 2, pp 545–548
19. Goh C, Li Y (2001) GA automated design and synthesis of analog circuits with practical constraints. In: Proceedings of the congress on evolutionary computation, vol 1, pp 170–177
20. Koza J, Bennett F III (1997) Automated synthesis of analog electrical circuits by means of genetic programming. IEEE Trans Evol Comput 1(2):109–128
21. Alpaydin G, Balkir S, Dundar G (2003) An evolutionary approach to automatic synthesis of high-performance analog integrated circuits. IEEE Trans Evol Comput 7(3):240–252

22. Kruiskamp W, Leenaerts D (1995) DARWIN: CMOS opamp synthesis by means of a genetic algorithm. In: Proceedings of the 32nd annual ACM/IEEE design automation conference, pp 433–438
23. Antao B, Gielen G, Rutenbar R (2002) Computer-aided design of analog integrated circuits and systems. Wiley, USA
24. Synopsys (2013) HSPICE homepage. http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx
25. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359
26. Price K, Storn R, Lampinen J (2005) Differential evolution: a practical approach to global optimization. Springer, New York
27. Michalewicz Z (1996) Genetic algorithms+ data structures. Springer, New York
28. Deb K, Agrawal R (1995) Simulated binary crossover for continuous search space. Complex Syst 9(2):115–148
29. Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm Intell 1(1):33–57
30. Rechenberg I (1994) Evolution strategy. Computational intelligence: imitating. Life 1:147–159
31. Hansen N (2006) The CMA evolution strategy: a comparing review. Towards a new evolutionary computation. Springer, pp 75–102
32. Gregory M, Bayraktar Z, Werner D (2011) Fast optimization of electromagnetic design problems using the covariance matrix adaptation evolutionary strategy. IEEE Trans Antennas Propag 59(4):1275–1285
33. Lagarias J, Reeds J, Wright M, Wright P (1998) Convergence properties of the Nelder-Mead simplex method in low dimensions. Siam J Optim 9:112–147
34. Chakraborty U (2008) Advances in differential evolution. Springer, Heidelberg
35. Liu B, Wang Y, Yu Z, Liu L, Li M, Wang Z, Lu J, Fernández F (2009) Analog circuit optimization system based on hybrid evolutionary algorithms. Integr VLSI J 42(2):137–148
36. Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186(2):311–338
37. Medeiro F, Fernández F, Dominguez-Castro R, Rodriguez-Vazquez A (1994) A statistical optimization-based approach for automated sizing of analog cells. In: Proceedings of the IEEE/ACM international conference on computer-aided design, pp 594–597
38. Venkatraman S, Yen G (2005) A generic framework for constrained optimization using genetic algorithms. IEEE Trans Evol Comput 9(4):424–435
39. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
40. Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
41. Coello C, Lamont G, Veldhuizen D (2007) Evolutionary algorithms for solving multi-objective problems. Springer, New York
42. Li H, Zhang Q (2009) Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. IEEE Trans Evol Comput 13(2):284–302
43. Zielinski K, Laur R (2006) Constrained single-objective optimization using differential evolution. In: Proceedings of IEEE congress on evolutionary computation, pp 223–230