Pareto Optimization of Analog Circuits Using Reinforcement Learning

KARTHIK SOMAYAJI NS and PENG LI, University of California, Santa Barbara, USA

Analog circuit optimization and design presents a unique set of challenges in the IC design process. Many applications require the designer to optimize for multiple competing objectives, which poses a crucial challenge. Motivated by these practical aspects, we propose a novel method to tackle multi-objective optimization for analog circuit design in continuous action spaces. In particular, we propose to (i) extrapolate current techniques in Multi-Objective Reinforcement Learning to continuous state and action spaces and (ii) provide for a dynamically tunable trained model to query user defined preferences in multi-objective optimization in the analog circuit design context.

CCS Concepts: • **Hardware** → **Circuit optimization;**

Additional Key Words and Phrases: Analog circuit optimization, machine learning, Pareto optimization

ACM Reference Format:
Karthik Somayaji NS and Peng Li. 2024. Pareto Optimization of Analog Circuits Using Reinforcement Learning. *ACM Trans. Des. Autom. Electron. Syst.* 29, 2, Article 37 (February 2024), 14 pages. https://doi.org/10.1145/3640463

1 INTRODUCTION

In recent years, innovations in the field of circuit design and embedded systems have led to rapid development of analog and digital IC design. With growing demand, it is the need of the hour to employ automated design technologies for the optimization of analog ICs in particular. In traditional IC design, human experts tune circuit parameters to ensure optimal functionality. However, the sheer number of design parameters coupled with complex device characteristics makes human design very laborious and time intensive. Additionally, in practical scenarios, where multiple objectives need to be optimized, the problem becomes even more challenging as the optimization process needs to be aware of the tradeoffs in the objectives.

In this work, we focus on analog design while optimizing the circuit performance for not just a single objective but for multiple objectives that may often have competing relationships. Due to the nature of complementary relations among different objectives in complex circuits, it becomes challenging to apply simple black-box optimization techniques. Although analytical methods exist, it becomes intractable to solve them for high-dimensional systems like analog circuits.

The funding for this project was provided by Semiconductor Research Corporation Task No. 2810.043 through UT Dallas' Texas Analog Center of Excellence, the U.S. Department of Energy's "Data-Driven Decision Control for Complex Systems (DnC2S)" project (research participation of the University of California at Santa Barbara under award number DE-SC0021321), and the National Science Foundation (NSF) under Grant No. 1956313.

Authors' address: K. S. NS and P. Li, Electrical & Computer Engineering, Harold Frank Hall, University of California Santa Barbara, CA 93106, USA; e-mails: karthi@ucsb.edu, lip@ucsb.edu.

ACM 1084-4309/2024/02-ART37
https://doi.org/10.1145/3640463

Motivated by the increased relevance of applying data-driven models to solve analog design, we propose a **reinforcement learning– (RL)** based reusable agent as a solution to optimize for multiple objectives in higher dimensions. We demonstrate the effectiveness of using RL over other methods in **multi-objective optimization (MOO)**. In MOO, due to the presence of tradeoffs between multiple objectives, there exists no single, unique optimal solution. However, a set of optimal design points exists as a solution. This set of optimal solutions is called the Pareto set. The aim is therefore to find the Pareto set as quickly as possible through the optimization process.

2 RELATED WORK

2.1 Evolutionary Algorithms

In the past, some works have studied the methods of generating the Pareto set by both analytical and data-driven methods. Analytical methods, [4], use line search approaches that depend on gradient information to find the Pareto set. However, increasing the dimensionality of the objective space limits an analytically solvable solution to the line search. Another popular method used is NSGA-II [3], which utilizes a genetic algorithm approach. This algorithm introduces a fast approach to find the dominance depth of each sample point in the population. After sampling an initial population of the design space, the genetic algorithm searches the design space using crossover and mutations. There is a fixed budget on the number of function evaluations, and the algorithm ends as this budget is reached. By carefully adjusting the crossover and mutations, the algorithm is able to achieve exploration of the state space.

2.2 Bayesian Optimization

Reference [15] utilizes Gaussian Processes and the Bayesian framework to query for points in the design space. Here, Pareto set generation is accomplished by iterative search and dominance depth assignment to the points sampled. The work in Reference [9] employs search over the objective space by using a combination of genetic algorithm and Bayesian optimization. The Gaussian process regression framework is used to provide a surrogate model that is then used for pre-selecting Pareto dominant points using genetic algorithms. However, Reference [6] proposes analog circuit sizing in two phases, alternating between Bayesian optimization and evolutionary Pareto front search using similar approaches to Reference [9] while introducing constraints on performance specifications and parasitics. Yet another work that uses Bayesian optimization for multi-objective optimization is Reference [14]. The authors of Reference [14] propose to reduce the time complexity of matrix inversion from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ using an incremental learning technique and uses a modified acquisition function matrix suited for multi-objective optimization. Reference [13] use similar self-adaptive incremental learning techniques to Reference [14] to use surrogate approximators for pre-selection of valid Pareto optima; design points for evolutionary algorithms to simulate. Reference [7] also use Bayesian optimization for LDE aware analog circuit sizing. Our proposed method is based on complete neural network implementations. The time complexity of operation of our method is independent of the number of samples collected unlike any of the Bayesian optimization-based approaches, which makes our method a lightweight optimization engine. Second, the predictive ability of the proposed method to generate new Pareto optimal points along a user specified preference direction during inference time makes our method tunable to specific user preferences during test time.

2.3 Reinforcement Learning

Reinforcement learning recently has been used in single-objective optimization or composite multi-objective optimization in circuit design in References [2, 8, 10, 11]. These works predominantly use off-policy RL algorithms under continuous action spaces. However, they do not consider different tradeoffs in design objectives to form a Pareto set. Another work that closely resembles our work is Reference [5]. Reference [5] proposes to perform single-objective optimization using a set of static weights to weight each objective and use this as a reward function in the reinforcement learning framework. However, this approach does not explicitly capture tradeoffs between different objectives in the optimization process. Thus our work is different from Reference [5] on three main algorithmic fronts.

(i) We use vectorized rewards instead of scalar rewards to specify the exact values of the objectives being optimized (rewards are not scalarized—for example α_1.Gain + α_2.UGF). (ii) We have a dynamically changing preference direction to weight the objectives. This is unlike fixed weights that Reference [5] uses. This dynamically changing preference direction G is used to find the Pareto optimal points on the Pareto front, which is a capability not explored by Reference [5]. (iii) We propose a vectorized version of the DDPG algorithm where the Q-function is itself vectorized to perform multi-objective optimization. This is unlike Reference [5] where Q values are scalars and do not consider tradeoffs between multiple objectives.

In the field of **multi-objective RL (MORL)** in general, the work in Reference [12] introduces methods to query for the Pareto set. But these approaches in MORL, deal with scenarios where the actions are discrete and countably few. For analog circuit sizing, although sizing solutions are typically on a grid, the number of possible values on the grid might be many in number and can be thought of as continuous. While using MORL for such problems, it calls for designing RL algorithms that operate with continuous actions instead of just discrete ones.

3 OUR CONTRIBUTIONS

In our work, we extend RL to optimize for multiple objectives in cases where actions(incremental sizing changes) are continuous valued and build a tradeoff-aware RL agent. We also demonstrate why RL-based approaches to Pareto optimization of analog circuits in particular, can have potential benefits over other data-driven methods like genetic algorithms and Bayesian Optimization. The key contributions of this work are as follows:

(1) We propose a sample-efficient and easy-to-train MORL algorithm to form a well-approximated Pareto set of the analog circuit, where the actions of the RL agent (fine-tuned sizing solutions) are continuous valued.
(2) Next, we use the predictive power of the trained RL agent to demonstrate that the RL agent supports querying the analog circuit for user-defined/custom preferences among objectives for which to optimize. Our work presents a model to query for unseen design points in the training process, based on the designer's choice, and helps augment the current Pareto set.
(3) We demonstrate, through extensive experiments, the effectiveness of using the proposed RL algorithm for black-box multi-objective analog circuit optimization. We illustrate the performance improvements of our algorithm over previous methods like NSGA-II, **Bayesian Optimization (BO)**, and Monte Carlo sampling. We provide a schematic flow of the proposed algorithm in Figure 1.

4 MULTI-OBJECTIVE OPTIMIZATION VIA REINFORCEMENT LEARNING

4.1 Problem Formulation

The goal in multi-objective optimization is to find a set of optimal solutions—the Pareto set. The solutions of the Pareto set do not dominate any other solution in the same set. A point $\mathbf{s_1}$ dominates another point $\mathbf{s}_2$ if $\forall i \in \{1, 2, \ldots m\}$, $f_i(\mathbf{s}_1) \geq f_i(\mathbf{s}_2)$, and $\exists j \in \{1, 2, \ldots, m\}$ such that $f_j(\mathbf{s}_1) > f_i(\mathbf{s}_2)$. Thus, we say $(\mathbf{s}_1 \succ \mathbf{s}_2)$. With the notion of dominance established, we want to find all the points $\{\mathbf{s}\}$ that are not dominated by any other point. The set of these points is called the Pareto set $\mathcal{S}$.

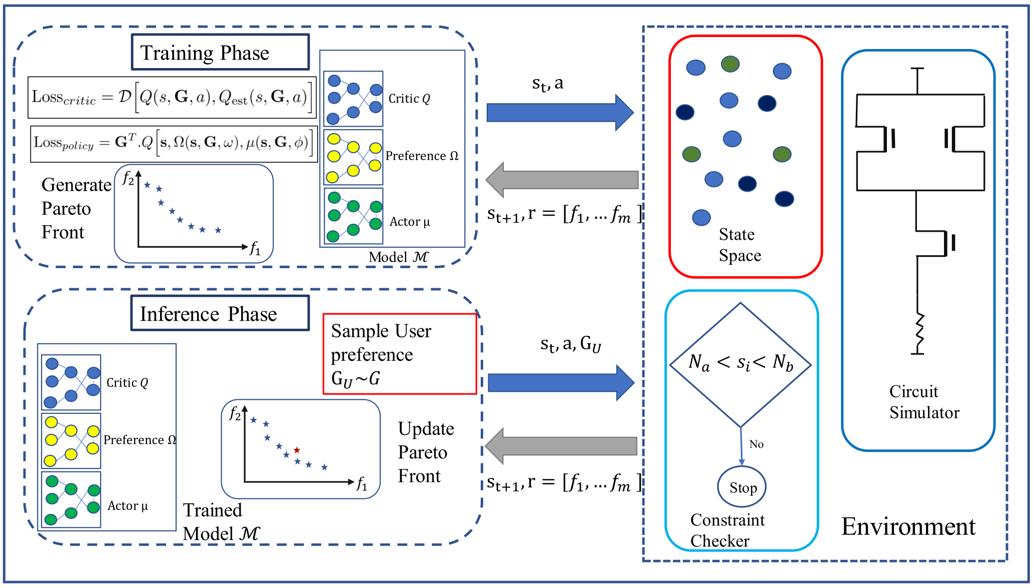

Training Phase
$\text{Loss}_{critic} = \mathcal{D}\Big[Q(s,\mathbf{G},a), Q_{\text{est}}(s,\mathbf{G},a)\Big]$
$\text{Loss}_{policy} = \mathbf{G}^T.Q\Big[\mathbf{s},\Omega(\mathbf{s},\mathbf{G},\omega),\mu(\mathbf{s},\mathbf{G},\phi)\Big]$
Generate Pareto Front
Critic Q
Preference Ω
Actor μ
Model $\mathcal{M}$
s_t, a
s_{t+1}, r = [f_1, ... f_m]
Inference Phase
Sample User preference $G_U \sim G$
s_t, a, G_U
Update Pareto Front
Trained Model $\mathcal{M}$
State Space
$N_a < s_i < N_b$
No
Stop
Constraint Checker
Circuit Simulator
Environment

Fig. 1. Flow Diagram illustrating our algorithm for Multi-objective Optimization using reinforcement learning.

Let us say that we are optimizing for multiple objectives given as

$$F(\mathbf{s}) = [f_1(\mathbf{s}), f_2(\mathbf{s}), \ldots, f_m(\mathbf{s})]^{\mathrm{T}}, \tag{1}$$

where $\mathbf{s} \in \mathbb{S} \subset \mathbb{R}^D$ and m is the number of objectives to be optimized and D is the dimensionality of the design space. Considering a maximization objective, our aim is to find

$$\max_{\mathbf{s} \subset \mathbb{R}^{\mathrm{D}}} F(\mathbf{s}). \tag{2}$$

The set of solutions $\mathbf{s} \in \mathcal{S}$, for the above equation belongs to the Pareto set and the resulting metric or **figure of merit (FOM)** values F, constitutes the Pareto front $\mathcal{P}$.

Next we briefly introduce some important concepts used in this work.

4.2 Goal Vector and Multi-Goal Reinforcement Learning Setup

To apply reinforcement learning to solve multi-objective optimization problems, we need to specify the internals of the RL agent. Keeping in mind the necessity to optimize for m objectives, we define the concept of a goal vector defined as

$$\mathbf{G} = [g_1, g_2, g_3, \ldots, g_m]^{\mathrm{T}}, \tag{3}$$

where $[g_i]_{i=1}^m$ is the preference or the weight associated with each objective to be optimized. The realization of the goal vector $\mathbf{G} \in \mathcal{G}$, is such that $\mathcal{G} \subseteq \{0,1\}^m$ and

$$\sum_{i=1}^{m} g_i = 1, \tag{4}$$

where $i = [1, 2, \ldots m]$.

Thus, the goal vector specifies the designer's intent to trade off among multiple competing objective values and specifies the direction of the search in the m-dimensional objective space. We next look at how the goal vector is included in the reinforcement learning flow. We first define the notion of the state vector s, which specifies the sizing solutions for the analog circuit,

$$\mathbf{s} = \left[\ldots, W_i, \ldots, L_j, \ldots\right]^{\mathrm{T}}, \tag{5}$$

where$W$ and L stand for the width and length of transistors respectively. We enforce the state space to be bounded within a certain normalized range, i.e., constrain each entry of the state vector $a < \mathbf{s}_i < b$ to adhere to the operating technology. We illustrate this in Figure 1 in the constraint-checker module.

To integrate the goal vector into the RL state, we simply concatenate the goal vector with the **s** and define it as the goal-state of the system $\mathbf{s}_G$,

$$\mathbf{s}_G = \left[\ldots, W_i, \ldots, L_j, \ldots, \mathbf{G}\right]^{\mathrm{T}}. \tag{6}$$

The goal-state of the system specifies the current sizing solution in addition to also giving information about the designer's intent or preference **G**. The goal-state can be thought of as an extension to the state space. The same sizing solution **s** could differ, based on the specified goal state **G**, in the state space according to designer's preference.

We define the action a, taken by the policy network of the RL agent, as the incremental change in the sizing solutions of the state vector **s**. Actions are sampled from the set of actions $\mathcal{A}$ that consist of continuous values. Although sizing in analog circuits is on a grid, the precision change in the sizing ($\delta L_i, \delta W_i$ - lengths and widths) can be thought of as continuous valued. This is also similar to the modelling choice chosen by Reference [2] for the definition of actions,

$$a = [\ldots, \delta W_i, \ldots, \delta L_i, \ldots]^{\mathrm{T}}. \tag{7}$$

Unlike the traditional RL framework, the reward in our work also targets at multi-objective optimization and thus is also m-dimensional. The reward vector has the same dimensionality as the number of objectives being maximized. The reward can be any of the FOM of the circuit performance that are competing. These FOMs are normalized between certain ranges, usually between 0 and +1,

$$r(\mathbf{s}, a) \in \{0, 1\}^m. \tag{8}$$

4.3 The Bellman Operator for Multi Objective Reinforcement Learning

In Q-learning, the Q value is a measure of goodness of a given state–action pair $(\mathbf{s}, a)$. The agent updates its estimate of this goodness through the Bellman equation given as

$$Q(\mathbf{s}, a) = r(\mathbf{s}, a) + \max_{a' \in \mathcal{A}} \gamma . Q(\mathbf{s}', a'), \tag{9}$$

where $r(.)$ is the reward and $\mathbf{s}'$ and a' indicate the next state and action seen in the RL trajectory.

As indicated in the work in Reference [12], we see that the Q function, in the multi-objective setting, unlike single-objective optimization, is also m dimensional. The Q value is a measure of goodness of a given goal-state–action pair $(\mathbf{s}_G, a)$. The inputs to the Q network are the goal-state $\mathbf{s}_G$ and the action a. The Bellman operator in the multi-objective setting is given by

$$\mathcal{T}Q(\mathbf{s}, \mathbf{G}, a) = r(\mathbf{s}, a) + \gamma \mathbb{E}_{\mathbf{s}' \sim P(.|\mathbf{s}, a)}[\mathcal{H}[Q(\mathbf{s}', \mathbf{G})]], \tag{10}$$

where the filter $\mathcal{H}$ is called the Bellman optimality filter and $P(\cdot \mid \mathbf{s}, a)$ is the transition probability and $\mathbf{s}'$ is the next state seen in the RL trajectory,

$$\mathcal{H}[Q(\mathbf{s}', \mathbf{G})] = \arg_Q \sup_{G' \in \mathcal{G}, a \in \mathcal{A}} \mathbf{G}^T \cdot Q(\mathbf{s}, G', a). \tag{11}$$

The optimality filter provides ($\arg_Q$), the Q value that maximizes the dot product $\mathbf{G}^T \cdot Q(\mathbf{s}, G', a)$. In other words, it finds the Q value in alignment with the preference **G**. In the single-objective case,

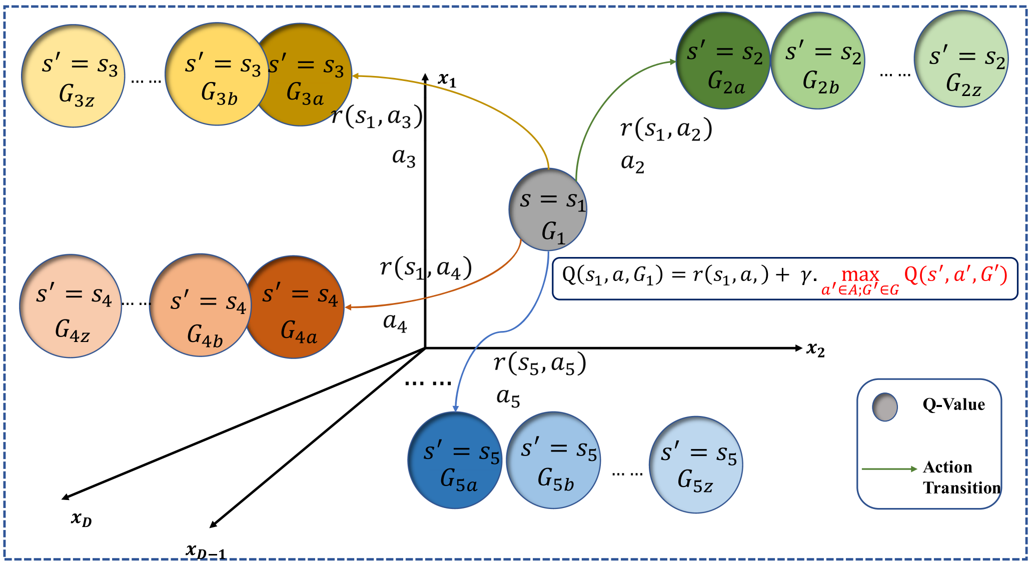


Fig. 2. Starting at sizing solution $\mathbf{s}_1$, and for each preference $\mathbf{G}_1$, the Q value update happens such that we search for a vector Q at the next sizing solution $\mathbf{s}'$, action $a' = \mu(\mathbf{s}', \mathbf{G})$, and preference $\mathbf{G}' = \Omega(\mathbf{s}', \mathbf{G})$ that maximizes the dot product in Equation (12). The figure above shows all possible values of a' and $\mathbf{G}'$. Each circle represents Q value for the action specified by the arrow and the goal-state specified within the circle.

this just reduces to finding the Q value for the best action, i.e., $\max_{a' \in \mathcal{A}} Q(\mathbf{s}, a')$. Thus, Equation (11) is a general form of the single-objective case. The authors of Reference [12] elucidate that the optimality operator $\mathcal{T}$ is a γ-contraction under a certain distance metric. The authors prove through the Banach fixed point theorem that the operator $\mathcal{T}$ has a fixed point, meaning that repeated application of the operator on the multi-objective Q function leads to convergence of the Q function. Figure 2 gives a visual intuition of the Q updates.

5 MULTI-OBJECTIVE REINFORCEMENT LEARNING FOR CONTINUOUS SIZING SOLUTIONS IN ANALOG CIRCUITS

With the background of the multi-dimensional Bellman update explained, let us consider that we plan to optimize for m objectives in a black-box system, all m of which may have competing relations. This means that optimizing for a set of objectives automatically compromises others. Each of the m dimensions of the Q network ascertain the goodness of taking an action a for the respective objective being maximized.

To find the Pareto front $\mathcal{P}$ for the analog circuit, we need to find the preference $\mathbf{G}'$ and the action a, which maximize (11). In a discrete action setting, we know that, the optimality filter is given as in Equation (11). However, in the case where we have continuous actions, it makes Equation (11) intractable and we cannot manually search for all these sizing solutions and preferences.

We propose to solve this by using two different policy networks that approximate the optimal Q value update as follows:

$$\mathcal{H}[Q(\mathbf{s}', \mathbf{G})] \approx \underset{Q}{\arg} \mathbf{G}^T \cdot Q[\mathbf{s}, \Omega(\mathbf{s}, \mathbf{G}, \omega), \mu(\mathbf{s}, \mathbf{G}, \phi)]. \tag{12}$$

The two different networks we use for the purpose of multi-objective optimization are the actor policy network $\mu(.)$ and the preference policy network $\Omega(.)$.

5.1 Actor Policy

We propose an actor network that is a parameterized network that approximates the best action to take, given the current sizing solution $\mathbf{s}$ and the designer's preference $\mathbf{G}$. In other words, the actor policy actively tries to push the next query sizing solution $\mathbf{s}'$ in the direction of optimizing the objectives according to the designer's preference $\mathbf{G}$. The actor policy is parameterized by ϕ,

$$\text{Policy}_{actor} = \mu(.|\mathbf{s}, \mathbf{G}, \phi). \tag{13}$$

5.2 Preference Policy

For a given preference level $\mathbf{G}$ specified by the user, the preference policy suggests the most optimal goal-state that needs to be prescribed to maximize the dot product Q network output and the user preference $\mathbf{G}$. Both the preference policy and the actor policy are part of the optimality filter $\mathcal{H}$. The preference policy is parameterized by ω,

$$\text{Policy}_{preference} = \Omega(.|\mathbf{s}, \mathbf{G}, \omega). \tag{14}$$

The preference policy and the actor policy networks suggest the optimal search direction and the incremental change in sizing so that the Q value vector is aligned with the user preference $\mathbf{G}$. Next, we look at how the RL agent is trained.

5.3 Training Phase

The training phase entails the convergence of the critic, actor, and the preference networks in accordance with the Bellman optimality filter. The estimated value that the current critic needs to converge to, for a goal-state–action pair $(\mathbf{s}_G, a)$, is as follows:

$$Q_{\text{est}}(\mathbf{s}, \mathbf{G}, a) = r(\mathbf{s}, a) + \gamma Q[\mathbf{s}', \Omega(\mathbf{s}', \mathbf{G}, \omega), \mu(\mathbf{s}', \mathbf{G}, \phi)]. \tag{15}$$

We define the critic loss as a distance minimization on a distance operator $\mathcal{D}$ between the current Q vector and the estimated Q_{est} vector,

$$\text{Loss}_{critic} = \mathcal{D}[Q(s, \mathbf{G}, a), Q_{\text{est}}(s, \mathbf{G}, a)]. \tag{16}$$

We used $\mathcal{D}$ as a L_2 norm of the Q vector value difference, i.e., $\mathcal{D}(Q, Q_{est}) = ||Q - Q_{est}||_2^2$. The policy to drive the right actions and preferences is trained by maximizing the critic's output for a given preference level $\mathbf{G}$ as shown in Equation (12),

$$\text{Loss}_{policy} = \mathbf{G}^T \cdot Q[\mathbf{s}, \Omega(\mathbf{s}, \mathbf{G}, \omega), \mu(\mathbf{s}, \mathbf{G}, \phi)]. \tag{17}$$

The policy loss can be seen as driving the actor and preference policies to output values that drive the Q function to align with the given preference $\mathbf{G}$. The samples collected through the training process reflect the RL agent's ability to intelligently explore different regions of the design space (space of sizing solutions), while being aware of the tradeoffs in the objectives it tries to optimize for. When new objective values are received through training, we collect them and pass them through a Pareto front identifying filter function, which takes all data points (composed of objective values) $\text{F} = \{f_1, f_2, \ldots, f_m\}$ and outputs a Pareto front. The resulting Pareto Front is the result of training the agent and comprises a certain number of points in it. Note that the set of these Pareto front points is static, meaning that they are a fixed set of points that are identified during the training phase,

$$\mathcal{P} = \text{PF}\Big(\{\text{F}_t\}_{t=0}^T\Big), \tag{18}$$

where T is the number of training steps. We show the training procedure for continuous Pareto optimization for analog circuits in Algorithm 1.

ALGORITHM 1: Pareto Optimization in Analog Circuits Using Reinforcement Learning: Training Phase

```
   Input  : Objectives to optimize for $\{f_1, f_2 \dots f_m\}$
   Output : Pareto Front $\mathcal{P}$; Pareto Set $\mathcal{S}$; Trained Model $\mathcal{M}(\theta, \phi, \omega)$.
 1 Initialize Pareto Front $\mathcal{P} = \emptyset$ and Pareto Set $\mathcal{S} = \emptyset$ or use hybrid approach to initialize.
 2 Initialize Critic $Q_\theta$, Actor $\mu_\phi$, Preference network $\Omega_\omega$
 3 for $k \leftarrow 1$ to $N$ do
 4     Sample $s_0$ randomly or from $\mathcal{S}$
 5     Sample preference $\mathrm{G}_k \in \mathcal{G}$.
 6     for $t \leftarrow 1$ to $L$ do
 7         Select action $a_t \leftarrow \mu(s_t, \mathrm{G}_k, \phi)$;
 8         Observe the new state $s_{t+1} \leftarrow s_t + a_t$;
 9         Collect vector reward $r_t = r(s_t, a_t) \subseteq \mathbb{R}^m$;
10         Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{R}$;
11         Update $\mathcal{P} = \mathrm{PF}(\mathcal{P}, \{\mathrm{r_t}\})$;
12         Update $\mathcal{S}$ with states in $\mathcal{P}$;
13         Sample a batch of $B$ transitions $(s_i, a_i, r_i, s_{i+1})$ from replay buffer $\mathcal{R}$;
14         Assign preference for transition $i$ as $\{\mathrm{G}_z\}_{z=1}^{Z}$;
15         Compute critic loss $\mathrm{Loss}_{critic} = \frac{1}{B} \cdot \frac{1}{Z} \sum_i \sum_z \mathcal{D}[Q(s_i, \mathrm{G}_z, a_i), Q_{\mathrm{est}}(s_i, \mathrm{G}_z, a_i)]$;
16         Compute Policy Loss $\mathrm{Loss}_{policy} = \frac{1}{B} \cdot \frac{1}{Z} \sum_i \sum_z \mathrm{G}_z^{\mathrm{T}} . Q[s_i, \Omega(s_i, \mathrm{G}_z, \omega), \mu(s_i, \mathrm{G}_z, \phi)]$;
17         Update network parameters:
18         Critic:- $\theta : \theta - \nabla_\theta \mathrm{Loss}_{critic}$
19         Actor:- $\phi : \phi - \nabla_\phi \mathrm{Loss}_{policy}$
20         Preference:- $\omega : \omega - \nabla_\omega \mathrm{Loss}_{policy}$
21     end
22 end
23 return $\mathcal{P}, \mathcal{S}, \mathcal{M}(\theta, \phi, \omega)$
```

5.4 Inference Phase

The RL approach to Pareto optimization uses neural networks as function approximators to predict the next action to take given a sizing solution. This way, the trained RL agent has an in-built knowledge of the state transition from one sizing solution to the next, in the circuit simulator. We can thereby exploit the trained agent to query for different user preferences $\mathrm{G_U}$ and augment the points on the Pareto front $\mathcal{P}$ generated during training. This has the benefit of adding additional points on the Pareto front that are not seen during training.

We know that the actor policy network takes into account the designer's preference $\mathbf{G}$ along with the state $\mathbf{s}$, i.e., $\mu(\mathbf{s}, \mathbf{G}, \phi)$. The trick here is to initialize any random state $\mathbf{s}$ along with the designer's particular preference $\mathrm{G_U}$ and let the trained RL agent act according to its actor policy for a few steps. The policy forces the RL agent to take steps in the direction of the preference vector $\mathrm{G_U}$. Thus, this trained model approach can dynamically output design points according to the designer's preference by exploiting the predictive power of the RL agent. We demonstrate this in Algorithm 2.

5.5 Mini-Batch Optimization

To update the critic, actor and the preference losses, we use mini-batch optimization. The mini-batch is sampled from the replay buffer. The batch size is set as B. Also, for each entry in the batch, we randomly sample Z preference directions G_Z. The losses are averaged over the preferences G_Z

ALGORITHM 2: Pareto Optimization in Analog Circuits Using Reinforcement Learning: Inference Phase.

```
   Input  : Pareto Front $\mathcal{P}$ ; Pareto Set $\mathcal{S}$ ; Trained Model $\mathcal{M}(\theta, \phi, \omega)$.
   Output : Updated Pareto Front $\mathcal{P}_{\text{updated}}$
1  Set Designer's custom linear preference $\mathbf{G}_\mathrm{U}$
2  $\mathcal{P}_{\text{updated}} = \mathcal{P}$
3  for $s_k$ in $\mathcal{S}$ do
4      Initialize $s_{k,0}$
5      for $t \leftarrow 1$ to $L$ do
6          Select action $a_t \leftarrow \mu(s_{k,t}, \mathrm{G}_\mathrm{U}, \phi)$;
7          Observe the new state $s_{k,t+1} \leftarrow s_{k,t} + a_{k,t}$;
8          Collect reward $r_{k,t} = r(s_{k,t}, a_{k,t})$;
9          Update $\mathcal{P}_{\text{updated}} = \mathrm{PF}(\mathcal{P}_{\text{updated}}, \{\mathrm{r_{k,t}}\})$;
10     end
11 end
12 return $\mathcal{P}_{\text{updated}}$
```

first, for each entry in the batch and then over all B entries. We show the RL algorithm flow in Figure 1.

6 EXPERIMENTAL RESULTS

Having explained our method in the previous sections, we now look at how we demonstrate the effectiveness of our results. We experiment with three circuits. The circuit simulator used was the Synopsys HSpice Circuit simulator. The circuits were designed under a commercial 90-nm technology. We run our experiments on two different nodes. The circuit simulator was run on an Intel Core i5-6500 CPU with a clock speed of 3.2 GHz. The tensor computations to update the RL agent networks were run on a Nvidia GeForce RTX 3090 GPU to speed up computations with a 24-GB RAM. We primarily concern ourselves with optimizing the gain and the bandwidth. These two metrics are competing in the sense that trying to maximize the gain automatically compromises the maximization of bandwidth and vice versa. These metrics are normalized in the range of [0,1]. The metrics that we use to compare the quality of Pareto fronts are the hypervolume and the fractional contribution, both of which are explained in the next subsection. We used the Python package PyGMO [1] to calculate the hypervolumes.

6.1 Metrics Used

- The *hypervolume* $\mathcal{HV}$, a metric used in works like Reference [9], is the discretized volume between the finite number of points on the approximated Pareto Front and a reference point in m-dimensional space. Since, we are interested in maximizing the objectives, the best Pareto front is the Pareto front for which the hypervolume is minimum. We abbreviate the hypervolume as $\mathcal{HV}$. We provide a brief explanation of why the proposed algorithm helps in decreasing the $\mathcal{HV}$. Consider, for a two-objective problem, the scalarized objective of the form of $\alpha f_1 + (1-\alpha) f_2$. This scalarized objective is maximized when the preference vector $[\alpha, 1-\alpha]$ is in perfect alignment with the multi-objective reward $[f_1, f_2]$. Next, we note that any $[f_1', f_2']$ that aligns with $[\alpha, 1-\alpha]$ maximizes the scalarized objective $\alpha f_1' + (1-\alpha) f_2'$ better if $f_1' > f_1; f_2' > f_2$, i.e., when $[f_1', f_2']$ dominates $[f_1, f_2]$. Since we know that hypervolume calculated with Pareto dominant points will be the least, the scalarized objective approach that we propose still tends to increase the hypervolume by actively searching for Pareto dominant design points.

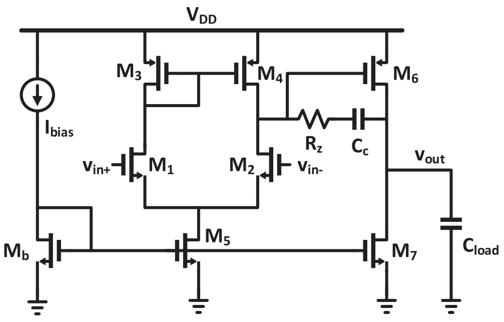
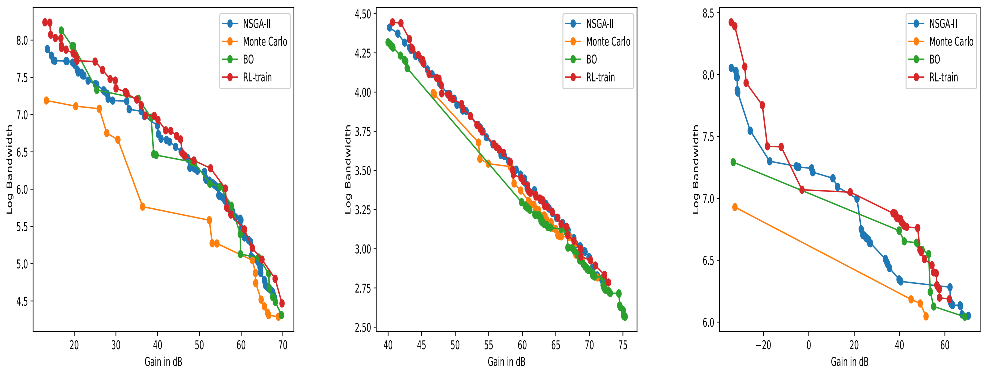


Fig. 3. The two-stage differential amplifier schematic.



(a) Two-stage differential amplifier. (b) Folded-cascode amplifier. (c) Hysteresis comparator.

Fig. 4. The generated Pareto Fronts. The gain and the bandwidth are maximized (they form a competing pair of objectives).

- The *fractional contribution* $\mathcal{FC}$, is a we propose. It gives the fraction of points contributed by the Pareto front $\mathcal{A}$, to the final Pareto front formed by combining the two Pareto fronts $\mathcal{A}$ and $\mathcal{B}$. Thus, we have the relation that $\mathcal{FC}(\mathcal{A}, \mathcal{B}) = 1 - \mathcal{FC}(\mathcal{B}, \mathcal{A})$. If $\mathcal{FC}(\mathcal{A}, \mathcal{B}) > 0.5$, then it means that the Pareto front $\mathcal{A}$ is the major contributor to the resultant front of $\mathcal{A}$ and $\mathcal{B}$. This metric serves to also quantify the quality of the front $\mathcal{A}$ in comparison with $\mathcal{B}$.

Two-stage Differential Amplifier

The two-stage differential amplifier is shown in Figure 3. The circuit consists of a 14-parameter search space, which includes sizing solutions of the circuits along with the resistor and capacitor values.

As can be seen in Figure 4(a), our RL method during training (RL-train), visually generates a Pareto front of a superior quality in comparison with NSGA-II, BO, and Monte Carlo sampling. Table 1, enlists the hypervolume along with the number of samples and the runtime for each algorithm. First, we see that RL-train shows the least hypervolume and thus performs better than all the three reference methods. Next, we see that Monte Carlo and BO take almost 6× the data consumed by RL-train but produce a Pareto front with a larger hypervolume that can also be visually inspected in Figure 4(a). Furthermore, due to the sample efficiency, simulation time is greatly reduced. A drawback of BO is that it has a complexity of $O(N^3)$ and is thus very slow. We believe that our RL-train method performs better than NSGA-II due to the fact that the RL agent

Table 1. Hypervolume Comparison of the Pareto Fronts for the Differential Amplifier

Method	$\mathcal{HV}$	No. of samples	Simulation Time
Monte Carlo	0.83	15,000	~4 hr
NSGA-II	0.67	2,500	~1 hr
BO	0.64	15,000	~10 hr
RL-train	**0.62**	**2,500**	**~1 hr**

Bold text in the tables indicates the best performance metrics across different methods for multi-objective optimization.

Table 2. The Fractional Contribution between All Pairs of Methods for the Differential Amplifier

$\mathcal{FC}$	RL-train	NSGA-II	BO	Monte Carlo
RL-train	—	**0.6**	**0.76**	**1.0**
NSGA-II	0.4	—	0.81	0.98
BO	0.24	0.19	—	1.0
Monte Carlo	0.0	0.02	0.0	—

Bold text in the tables indicates the best performance metrics across different methods for multi-objective optimization.

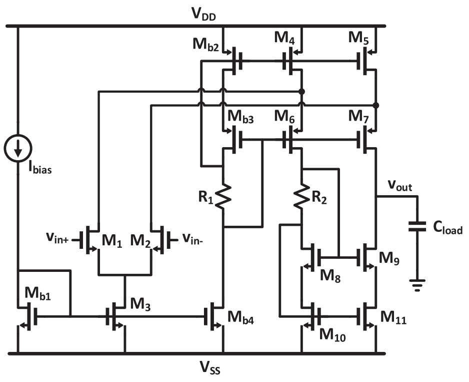


Fig. 5. The folded-cascode amplifier schematic.

understands how sizing affects both the gain and the bandwidth due to the predictive power of the Q network.

In Table 2, the first row demonstrates the fraction of points contributed by the Pareto front $\mathcal{A}$ = RL-train when combined with Pareto fronts $\mathcal{B}$ = NSGA-II, BO, and Monte Carlo. As can be seen RL-train has $\mathcal{FC}$ values greater than 0.5 for all the three comparison methods, which means that RL-train is the dominant contributor to the combination Pareto front.

Folded Cascode Amplifier

We next experiment with the folded cascode amplifier. The amplifier has 18 parameters in its search space. As seen in Table 3 the RL-train has the least hypervolume among all three methods and thus has a better quality Pareto front. It also consumes the least amount of data and has the least simulation time. Table 4 indicates again that the RL-train is a dominant contributor to the combination Pareto fronts.

Table 3. Hypervolume Comparison of the Pareto Fronts for the Folded Cascode Amplifier

Method	$\mathcal{HV}$	No. of samples	Simulation Time
Monte Carlo	0.30	15,000	~4 hr
NSGA-II	0.25	2,500	~1 hr
BO	0.32	15,000	~10 hr
RL-train	**0.26**	**2,500**	**~1 hr**

Bold text in the tables indicates the best performance metrics across different methods for multi-objective optimization.

Table 4. The Fractional Contribution between All Pairs of Methods for the Folded Cascode Amplifier

$\mathcal{FC}$	RL-train	NSGA-II	BO	Monte Carlo
RL-train	—	**0.55**	**0.79**	**0.97**
NSGA-II	0.45	—	0.71	0.94
BO	0.21	0.29	—	0.7
Monte Carlo	0.03	0.06	0.3	—

Bold text in the tables indicates the best performance metrics across different methods for multi-objective optimization.

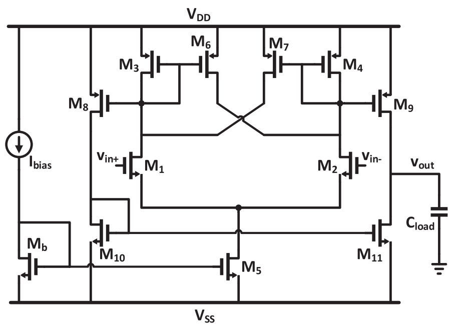


Fig. 6. The hysteresis comparator schematic.

Hysteresis Comparator

The hysteresis comparator has a 12-parameter design space and presents a more complex Pareto front. However, it can be seen that the RL-train method we propose is able to find a better Pareto front. As in the other circuits, we see the best hypervolume and also a best fractional contribution by RL-train.

6.2 Inference Phase

As mentioned before, we can use the predictive power of the trained RL agent to verify if more points can be populated on the Pareto front. To do so, we sweep through the possible values of user preferences $\mathbf{G}_U$ monotonically for some random state initialization near the current Pareto front $\mathcal{P}$. We record the number of time steps required to reach a solution point that dominates points in the current Pareto front $\mathcal{P}$. We do this over N state initializations and note how efficient the search process is, in finding the dominating Pareto front point. We limit the number of steps in each RL run (as in Algorithm 2) in inference to $L = 10$. We define $x_{U,i}$ to be the minimum number

Table 5. Hypervolume Comparison of the Pareto Fronts for the Hysteresis Comparator

Method	$\mathcal{HV}$	No. of samples	Simulation Time
Monte Carlo	1.91	15,000	~4 hr
NSGA-II	1.76	5,000	~2 hr
BO	1.66	15,000	~10 hr
RL-train	**1.60**	**5,000**	**~2 hr**

Bold text in the tables indicates the best performance metrics across different methods for multi-objective optimization.

Table 6. The Fractional Contribution between All Pairs of Methods for the Hysteresis Comparator

$\mathcal{FC}$	RL-train	NSGA-II	BO	Monte Carlo
RL-train	—	**0.61**	**0.9**	**1.0**
NSGA-II	0.39	—	0.83	1.0
BO	0.1	0.17	—	0.91
Monte Carlo	0.0	0.0	0.09	—

Bold text in the tables indicates the best performance metrics across different methods for multi-objective optimization.

Table 7. The Efficiency Comparison of the Trained Models for Different Circuits

Circuit	η
Two-stage Diff Amp.	100%
Folded Cascode Amp.	98%
Hysteresis Comp.	91%

of steps taken to reach a Pareto-dominant point over all sweeps $\mathbf{G}_U$ for a given state initialization i. The new Pareto point prediction capability of the trained agent is high if $x_{U,i} \rightarrow 0$. In short we define the efficiency of the trained agent as

$$\eta = \frac{1}{N} \sum_{i=1}^{N} \left[1 - \frac{x_{U,i}}{L} \right]. \tag{19}$$

From Table 7, all the trained models exhibit good prediction capabilities for the dynamic user preference $\mathbf{G}_U$, which demonstrates that the trained RL agent can be saved to query for Pareto front points not seen during training.

7 CONCLUSION AND DISCUSSIONS

In our work, we propose a RL algorithm for Pareto optimization tailored for analog circuit design. Our method is (1) sample efficient and (2) shows competing performances with standard reference genetic algorithms like NSGA-II and also with BO. (3) Furthermore, the saved RL agent can be used for augmenting the Pareto front obtained during training. Our experimental results illustrate the efficiency of the proposed methods that outperform all the reference methods and suggest a promising direction for Pareto optimization in analog circuits.

By leveraging the capabilities of reinforcement learning in combination with deep neural networks, we anticipate that our approach can effectively adapt to circuits with an expanded set of design parameters. However, our method involves the utilization of preference direction sampling to construct the Pareto front. As the number of objectives increases, covering all potential preference directions may prove to be challenging. We plan to address such questions as part of our future research efforts.

REFERENCES

[1] Francesco Biscani and Dario Izzo. 2020. A parallel global multiobjective framework for optimization: Pagmo. *J. Open Source Softw.* 5, 53 (2020), 2338. https://doi.org/10.21105/joss.02338

[2] Ahmet F. Budak, Prateek Bhansali, Bo Liu, Nan Sun, David Z. Pan, and Chandramouli V. Kashyap. 2021. DNN-Opt: An RL inspired optimization for analog circuit sizing using deep neural networks. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC'21)*. 1219–1224. https://doi.org/10.1109/DAC18074.2021.9586139

[3] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[4] Crina Grosan and Ajith Abraham. 2010. Approximating Pareto frontier using a hybrid line search approach. *Inf. Sci.* 180, 14 (2010), 2674–2695. https://doi.org/10.1016/j.ins.2009.12.018

[5] N. S. Karthik Somayaji, Hanbin Hu, and Peng Li. 2021. Prioritized reinforcement learning for analog circuit optimization with design knowledge. In *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC'21)*. 1231–1236. https://doi.org/10.1109/DAC18074.2021.9586189

[6] Tuotian Liao and Lihong Zhang. 2017. Parasitic-aware GP-based many-objective sizing methodology for analog and RF integrated circuits. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. 475–480. https://doi.org/10.1109/ASPDAC.2017.7858368

[7] Tuotian Liao and Lihong Zhang. 2022. High-dimensional many-objective bayesian optimization for LDE-aware analog IC sizing. *IEEE Trans. VLSI Syst.* 30, 1 (2022), 15–28. https://doi.org/10.1109/TVLSI.2021.3102088

[8] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. 2020. AutoCkt: Deep reinforcement learning of analog circuit designs. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'20)*.

[9] Cătălin Vişan, Octavian Pascu, Marius Stănescu, Elena-Diana Şandru, Cristian Diaconu, Andi Buzo, Georg Pelz, and Horia Cucu. 2022. Automated circuit sizing with multi-objective optimization based on differential evolution and Bayesian inference. *Knowledge-Based Systems* 258 (2022), 109987.

[10] Hanrui Wang, K. Wang, J. Yang, Linxiao Shen, N. Sun, Haeseung Lee, and Song Han. 2020. GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*, 1–6.

[11] Hanrui Wang, Jiacheng Yang, Hae-Seung Lee, and Song Han. 2018. Learning to design circuits. arXiv preprint arXiv:1812.02734 (2018).

[12] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. *Adv. Neural Inf. Process. Syst.* 32 (2019).

[13] Sen Yin, Wenfei Hu, Wenyuan Zhang, Ruitao Wang, Jian Zhang, and Yan Wang. 2022. An efficient kriging-based constrained multi-objective evolutionary algorithm for analog circuit synthesis via self-adaptive incremental learning. In *Proceedings of the 27th Asia and South Pacific Design Automation Conference (ASP-DAC'22)*. 74–79. https://doi.org/10.1109/ASP-DAC52403.2022.9712601

[14] Sen Yin, Ruitao Wang, Jian Zhang, Xiaosen Liu, and Yan Wang. 2023. Fast surrogate-assisted constrained multiobjective optimization for analog circuit sizing via self-adaptive incremental learning. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.* 42, 7 (2023), 2080–2093. https://doi.org/10.1109/TCAD.2022.3221694

[15] Guo Yu and Peng Li. 2007. Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling. 464–469. https://doi.org/10.1109/ICCAD.2007.4397308

Received 30 May 2023; revised 14 November 2023; accepted 28 December 2023