May 2016 UILU-ENG-16-2201

DUPLEX: SIMULTANEOUS PARAMETER-PERFORMANCE EXPLORATION FOR OPTIMIZING ANALOG CIRCUITS

Seyed Nematollah Ahmadyan and Shobha Vasudevan

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801
University of Illinois at Urbana-Champaign

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 2016	

4. TITLE AND SUBTITLE
Duplex: Simultaneous Parameter-Performance Exploration for Optimizing Analog Circuits

5. FUNDING NUMBERS
NSF CCF 14-23431

6. AUTHOR(S)
Seyed Nematollah Ahmadyan and Shobha Vasudevan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1308 W. Main St., Urbana, IL 61801-2307

8. PERFORMING ORGANIZATION REPORT NUMBER
UILU-ENG-16-2201

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Science Foundation
4201 Wilson Boulevard
Arlington, VA 22230

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We present Duplex random tree search, an algorithm to optimize performance metrics of analog and mixed signal circuits. Duplex determines the optimal design, the Pareto set, and the sensitivity of a circuit's performance metrics to its parameters. We demonstrate that Duplex is 5× faster than the state-of-the-art and finds the global optimum for a design whose previously published result was a local optimum. We show our algorithm's scalability by optimizing a system-level post-layout charged-pump PLL circuit.

14. SUBJECT TERMS
1. Design space exploration; 2. Circuit simulation; 3. Circuit synthesis; 4. Optimization; 5. Random tree search

15. NUMBER OF PAGES
9

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Duplex: Simultaneous Parameter-Performance Exploration for Optimizing Analog Circuits

Seyed Nematollah Ahmadyan and Shobha Vasudevan
Coordinated Science Lab, Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign
{ahmadya2, shohbav}@illinois.edu

ABSTRACT

We present Duplex random tree search, an algorithm to optimize performance metrics of analog and mixed signal circuits. Duplex determines the optimal design, the Pareto set and the sensitivity of circuit's performance metrics to its parameters. We demonstrate that Duplex is 5× faster than the state-of-the-art and finds the global optimum for a design whose previously published result was a local optimum. We show our algorithm's scalability by optimizing a system-level post-layout charged-pump PLL circuit.

Keywords

Design space exploration, Circuit simulation, Circuit synthesis, Optimization, Random tree search

1. INTRODUCTION

In the traditional analog/RF IC design flow, designers would manually calculate optimal assignments to a circuit's parameters to ensure that the design meets the performance specification requirements [1][2][3]. In modern designs, analog and mixed signal ICs are ubiquitous due to their desirable flexibility in power, performance, etc. This coupled with shrinking transistor sizes, circuit complexity and new challenges in fabrication processes has made manual calculations infeasible [2][4][5].

Recent pioneering research has developed automatic optimization algorithms for analog design [4][6][7][8][9][7][10][11]. Despite this, some challenges still remain. Firstly, analog/RF circuits tend to have a complex state space with local minima and saddle points. State of the art optimization algorithms [10] can get stuck in local minima, resulting in a non-optimal design. Secondly, quantitatively explaining the decisions made by the optimization algorithm is important for designer interpretability during design optimization. Current optimization algorithms provide no such feedback to the user.

In this work, we propose a new optimization algorithm that addresses these challenges. We introduce **Duplex** random tree search, an algorithm meant for performance optimization of nonlinear analog circuits. Our algorithm does not get stuck in local minima and provides valuable feedback to the user in the form of performance to parameter sensitivity graphs and Pareto distributions. Duplex is also highly performance efficient and scalable, as demonstrated in our results.

The principle of Duplex is different from other known optimization algorithms like simulated annealing [1], gradient descent [12], etc., used in performance optimization. Other algorithms focus their search either on the parameter space (*e.g.* transistor width), or the performance space (metrics such as gain, bandwidth, etc.). In contrast, Duplex simultaneously analyzes both performance and parameter spaces. Global decisions are made in the performance space and actions are taken locally in the parameter space. To the best of our knowledge, this is the first algorithm to simultaneously keep track of a global objective/goal and use it to guide local search steps.

Duplex uses random tree search- a tree based simulation algorithm that also maintains the tree data structure as a record of the state space traversed. It maintains and simultaneously grows two homomorphic (mirrored) random trees- one in the parameter space and the other in the performance space. In the performance space, it uses the basic random tree search to find the globally optimal design by expanding the tree toward the *goal region*. In the parameter space, it decides which parameter needs to change to get closer to the goal region. This decision is made using a reinforcement learning algorithm [13] that evaluates the history of previous changes in the parameter tree based on a reward function. Duplex does not get stuck in local minima because of the probabilistic completeness property of random trees[14]. This is in contrast to random walk based methods like simulated annealing. The guidance in every step from the global search towards the local step decision helps in converging quickly to the optimal goal region.

The choice of random trees contributes to most of the advantages of Duplex. Random trees generally search the space more efficiently than Monte Carlo based simulation methods [14][15][16][17], contributing to Duplex's efficiency. Additionally, during the course of the simulation, a unified tree structure connecting performance and parameter space of the circuit is maintained. This helps generate by-products like Pareto surfaces and sensitivity analysis that provide design insights. Computing the exact Pareto surface is typically computationally very expensive [11][4] since Pareto sets are high dimensional surfaces in the parameter space. Duplex uses a statistical inference algorithm to infer the distribution of optimal states in the tree in the parameter space. It uses the inferred distribution of optimal parameter states as a Pareto surface. In addition, Duplex keeps track of how variations in a given parameter cause performance metrics to change and retrospectively generates a performance to parameter sensitivity graph. This is in contrast to typical algorithms that do not record circuit information during the optimization process.

Duplex is a scalable algorithm and can optimize system-level post-layout circuits. We demonstrate duplex's scalability by optimizing a system-level post-layout 1.6GHz charged-pump PLL circuit [18] (with over 131 CMOS transistors) as shown in Figure 10. Duplex's scalability is due to its open-ended search being restricted to the performance space, that tends to be much smaller than the parameter space, greatly reducing the size of the search space. The size of parameter space depends on the size of the circuit. The PLL circuit has over 100 CMOS transistors, but the performance space has only 5 dimensions (Table 2). The complexity of Duplex is not dependent on circuit size, allowing it to scale easily.

Duplex is computationally highly efficient. We demonstrate that Duplex has an 81% (up to 5×) more speedup as compared to state-of-the-art results [10] on the same design (two stage operational amplifier [10]). Notably, this design has local minima. Although a few branches in the parameter tree grow toward the local maxima, Duplex used the performance tree to grow toward the global optimum and successfully converged toward the global maximum (resulting in an opamp with $5GHz$ bandwidth), unlike [10] which got stuck in a local minima (reporting $2GHz$ as a maximum bandwidth for the circuit) (Figure 6). We also demonstrate Pareto surface computation and sensitivity analysis (Section 4.1). Duplex is a stable algorithm with little variance in its execution with different initial states. We demonstrate Duplex's stability by running it multiple times on a CMOS inverter circuit and optimizing the inverter for power and delay (Figure 7).

Our contributions are as follows. We present Duplex random tree search for performance space optimization of analog circuits that is more efficient than state-of-the-art. Duplex is inherently scalable and does not get stuck in local minima. We provide a simple and efficient technique for Pareto surface generation. We also present a technique to analyze the relative sensitivity of a parameter with respect to performance. With Duplex, we present the idea of optimization by simultaneously traversing dual spaces.

2. PRELIMINARIES

For a given circuit topology and process technology, performance of the circuits is measured with respect to metrics such as gain, slew rate and bandwidth. The circuit's performance depends on parameters such as transistor width, length, resistor and capacitor values. Let $P \in \mathbb{R}^n$ and $Q \in \mathbb{R}^m$ denote the parameter and performance space, respectively. Let n and m denote the number of parameters and performance metrics, respectively. We refer the points in the performance and parameter space as a *performance and parameter states*, respectively.

We model physical constraints of the circuit and manufacturing process as constraints in the parameter space. For example, for the inverter circuit in Figure 1, transistor M_1 could have a width constraint $1\mu m < M_1 \leq 10\mu m$. The parameter space may also have equality constraints enforced by layout design rules. For example, width of transistor M_1 should be twice the width of transistor M_2.

The parameter and performance variables can each have different scales (say Nano to Giga) and measuring units. We normalize across them by mapping every variable to the interval $[0, 1]$. In general, the size of a parameter space n is related to the number of components (size) of the circuit.

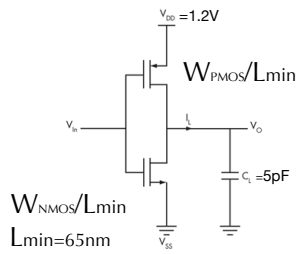


Figure 1: Schematic of an inverter circuit that we use as an illustrative example. We want to optimize the width of NMOS and PMOS transistors to minimize dynamic power and delay.

Constrained parameter space is a subset of the parameter space, bounded by the physical constraints of the circuit. A constrained parameter space is modeled as an intersection of k inequalities:

$$P = \{\mathbf{p} \mid \mathbf{C}\mathbf{p} \leq \mathbf{b}\} \tag{1}$$

where $\mathbf{C}$ and $\mathbf{b}$ are $k \times n$ and $n \times 1$ matrices. For each $\mathbf{p}$ in the set P, all sizing requirements of the circuit are met.

A *performance (parameter) variable* is a variable in the performance (parameter) space. A *performance (parameter) state* is a vector value assignment to all the performance (parameter) variables in the circuit. The relationship between parameter and performance spaces is shown in Figure 2. An instance of the circuit with a given parameter state corresponds to a specific performance state. This can be viewed as an *onto*, or many-to-one mapping f from many parameter states to one performance state. We can only evaluate mapping f point wise using numerical simulation (such as HSPICE).

Reachable performance space is the image Q of the constrained parameter space P in the performance space.

$$Q = \{\mathbf{q} \mid \mathbf{q} = f(\mathbf{p}) \text{ where } \mathbf{p} \in P\} \tag{2}$$

Goal region is a subset of the reachable performance space where the performance of the circuit is within the acceptable range, set by the designer.

$$Q_{\text{goal}} = \{\mathbf{q}_{\text{goal}} \mid \mathbf{G}\mathbf{q}_{\text{goal}} \leq \mathbf{d}\} \tag{3}$$

where $l \times m$ matrix $\mathbf{G}$ defines the constraints on the goal region.

Optimal state is any state in the *goal region* of the performance space.

For the inverter in Figure 1, parameter space is R^2 and consists of the widths of NMOS and PMOS transistors. The optimization goal is to minimize power, rise and fall time delay of the circuit. Specifically, we want $power \leq 100\mu W$ and $delay \leq 10ps$. An example of the parameter state is $(w_{pmos}, w_{nmos}) = (4\mu m, 2\mu m)$. Similarly, a performance state is a vector $(p, d_{rise}, d_{fall}) = (50\mu W, 5ps, 4ps)$.

3. THE DUPLEX RANDOM TREE SEARCH ALGORITHM

We propose *Duplex*, our algorithm for optimizing analog circuits using random trees.

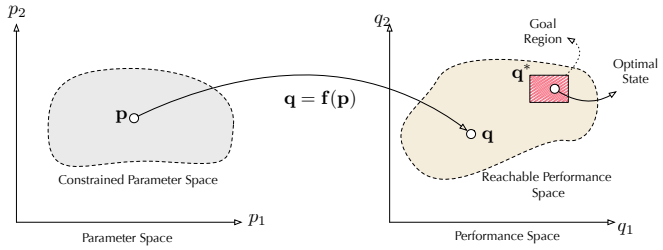


Figure 2: The relation between *constrained parameter space* (left) and the *reachable performance space* and the *goal region* (right).

3.1 Random Tree Search

Random tree is a tree structure [15][14][16] that is constructed in the continuous space $\mathbb{R}^n$. Each node in the tree is a vector in $\mathbb{R}^n$. Each node can have multiple children. The tree is initialized by fixing its root to a specific state in the space. The random tree is constructed incrementally.

Random trees are shown to consistently outperform random walk based search methods such as Monte Carlo simulations for search applications [15][17][16]. Efficiency improvement can be credited to the data structure maintained by the random tree algorithm during the simulation. While growing, it samples a new state in the goal region (desired solution set), and then determines which state is closest (in $\mathcal{L}_2$-norm sense) to that sampled goal state among all of the previously visited states in the tree. It simulates a path between the closest state and the newly sampled state and adds the new state to the tree. This is in contrast to the memory-less sampling of points in the Monte Carlo based methods.

Duplex simultaneously constructs and maintains two different, but mirrored (homomorphic) random trees in the performance and parameter space. Figure 4 shows the parameter and performance random tree growing in the parameter and performance space. Intuitively, the performance tree is the *mirror* of the parameter tree in the performance space. Let T_q denote the performance tree and T_p denote the parameter tree. Let $\mathbf{q}^{(i)}$ and $\mathbf{p}^{(i)}$ denote the i^{th} nodes in the performance and parameter tree, respectively. Let $\mathbf{Q}^*$ denotes the goal region in the performance space.

These trees represent different relationships. An edge in the parameter tree indicates that the two parameter states connected to that edge differ in *exactly one* variable. An edge in the performance tree between indicates that the corresponding states in the parameter tree are connected. For each node $\mathbf{p}$ in the parameter tree, there exists a corresponding node in the performance tree, and vice versa. The corresponding node in the performance tree is computed by simulating the circuit with the given parameters.

For the inverter circuit, each node in the parameter tree is a two-dimensional vector $\mathbf{p} = (w_{\text{nmos}}, w_{\text{pmos}})$, corresponding to the width of NMOS and PMOS transistors. Each node in the performance tree is an assignment of vector of performance metrics $\mathbf{q} = (power, d_{\text{rise}}, d_{\text{fall}})$. Performance node $\mathbf{q}$ is computed by simulating the inverter circuit with the parameter vector $\mathbf{p}$ using HSPICE.

3.2 The Duplex algorithm

Figure 3 shows the flow of the Duplex algorithm. Duplex advances toward the goal region $\mathbf{Q}^*$ in the performance space. In every iteration, it navigates the performance space to get closer to the goal region ($\mathbf{q}_{sample}$). This is the *global search step*. When it finds a close enough state ($\mathbf{q}_{near}$) to the goal region, it looks up the corresponding mirror image of that state in the parameter space ($\mathbf{p}_{near}$). For the mirror state, it finds a neighbor state by perturbing a single parameter in the mirror state ($\mathbf{p}_{new}$). This *local step* in the parameter space is the action the algorithm takes based on the guidance from the performance space. A performance state corresponding to the neighbor state is added in the performance space ($\mathbf{q}_{new}$). The algorithm continues until it reaches an optimal state in the performance space.

3.3 Global search steps in performance space

Duplex biases the search by growing the performance tree toward the goal region. In every iteration, it uniformly samples the goal region to find a candidate optimal state q_{sample}. Duplex's objective in this iteration is to get closer to q_{sample}. It finds the closest state as per Euclidean distance in the performance tree from q_{sample}. It uses the KD-tree algorithm[14] for efficient search of the tree in the performance space. For this q_{near}, it then looks up the corresponding state in the parameter space and finds p_{near}.

3.4 Local coordinated steps in the parameter space

In this phase, Duplex's objective is to find a state p_{new} in the parameter space that is a neighbor of p_{near} and its image q_{new} will be closer to the goal region. It perturbs exactly one parameter in the parameter state p_{near} to obtain a new neighbor state $\mathbf{p}_{\text{new}}$. There are three reasons why duplex only perturbs a single parameter (coordination) at each iteration: Firstly, optimizing a circuit with multiple parameters can be done by iteratively optimizing single parameters in rotation, as in the coordinated descent algorithm [12]. Secondly, for circuit design, explaining the results of the learning algorithm is very valuable. By using coordinated steps instead of the gradient, Duplex is able to use reinforcement learning and compute the sensitivity of performance metrics to parameters. Finally, some parameters might not have significant impact on the performance metrics. By coordinating one parameter at a time, we can find these less significant parameters and i) avoid perturbing them in the future and ii) report them to the designer.

Duplex uses a reinforcement learning algorithm [13] to determine which parameter variable ($p_j \in \mathbf{p}_{\text{new}}$) to perturb. It uses an annealing learning rate [12] to determine how much to perturb the j^{th} parameter in $\mathbf{p}_{\text{new}}$.

3.4.1 Reinforcement learning

Since we treat the circuit as a blackbox, the gradient information is not available. Without the gradient, analytically computing an optimal local step is not possible. Instead, Duplex relies on the history of the previously taken steps to learn what is the best step in future. Duplex keeps a history of how influential each parameter is in getting closer to the goal region. Every parameter state $\mathbf{p}_{near}$ has a reward vector Q associated with it, that is initialized to all ones at the root of the parameter tree. After each iteration, Q is updated, depending on whether changing the j^{th} parame-

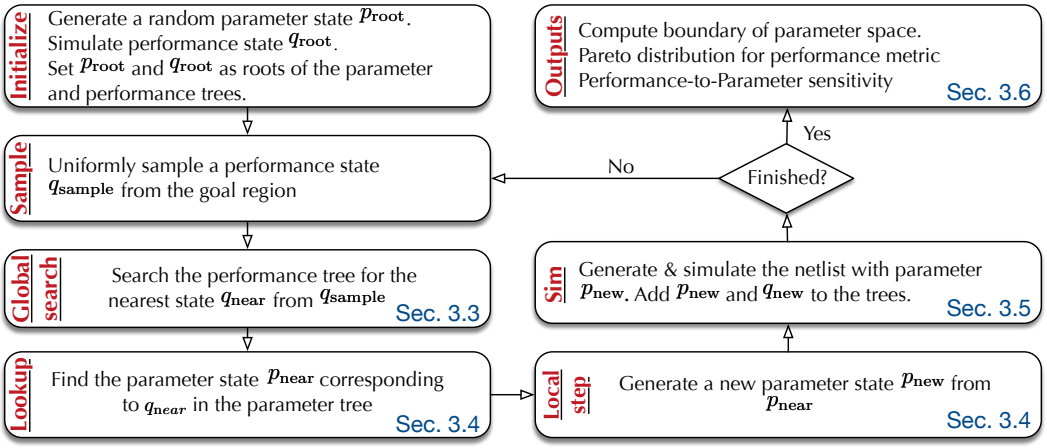


Figure 3: Flowchart of the Duplex random tree search algorithm for performance optimization.

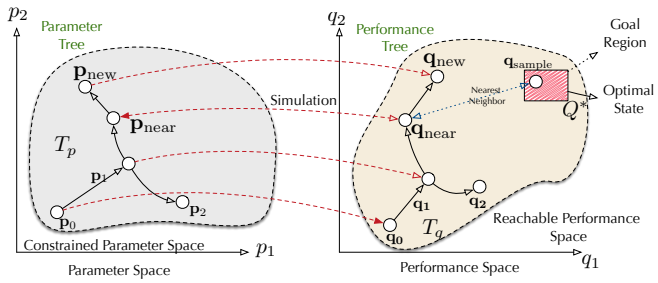


Figure 4: Growing parameter and performance tree in the parameter and performance space.

ter resulted in the corresponding performance state getting closer to the goal region or not.

The new neighbor state $\mathbf{p_{new}}$ differs by the parent parameter state only in parameter j, so we compute the reward vector according to Equation 4.

$$Q(\mathbf{p}_{near}, j) \leftarrow Q(\mathbf{p}_{near}, j) + \gamma(\|q_{new}, Q^*\| - \|q_{near}, Q^*\|) \quad (4)$$

where $\|.\|$ is the distance from the performance state q and the goal region and γ is the discount rate. Next time the parameter state $\mathbf{p}$ is chosen, Duplex uses weighted uniform sampling on $Q(\mathbf{p})$ to select the parameter i.

The reward vector is inherited only by children states of a parent state. Reward vectors on two different paths do not influence each other. This is necessary to avoid making global mistakes in the random tree. Therefore, even if one branch of the tree is stuck in a local minima, the other branches are not affected.

3.4.2 *Annealing learning rate*

The learning rate α in Duplex is set such that initially we search the **space**, then we **converge** toward the optimum state. Duplex determines the length of each step, the extent to which the new parameter state should differ from the parent state, according to a *learning rate* α. Initially the length of the steps are very high (the search phase), but as we get closer to the optimum state, we anneal (gradually lower) the length of each step in order to converge toward the optimum.

The learning rate depends on the step length of the parent state, and a parameter K, and the initial step length α_0 specified by the user.

$$\alpha_{p_{\text{new}}} = \frac{\alpha_0}{1 + K \times \alpha_{p_{\text{near}}}} \quad (5)$$

The sign of the step length is chosen randomly. Duplex adds or subtracts the value of $\alpha_{p_{\text{new}}}$ to the j^{th} parameter in the parameter state vector. In Duplex, unlike other learning algorithms, the learning rate is dependent on the depth of the tree and not the number of iterations. After determining which parameter to change and how much to change that parameter, Duplex generates the new parameter state p_{new}.

3.5 Generating the new performance state

From the neighbor state p_{new}, Duplex generates the new performance state by using a numerical simulator like HSPICE to evaluate the sampled parameter. The values of the performance metrics (gain, bandwidth etc.) form the state vector of q_{new}. The pair (p_{new}, q_{new}) is added to the parameter and performance trees respectively and the reward vector for p_{new} is updated.

3.6 Other outputs: Pareto distribution and sensitivity analysis

After reaching the goal region, the algorithm generates the Pareto surface of the design space. Let S denote the set of performance states in the goal region. Duplex computes the Pareto set by gathering all the corresponding parameter states to the set S. It infers the mixture distribution of the Pareto set using variational Bayesian inference [12].

Duplex also analyzes the *sensitivity* of each performance metric j to each parameter i. It records the result in the sensitivity variable $\mathbf{s}s_{ij}$. Duplex traverses the random tree to determine the number of times a parameter has changed and the extent to which it has changed. In a manner similar to covariance computation, the relative change in the performance due to a parameter is of interest.

Let $f_{j,q}$ denote the value of performance metric j at state q. At each iteration, if changing parameter i results in

change in $f_{j,q}$, we record the difference in variable $\delta_{q_{new},i,j}$:

$$\delta_{q_{new},i,j} = |f_{q_{new},j} - fq_{near}, j| \quad (6)$$

where q_{near} is the parent state of q_{new}. So $\delta_{q,i,j}$ is the difference of performance j between the new state q and it's parent when we change parameter i. Duplex updates the sensitivity matrix according to $\Delta s_{ij} = |\frac{\delta_{q,i,j}}{f_{j,q}}|$.

$$\mathbf{s}_{ij} = \sum_{q} |\frac{\delta_{q,i,j}}{f_{j,q}}| \quad (7)$$

After termination, Duplex normalizes each row (j) in the sensitivity matrix $\mathbf{s}$.

3.7 Termination and complexity analysis

The objective of Duplex algorithm is to reach a goal region. The Duplex algorithm will terminate when it finds sufficient optimal states in the performance tree within the goal region, or if it has reached the maximum number of iterations[1].

Duplex's approach toward search is a twofold: 1) Global search in the performance space, where it searches for the nearest visited state and biases the search toward the goal region, and 2) local search in the parameter space, where it takes the best action from the given parameter state according to the past simulation history. In comparison to the local search, global search typically provides significant efficiency improvement; However it is very expensive and does not scale beyond 100 dimensions. In duplex, we only perform global search in the performance space, which typically is very small (to the order of tens of dimensions). Since the dimension of performance space is usually very small (in comparison to the parameter space) search in the performance space is very efficient. In our implementation, we used KD-tree data structure [14] as our database for closest state search queries. Therefore, the complexity of search for Duplex is $O(n \times m \times \log(n))$ where n is the number of iterations and m is the number of performance metrics.

The Duplex algorithm, unlike conventional search methods such as simulated annealing, does not get stuck in local minima of the performance space. Even if some branches of the random tree do get stuck in local minima, the algorithm simultaneously grows other branches outside the minima and converges toward the global optimum. Therefore, the probability of finding the optimum state goes to 1 as times goes toward infinity. This is based on the probabilistic completeness propertu of the random tree search algorithm [14].

4. EXPERIMENTAL RESULTS

In order to show the effectiveness, efficiency and scalability of Duplex algorithm we used three case-studies: 1) a CMOS inverter, which we used as a proof-of-concept and to analyze the Duplex algorithm, 1) an operational amplifier circuit (from [10]), which we used to demonstrate the efficiency of the algorithm and to show that our algorithm does not get stuck in local minima, and 3) a system-level post-layout charge-pump PLL circuit, which we used to demonstrate Duplex's scalability and practicality for high-dimensional system-level circuit.

[1]Duplex is, in a certain sense, a search algorithm for high-dimensional continuous spaces. Thus, it is technically different from other optimization techniques such as simulated annealing or gradient descent. Unlike optimization methods, Duplex does not try to optimize an objective function after reaching the goal region and meeting the performance requirement of the circuit. Although search algorithms can be used as an optimization algorithms and vice-versa.

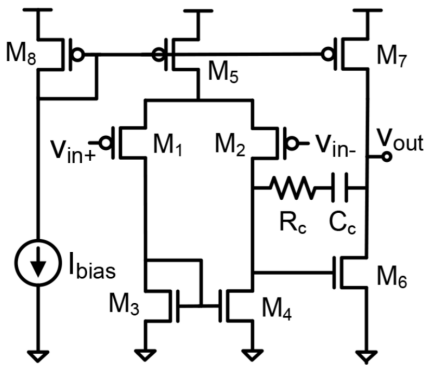


Figure 5: Schematic of a two-stage operational amplifier.

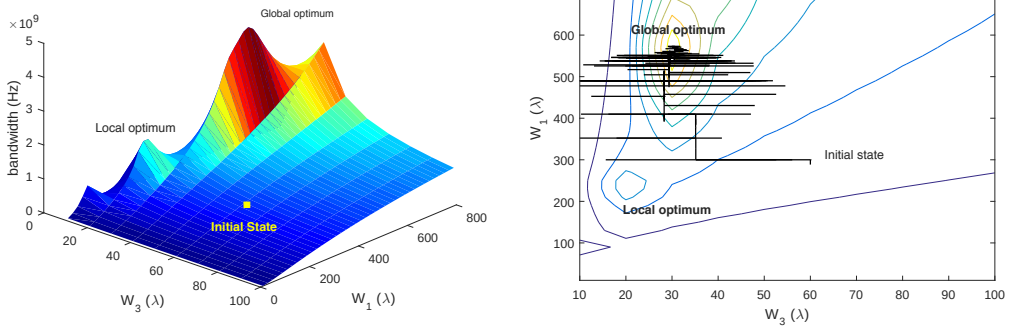
We use the Duplex algorithm to explore the performance space of a two-stage operational amplifier with frequency compensation [10] as shown in Figure 5[2]. The opamp circuit has many parameter and performance variables, demonstrating the scalability and efficiency of the Duplex algorithm. The circuit was designed in 65nm library and the supply voltage was $1.2V$. The main objective is to meet the bandwidth requirement of the circuit. There are 7 design variables in the circuit (the capacitor C_c, the bias current I_{bias}, and the width of transistors W_1, W_3, W_5, W_6 and W_8. The other parameters can be calculated from these parameters. Let $\lambda = 30nm$. The length of all transistors are set to $L_{min} = 10\lambda$ to meet an acceptable output resistance and intrinsic gain.

Table 1: Performance specification for the opamp circuit and the result of circuit optimization. Duplex determines the optimum value for the parameters and performance metrics of the circuit.

Performance metric	Performance Spec. set by. designer	Optimum Value computed by Duplex
Power	$< 0.5mW$	$0.4763mW$
Phase margin	$> 45°$	$109°$
Gain margin	$> 5dB$	$11.22dB$
DC gain	$> 30dB$	$76.59dB$
Slew rate	$> 10\frac{V}{\mu sec}$	$55.65\frac{V}{\mu s}$
Bandwidth	$> 2GHz$	$5.766GHz$

We executed Duplex to optimize the parameters in order to meet the specifications. We designed the circuit in the same process, used the same performance specification and applied the same inputs as [10]. It took approximately 20 minutes and 857 HSPICE simulations to perform the optimization and generate 100 optimal states within the goal region on a Windows machine equipped with a Core-i5 processor and $16GB$ memory to optimize the opamp circuit.

[2]We selected the same opamp case-study as [10] in order to compare Duplex with the state of the art.



(a) The bandwidth w.r.t. the width of transistor M_1 and M_3. The bandwidth objective has one global maximum and multiple local maxima in the state space.

(b) The parameter tree explores the space and converges to the global maxima, while not getting trapped in the local maxima.

Figure 6: Using Duplex for optimizing the bandwidth of the Opamp

The majority of the time was spent on HSPICE simulation and the Duplex's performance overhead was negligible.

[10] reported 4625 SPICE simulations to compute the optimal design. **In comparison, Duplex finished in 857 HSPICE simulations, demonstrating a** 81% **more performance efficiency than [10].** Furthermore, **Duplex improved the quality of the optimization results by increasing the circuit's bandwidth to** $5.7GHz$**, up to** 250%**, in comparison to [10] where they reported the optimized bandwidth of** $2.2GHz$**.** Notable, the opamp design demonstrates how Duplex escapes getting stuck in local minima.

Figure 6 shows how Duplex simultaneously explores the parameter and performance space and avoids the local optima. On the left, Figure 6a shows the circuit's bandwidth w.r.t. size of transistors M_1 and M_3, assuming other parameters are set to optimal value. Let w_1 and w_3 denote the width of transistor M_1 and M_3, respectively. Due to symmetry, size of transistors M_2 and M_4 are equal to M_1 and M_3, respectively. There is one global maximum, located at $(w_1, w_3) = (590\lambda, 30\lambda)$, However there are multiple local maxima throughout the space. The objective of Duplex was to maximize bandwidth without getting stuck in local maxima. We set the initial state at $w_1, w_3 = (300\lambda, 60\lambda)$ and executed Duplex. Figure 6b shows the contour plot of the bandwidth. We rendered the parameter tree over the contour plot to show how Duplex explores the parameter space. **Even though a few branches in the parameter tree grow toward the local maximum at** $(250\lambda, 20\lambda)$**, Duplex used the performance tree to grow toward the global optimum and successfully converged toward the global maximum.**

As the algorithm got closer toward the goal region, the annealing step length caused Duplex to take smaller steps and remains within the goal region. As a result, many of samples were generated within the goal region. At each iteration, Duplex only changed one parameter, making all edges in the parameter tree parallel to the $w_1 - w_3$ axis. Hence, many of the states where the w_1 or w_3 were unchanged are not shown in the projected figure. In order to increase the bandwidth, Duplex aggressively increased the size of transistor M_1. The

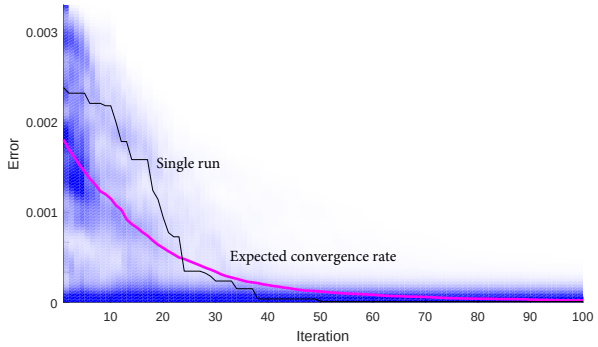


Figure 7: The convergence rate w.r.t. number of iterations for the Duplex algorithm for the inverter case study. Our algorithm converges very fast toward the optimum design from any initial state. Duplex is not sensitive to the choice of initial state.

opamp's unity-gain bandwidth can be approximated as [10] $w_c = \frac{g_{m1}}{C_c}$. This suggests that the bandwidth can be increased by transconductance of the first stage, which in turn can be achieved by sizing up the input transistors M_1 and M_2. Bandwidth can also be increased by reducing the compensation capacitor C_c or increasing the bias current I_{bias}. Duplex automatically performed all of these optimizations in order to meet the specification.

We use the CMOS inverter from Figure 1 to demonstrate a few outputs of Duplex. The inverter is designed in $65nm$ process.

Fig. 7 shows the *visually weighted regression* plot for convergence rate for the Duplex algorithm for the inverter case study. We measure error as the minimum distance from every step in the performance tree toward the center of the goal region. We executed Duplex for 100 independent runs with a random initial state and draw the overlapping convergence plots in the visually weighted regression plot. We also draw the average of all the convergence plot as the expected convergence rate. As shown in the convergence figure, Duplex

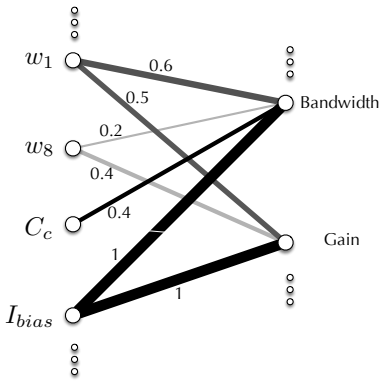


Figure 8: The sensitivity graph visualizing performance to parameter sensitivity for opamp case-study. Edges are annotated with how sensitive is a performance metric (a node in the right side) is to a particular parameter (a node on the left side).

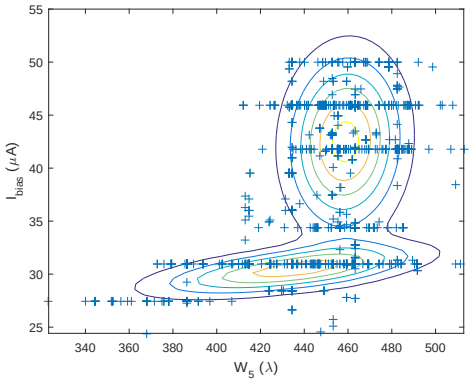


Figure 9: Distribution of the optimal parameters for the opamp circuit. Duplex computes the Pareto set as a mixture Gaussian distribution by inferring the distribution of the samples in the goal region. Pareto surface is computed from the CDF of the pareto distribution. We use the mean of the distribution as the optimum state.

quickly converges toward the goal region in the performance space. Fig. 7 highlights two facts about the Duplex algorithm. 1) Duplex converges exponentially fast toward the goal region and 2) Duplex is very stable with respect to the choice of the initial state

In our experiment, we uniformly sampled the parameters for the initial (root) state in the parameter space, which is the reason for high error variance in the beginning. On the other hand, toward the end of the algorithm the variance in error is low because Duplex converges to the optimum results regardless of the choice of the initial state.

4.1 Performance to Parameter Sensitivity

We visualize the performance to parameter sensitivity matrix s_{ij} using a bipartite sensitivity graph as shown in Figure 8. The left side of the sensitivity graph denotes the parameters of the circuit (such as width of transistors or bias current) and the right side denote the performance metric measured by the Duplex algorithm (such as bandwidth, power and gain). The thickness of each edge between a parameter and performance node denote the sensitivity. Due to the lack of space, we only showed the partial graph of the most important nodes and leave out the rest.

Each row of the sensitivity matrix is normalized. Hence, for each given performance metric, one parameter has an edge of thickness 1.0, denoting the most influential parameter to that performance, and the other parameters have values between $[0, 1]$. For the opamp circuit, the sensitivity graph implies bandwidth depends on the bias current i_{bias}, compensation capacitor C_c and the width of transistors M_1 and M_8. This observation supports our bandwidth analysis earlier. Similarly, the gain is very sensitive to the width of transistor M_1 and sizing of the current mirror M_8, and the bias current. However, the gain is not sensitive to the compensation capacitor. We also observed that the biasing current was the most sensitive parameter in the design.

4.2 Pareto distribution inference

To compute the Pareto distribution, we collect the parameter samples that results in acceptable performance from the circuit. Figure 9 shows the Gaussian mixture distribution of those samples for the opamp circuit, projected to W_5, I_{bias} plane, where W_5 is a width of transistor M_5 and I_{bias} is the bias current. The mean of the Pareto distribution indicates the optimal value of the parameter. Furthermore, we can generate more optimal design parameters from the Pareto distribution and predict the yield for the circuit.

4.3 Optimizing the PLL circuit

The *Charge-Pump PLL (CP-PLL)* [19][18] is one of the key building blocks in many analog IPs and SoCs. The PLL can be used in various applications such as clock synchronization and jitter mitigation.

We used a low-noise 1.6GHz CP-PLL circuit as a system-level example to demonstrate Duplex's scalability. The schematic of the CP-PLL circuit is shown in Figure 10 [19]. The CP-PLL circuit consists of five blocks: Phase detector, a charge-pump circuitry, a loop filter, a voltage controlled oscillator (VCO) and a frequency divider, arranged in a feedback configuration [19]. The circuit has 131 CMOS transistors. The circuit is designed using TSMC $0.18um$ process, using supply voltage $1.8V$. The reference clock was set at 200MHz.

The first block in the CP-PLL circuit was the phase detector circuit. The phase detector compares the clock produced from the VCO with the reference clock and produces an error signal proportional to the phase difference between its inputs. The phase detector block was implemented using a NOR gate, hence it was balanced but not very power efficient. We minimized the total power dissipation of the phase detector while maximizing the gain K_d. We set the width of the transistors in the NOR gate as a parameter.

The charge pump was a set of symmetrical current sources. Transistors $M_{21}, \ldots, M_{26}$ supply the pump-up current to the loop filter. It consists of an input differential pair $M_{21}M_{22}$, current mirror load M_{29}, output current source M_{28}, and pull-up transistors $M_{31}M_{32}$. A similar circuit is used to generate the pump-down current. We set the size of the input differential transistors pairs as a parameter for duplex in order to make the charge-pump circuit fully balanced. After the charge pump block, there was a filter to remove the high-frequency components of the signal introduced by the

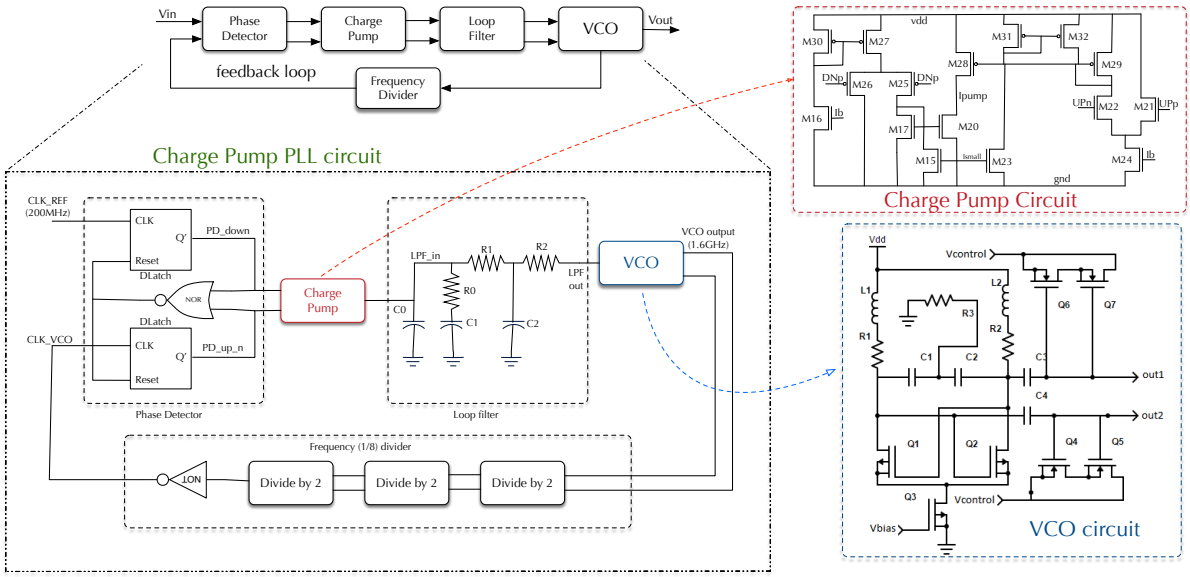


Figure 10: Schematic of a post-layout charge-pump PLL circuit.

phase detector circuit. The loop filter was implemented as a simple RC network and consisted of three resistors and capacitors that created a three-pole one-zero network [18]. The CP-PLL's stability was largely dependent on the value of capacitor C_1, and the bandwidth was dependent on the value of resistor R_1. After duplex optimization, the algorithm determined the optimum value for capacitor C_1 was $48pF$ and for resistor R_1 was $54k\Omega$.

The voltage controlled oscillator (VCO) [18] was supposed to produce a clock at $1.6GHz$. The input stage consists of $M_4, \ldots, M_7$ transistors which are used as varactors for frequency tuning. $M_1, \ldots, M_3$ offers negative conductance to fulfill the oscillating pre-conditions. R_0 and R_1 are parasitic resistance of L_0 and L_1. The output frequency of the VCO depends on the value of m_{mult}. The algorithm was optimizing the bias voltage of the VCO circuit. For the VCO circuit, we ensured the gain of the circuit was more than $25\frac{\text{Meg}}{\text{V}}$. The output of the VCO passes a series of 2:1 dividers, reducing the frequency from expected $1.6GHz$ to $200MHz$[18].

We optimized the PLL design such that it would meet the operating frequency of the PLL at $1.6GHz$ while optimizing the performance of phase detector gain and power and the VCO gain and phase noise. We executed duplex for 350 iteration which took approximately 13 hours. The algorithm found multiple configurations that satisfied the performance requirements of the CP-PLL circuit. Table 2 shows the result of the optimization.

5. RELATED WORK

Analog circuit optimization has been extensively studied in the past [1][2][20][11]. Classic techniques relied on generic optimization techniques (such as simulated annealing) to optimize the circuit's performance [1][2][4][20].

Recently, researchers used computational intelligence techniques to speedup circuit optimization [8][9][7]. [10][11] optimizes the circuit by discretizing and exploring the state space. In [10], the authors optimize an operational amplifier, which we used in Section 4. Other specialized optimization techniques have been employed to address circuit optimization [6] with added objectives, such as yield [4][10] or technology migration [7].

Table 2: Result of Duplex optimization of the CP-PLL circuit. Duplex determines the optimum value for the parameters and performance metrics of the circuit.

Perf. metric	Perf. Spec. set by. designer	Opt. Val. computed by Duplex
PD Gain K_d	$> 1uA/Deg$	$1.16uA/deg$
Frequency	$1.6GHz \pm 0.01$	$1.6025GHz$
VCO PhaseNoise	$< -60dB@60K$	$-96.25dB@60K$
VCO gain	$> 25MEG/V$	$39.1MEG/V$
PD Power	$< 20mW$	$12.85mW$

In summary, we presented a technique for efficient optimization of performance metrics in analog circuits. Our technique is significantly faster than state-of-the-art and gives insights about the circuit like sensitivity analysis. It also simplifies the computation of the Pareto optimal surface significantly. Our tree based optimization algorithm does not get stuck in local minima. The dual performance/parameter space search helps make the algorithm efficient and stay focused.

6. REFERENCES

[1] G. G. E. Gielen, H. Walscharts, and W. M. C. Sansen, "Analog circuit design optimization based on symbolic simulation and simulated annealing," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 3, pp. 707–713, 1990.

[2] G. G. E. Gielen and R. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," in *Proceedings of the IEEE*, pp. 1823–1824, IEEE, 2000.

[3] C. Toumazou and C. A. Makris, "Analog IC design automation. I. Automated circuit generation: new

concepts and methods," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 218–238, 1995.

[4] S. K. Tiwary, P. K. Tiwary, and R. A. Rutenbar, "Generation of yield-aware Pareto surfaces for hierarchical circuit design space exploration," *Design Automation Conference*, pp. 31–36, 2006.

[5] G. Yu and P. Li, "Hierarchical analog/mixed-signal circuit optimization under process variations and tuning," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 2011.

[6] L. C. Severo and A. Girardi, "A methodology for the automatic design of operational amplifiers including yield optimization," *26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–6, 2013.

[7] L. Qian, Z. Bi, D. Zhou, and X. Zeng, "Automated Technology Migration Methodology for Mixed-Signal Circuit Based on Multistart Optimization Framework," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, pp. 2595–2605, Dec. 2014.

[8] B. Liu, F. V. Fernández, and G. G. E. Gielen, "Efficient and Accurate Statistical Analog Yield Optimization and Variation-Aware Circuit Sizing Based on Computational Intelligence Techniques," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, pp. 793–805, June 2011.

[9] H. Lin and P. Li, "Circuit Performance Classification With Active Learning Guided Sampling for Support Vector Machines," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 34, no. 9, pp. 1467–1480, 2015.

[10] S. Jung, J. Lee, and J. Kim, "Variability-aware, discrete optimization for analog circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 8, pp. 1117–1130, 2014.

[11] G. Stehr, H. E. Graeb, and K. J. Antreich, "Analog Performance Space Exploration by Normal-Boundary Intersection and by Fourier–Motzkin Elimination," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 10, pp. 1733–1748, 2007.

[12] C. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[13] R. S. Sutton and A. G. Barto, ***Reinforcement Learning: An Introduction***. Cambridge, MA: MIT Press, 1998.

[14] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[15] S. Ahmadyan and S. Vasudevan, "Automated Transient Input Stimuli Generation for Analog Circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1–1, Oct. 2015.

[16] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, pp. 1–43, Mar. 2012.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[18] J. F. Parker and D. Ray, "A 1.6-GHz CMOS PLL with on-chip loop filter," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 337–343, Mar. 1998.

[19] K. Lim, C.-H. Park, D.-S. Kim, and B. Kim, "A low-noise phase-locked loop design by loop bandwidth optimization," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 6, pp. 807–815, 2000.

[20] X. Li, J. Wang, L. T. Pileggi, T.-S. Chen, and W. Chiang, *Performance-centering optimization for system-level analog design exploration*. IEEE Computer Society, May 2005.