Metamodel-Assisted Ultra-Fast Memetic Optimization of a PLL for WiMax and MMDS Applications

Oleg Garitselov[1], Saraju P. Mohanty[2], Elias Kougianos[3], and Oghenekarho Okobiah[4]
NanoSystems Design Laboratory (http://nsdl.cse.unt.edu)[1,2,3,4]
Department of Computer Science and Engineering [1,2,4] and Department of Engineering Technology [3]
University of North Texas, Denton, TX, USA.[1,2,3,4]
Email-ID: omg0006@unt.edu[1], saraju.mohanty@unt.edu[2], elias.kougianos@unt.edu[3], and oo0032@unt.edu[4]

***Abstract*—With CMOS technologies progressing deeper into the nano-scale domain the design of analog and mixed-signal components is becoming very challenging. The presence of parasitics and the complexity of calculations involved create an enormous challenge for designers to keep their design within specifications when reaching the physical layout stage of the design process. This paper proposes a novel ultra-fast design flow that uses memetic-based optimization algorithms over neural-network based non-polynomial metamodels for design-space exploration. A new heuristic optimization algorithm which is based on memetic algorithms and artificial bee colony optimization is introduced. The design flow relies on a multiple-layer feed-forward neural network metamodel of the nano-CMOS circuit. Using a CMOS PLL circuit it is shown that the proposed design flow is flexible and robust while it achieves optimal design to two different wireless specifications, WiMax and MMDS. Experimental results show that the proposed approach is 2.4× faster than the swarm based optimization over the same metamodels.**

***Index Terms*—Metamodeling, Neural Networks, Memetic, Nano-CMOS, PLL, Modeling, Circuit Optimization**

I. Introduction

Analog and mixed signal design processes usually involve very complex simulations. While the technology progresses forward, the design complexity has drastically increased, mainly due to parasitic interconnect effects. Even though simulation tools exist to help with the design, schematic-level simulation cannot predict the physical layout complexity. Hence,designers usually encounter problems in the post layout stage that affect the design specifications and skew the output of the designed system. The layout stage, which is a time consuming process by itself, usually takes more than one iteration and in the worst case the designer has to thoroughly revise the design. Therefore in many cases the optimization becomes suboptimal with designers trying to fit their design into tight specifications and time to market deadlines.

There have been many attempts to simplify the design stage to optimize the design back to specifications after the initial layout. Circuit simplification, i.e. macromodeling [1] and [2], uses simplified circuits that mirror the circuit's behavior to reduce its simulation time. It is usually conducted in the same simulator tool, i.e. SPICE. This approach is mainly used to reduce simulation time of the design block. It is not parameter sensitive and therefore cannot be used for optimization purposes. Modeling approaches for the circuit as a system (metamodeling) are also common (e.g. regression [3] and neural networks [4]). All these methods use mathematical and/or statistical analysis with tools such as MATLAB to create a formula or set of formulas to predict the output of the system with given parameters. The parameters are usually selected as device sizing, but it can be any other signal of the circuit such as temperature or even an input signal. The formula(s) later can be used to predict the system's behavior while adjusting the design parameters. Metamodeling is widely used in other engineering fields, especially when simulation is either expensive or time consuming.

This research focuses on eliminating the complexities of the long optimization process. The metamodeling approach that is used in this research is based on multilayer neural networks and is shown that the process is very flexible to multiple different design specifications. It is demonstrated on a circuit that was initially built outside the required specs. Two different specifications are used: 2.7 GHz for Multichannel Multipoint Distribution Service (MMDS) and 2.5 GHz WiMax applications and it is shown that the given design flow is flexible and robust.

The rest of the paper is organized in the following manner. Section II discussed the contributions of this paper and prior related research. Section III introduces the proposed design flow. Section IV presents the memetic-based optimization algorithm. Section V discuses the neural networking meta-models. Section VI presents the experimental results. Section VII has the conclusions and future research discussions.

II. Contributions of this Paper and Related Research

The **novel contributions of this paper** are as follows. A neural network metamodel based design optimization flow for analog/mixed-signal circuits is presented. A memetic algorithm is used and is shown to perform considerably better than

[0]This research is supported in part by SRC award P10883 and NSF awards CNS-0854182 and DUE-0942629.

the swarm optimization algorithm that was previously shown to have good results on analog circuits. To the best of the authors' knowledge, this is the first memetic based heuristic algorithm that utilizes the artificial bee colony algorithm at the local layer. The metamodeling approach which is applied to the physical design of a 180 nm CMOS phase lock loop shows considerable speed up. The approach is also proven to be flexible as the same circuit is brought to 2 different specifications.

Memetic algorithm applications are very few in the VLSI domain. In [5], a memetic algorithm is discussed for computing the capacitance coupling in VLSI circuits. In [6], a new sampling-based yield optimization algorithm based on Memetic Ordinal Optimization is presented that improves the yield efficiency of analog circuits. In [7], six different parallel memetic algorithms are investigated for solving the circuit-partitioning problem. In [8], a memetic algorithm is presented for power, delay and area optimization during VLSI partitioning. In [9], a memetic algorithm is presented to solve the floorplanning problem.

These memetic algorithms are used on actual circuits, not metamodels and also not specific to mixed-signal circuits which is the scope of the current paper. Thus, the current work significantly advances the state-of-the-art in analog/mixed-signal circuit optimization.

III. Proposed Ultra-fast Design Flow: Memetic-Based Algorithm and Nonpolynomial Metamodels

The novel ultra-fast design flow that uses a memetic-based optimization algorithm for design-space exploration over neural-network based nonpolynomial metamodels of a nano-CMOS PLL is the focus of this paper. This ultra-fast design flow encompasses three unique aspects: (1) the use of accurate yet fast *nonpolynomial metamodels instead of actual circuit netlists* for optimization, (2) the use of fast optimization algorithms, and (3) the use of minimal number of manual layout iterations to obtain design closure.

The proposed design flow using metamodels is shown in Fig. 1. The physical layout of the baseline design is parameterized and the design space is sampled with two different data sets, once for training samples and another for verification. A neural network is created for each output data set of the design. Computationally expensive optimization algorithms can be applied using metamodels due to the their computational efficiency and the optimized values are then used to adjust the initial physical layout to create the near optimal design. This design flow only uses 2 iterations for physical design, minimizing the amount of time the designer needs to spend on the circuit.

IV. The Proposed Memetic-Based Algorithm for Mixed-Signal Optimization over Metamodels

A. Background on Memetic Algorithms

In 1976, Dawkin [10] introduced the concept of a meme. In 1989, the term Memetic algorithm (MA) was introduced by P. Moscato [11]. The idea proposed a multilevel algorithm that combines evolutionary algorithms for global optimization with more powerful algorithms for the local search. As the genetic algorithm approaches the global optimum, the local optimizer is applied to each offspring before it is inserted into the population. The unique aspect of MA algorithms is that all chromosomes and offspring are allowed to gain some experience, through a local search, before being involved in the evolutionary process [11], [12].

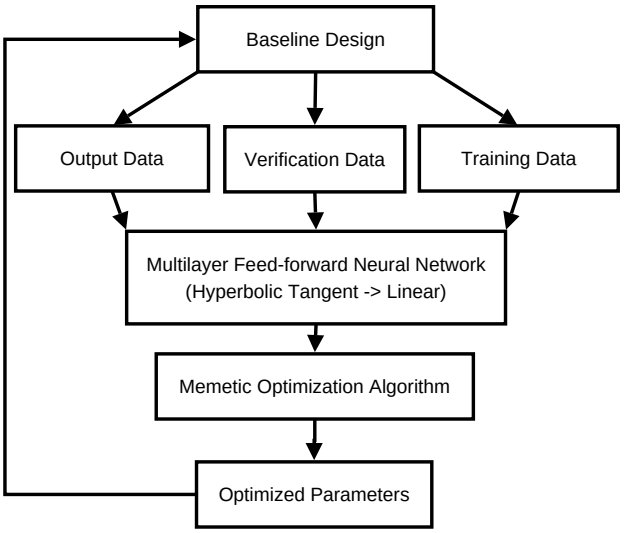


Fig. 1. Abstract of the metamodeling design flow.

A Genetic Algorithm (GA) is a computational model that mimics biological evolution, whereas MA mimics cultural evolution [10]. Several sets of parameters are common to both MA and GA: population size, number of generations, crossover rate, and mutation rate in addition to a local search mechanism. In an MA, a population consists solely of local optimum solutions [13].

B. The Proposed Memetic Algorithm

The proposed algorithm (Alg. 1) is a heuristic memetic optimization algorithm which includes an Artificial Bee Colony optimization (Alg. 2) at the local layer. As memes progress toward the goal on the global layer the local optimization does the fine tuning of the result for memes. Each artificial bee in the ABC algorithm acts as a comeme which is trying to find a better solution within bounds of the local search with the center as the main meme. The weight learning is used to adjust the probability of meme movement control on the global layer with the learning propagating from local to global layer. Since each meme, initially, has equal chance of moving in both directions the weights are used to teach memes to move in the appropriate direction. The weights are adjusted only when the meme reaches closer to the optimum, since only then further direction can be estimated. The mutation of the meme is simulated by applying the random location to the meme instead of using the regular move with the learning weighs being reset.

Algorithm 1 Global Optimizer for the Memetic Algorithm.

Initialize initial population.
Initialize weights.
count = 0.
while count $<$ max_iterations **do**
  count=count+1.
  Evaluate all individual populations.
  for each individual in the population **do**
    Select suitable memes.
    Process with Alg. 2 for local improvements.
    Replace selected meme with improved solution.
    Receive information of the improved location for that meme.
    Adjust meme's weights.
  end for
  Calculate probability for random selection operations = prob.
  if prob is crossover **then**
    Swap selected meme's parameters.
  else if prob is mutation **then**
    Replace selected meme with random solution.
    Reset selected meme's weights.
  end if
end while
Return optimal value and location.

A modified version of the ABC algorithm, which is shown in Alg. 2, was originally used in [14] and was found to be effective on AMS circuits with use of metamodels. It is applied as a local layer for the MA. More information on this algorithm can also be found in the same work. Essentially ABC is the artificial representation of the bee colony behavior as bees try to find the best food source.

V. Multilayer Feed-forward Neural Network based Nonpolynomial Metamodeling

A multiple layer neural network (NN) consists of inputs, a nonlinear activation function in the hidden layer, and a linear activation function in the output layer. This makes multilayer networks very flexible and powerful due to their ability to represent nonlinear as well as linear functions. The multilayer network needs to have at least one nonlinear function otherwise a composition of linear functions becomes just another linear function as follows [15]:

$$\hat{y} = \sum_{j=1}^{d} \beta_j b_j \left(v_j\right) + \beta_0. \tag{1}$$

The linear layer function has the following format:

$$v_i = \sum_{i=1}^{s} w_{ji} x_i + w_{j0}, \tag{2}$$

where w_{ji} is the weight connection between the jth component in the hidden layer and the ith component of the input. The nonlinear tanh activation functions used for the hidden layer

Algorithm 2 Local Layer Optimizer for the MA.

1: **Initialize** maximum iterations = max$_i$.
2: **Receive** initial meme = P and FoM.
3: **Calculate** boundaries for each parameter P(i)= [min,max] $\pm 10\%$ for local optimization.
4: **Define** number of bees NumberBees.
5: **Initialize** the value for how close worker bees will disperse = buffer.
6: **Initialize** bee_matrix(3,NumberBees) = [workers, onlookers, scouts].
7: **Set** bee_matrix first half to be workers and other onlookers.
8: **Initialize** food sources.
9: **while** (counter $\leq$ maxi) **do**
10: **for** i=1 to NumberBees **do**
11: **if** bee is worker **then**
12: **send** worker bee to a random known food source.
13: **Calculate** FoM from neural networks.
14: **if** FoM is better than old **then**
15: **Update** result and location.
16: **else**
17: **Convert** bee to onlooker.
18: **end if**
19: **else if** bee is onlooker **then**
20: **Calculate** probability if the food source is good.
21: **if** probability is high **then**
22: **Convert** bee to scout.
23: **Send** scout to random location around each P, where P=(P.min+random(1) $\times$ P.max) $\times$ buffer.
24: calculate FoM from neural networks.
25: **if** FoM is better than old **then**
26: **Update** result and location.
27: **Convert** bee to worker.
28: **end if**
29: **end if**
30: **else if** bee is scout **then**
31: **Pick** the best result = best_r.
32: **Send** the scout to random location for each P, where P=P.min+random(1)$\times$P.max.
33: **if** FoM is better than old **then**
34: **Update** result.
35: **Convert** bee to worker.
36: **end if**
37: **end if**
38: **if** FoM is better than old **then**
39: **Update** result and location.
40: **end if**
41: **end for**
42: counter = counter + 1.
43: **end while**
44: **Return** result and location.

are sigmoid and were shown to work best for PLL circuit previously [15]:

$$b_j(v_j) = \tanh(\lambda v_j). \tag{3}$$

The network training is performed to minimize the least square criterion:

$$E = \sum_{k=1}^{n} \left(y_k - \hat{y_k}\right)^2. \tag{4}$$

The input data set is generated from SPICE simulations. It is the same for every metamodel and is generated using Latin Hypercube Sampling (LHS). LHS supports any amount

of planes and is proven to work better than Monte Carlo due to the more even distribution of points with still the random factor that helps to detect nonlinearity. LHS divides each plane (parameter) into Latin squares and randomly picks a point from each square. Output is generated for each run from SPICE simulations, saving each needed value to its own data set. Hence, each metamodel will have its own target data set. This work targets NNs that have single output with multiple inputs.

Since the input data set has a large dynamic range, it is desirable either normalize or standardize the input data. If not, the training of higher values can outweigh the lower and the neural network will not train properly. In this work the data is normalized, since in our previous neural networks performed much better than without one.

Since a neural network is created for each desired output there is no need to standardize the output. The output standardization is usually used if there are more than one output and they are in different order, hence affecting the way weights converge during the learning process.

Statistical data is then collected to calculate the Root Mean Square Error(RMSE) and correlation coefficient (R^2) values for both sets.

The created model may fit perfectly to the training data set when the number of sampling points is equal to the number of unknown coefficients, a situation known as "overfitting". For this reason, the verification data set is created so that the points are at different locations than the ones used for fitting. If the verification dataset RMSE and R^2 values do not differ much from the training values, then the model has trained correctly, otherwise it must have been overfitted or trained improperly.

If the neural model did not train correctly, the training parameters of the model can be adjusted or additional sample points can be collected from the circuit simulation. Otherwise the neural network can act as an accurate metamodel for the PLL circuit.

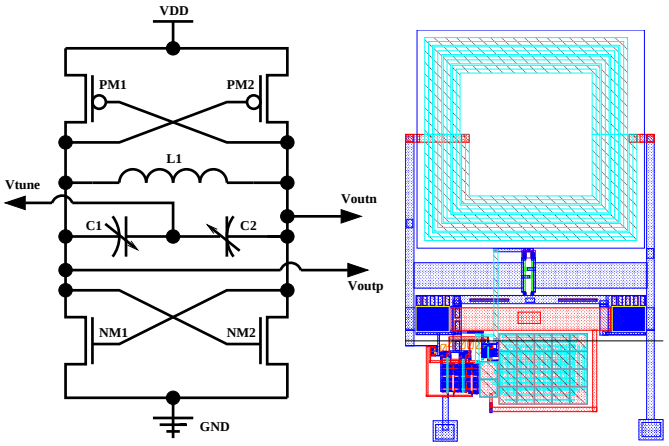
VI. Experimental Results

This section provides two case studies for the same baseline design PLL circuit. The optimization is conducted on the same metamodels that were generated as discussed previously. Two different design specifications for MMDS and WiMax applications are used to show that the initial design can be brought to any specification that is within the metamodel's reach. MA and ABC are compared for speed and accuracy.

A. Design of Test case circuits: WiMax and MMDS PLL

The phase lock loop is a closed feedback loop circuit and is based on an LC-VCO shown in figure 2. The baseline design of this circuit is used from our previous research [15], [14].

Functional simulation of the circuit physical design is shown in Fig. 3 with the PLL locking at 24.58 μs. The baseline phase noise diagram shown in figure 4 shows that the circuit has acceptable phase noise, -163 dBc/Hz at 10 Hz offset. All the simulations are done with use of SPICE. The average power consumption of the PLL circuit is 9.28 mW.



(a) Schematic (b) Layout

Fig. 2. Schematic and 180nm physical design of the LC-VCO.

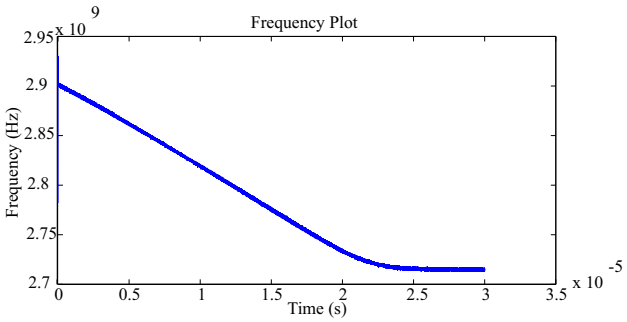


Fig. 3. Frequency plot for initial PLL circuit.

Table I shows the parameters selected for the PLL circuit. For each SPICE simulation, each parameter has been selected according to LHS sampling within its min-max range with a total of 100 simulations for training and 30 simulations for verification datasets.

B. Algorithm Setup

The selected Figure of Merit (FoM) function (Alg. 3) is designed to interface with either optimization algorithm. The FoM of interest is to minimize power as long as the PLL is working within the frequency constraints.

MMDS standards are set in FoM function (Alg. 3) for freq = 2.7 GHz $\pm 0.1\%$ and l_time = 200 ns while WiMax freq = 2.5

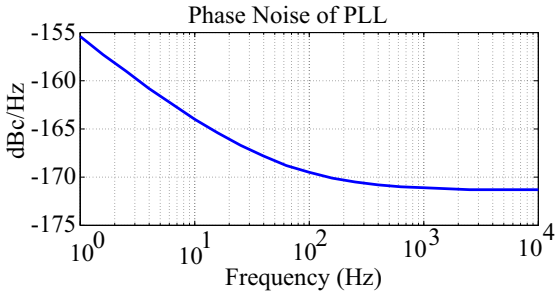


Fig. 4. Phase noise of the baseline PLL circuit.

TABLE I
PARAMETER RANGES AND OPTIMIZATION RESULTS OF PLL COMPONENTS FOR WIMAX AND MMDS SPECIFICATIONS

Circuit	Component	Parameter Name	min	max	MMDS		WiMax	
					Memetic	ABC	Memetic	ABC
Phase Detector	DFF1 PMOS	W_{ppd1}	0.4 μm	2 μm	0.77 μm	1.22 μm	2.0 μm	2.0 μm
	DFF1 NMOS	W_{npd1}	0.4 μm	2 μm	0.4 μm	0.61 μm	1.7 μm	1.71 μm
	DFF2 PMOS	W_{ppd2}	0.4 μm	2 μm	0.4 μm	0.68 μm	1.25 μm	1.25 μm
	DFF2 NMOS	W_{npd2}	0.4 μm	2 μm	0.4 μm	0.63 μm	0.71 μm	0.7 μm
	AND PMOS	W_{ppd3}	0.4 μm	2 μm	0.4 μm	1.64 μm	1.8 μm	1.79 μm
	AND NMOS	W_{npd3}	0.4 μm	2 μm	0.4 μm	1.01 μm	0.56 μm	0.56 μm
Charge Pump	M3, M4	W_{pCP1}	0.4 μm	2 μm	2.0 μm	2.77 μm	1.72 μm	1.7 μm
	M5, M6	W_{nCP1}	0.4 μm	2 μm	0.4 μm	1.67 μm	0.43 μm	0.43 μm
	M1, M2	W_{pCP2}	4 μm	20 μm	1.0 μm	1.6 μm	1.81 μm	1.8 μm
	M7, M8, M9	W_{nCP2}	2 μm	20 μm	0.77 μm	1.09 μm	1.07 μm	1.07 μm
LC-VCO	NM1, NM2	W_{nLC}	3 μm	20 μm	3.0 μm	6.7 μm	19.45 μm	19.45 μm
	PM1, PM2	W_{pLC}	6 μm	40 μm	13.9 μm	18.9 μm	26.6 μm	26.6 μm
Divider	M5	W_{n1Div}	0.4 μm	2 μm	0.4 μm	0.43 μm	0.88 μm	0.86 μm
	M6	W_{n2Div}	0.4 μm	2 μm	0.4 μm	0.74 μm	1.26 μm	1.26 μm
	M7	W_{n3Div}	0.4 μm	2 μm	0.4 μm	0.41 μm	1.72 μm	1.73 μm
	M8	W_{n4Div}	0.4 μm	2 μm	0.4 μm	0.63 μm	0.9 μm	0.92 μm
	M9	W_{n5Div}	0.4 μm	2 μm	0.4 μm	0.7 μm	0.98 μm	0.98 μm
	M1	W_{p1Div}	0.4 μm	2 μm	0.77 μm	1.27 μm	1.85 μm	1.85 μm
	M2	W_{p2Div}	0.4 μm	2 μm	0.4 μm	1.17 μm	1.83 μm	1.80 μm
	M3	W_{p3Div}	0.4 μm	2 μm	0.77 μm	1.83 μm	1.78 μm	1.74 μm
	M4	W_{p4Div}	0.4 μm	2 μm	0.77 μm	1.65 μm	2.0 μm	2.0 μm

Algorithm 3 FoM function for MMDS application

1: **Receive** P coordinates.
2: **Calculate** frequency from neural network with P parameters = freq.
3: **if** freq is within specification **then**
4: **Calculate** locking time from neural network with P parameters = l_time.
5: **if** l_time $<$ specifications **then**
6: **Calculate** power from neural network with P parameters = power.
7: FoM = 1/(power $\times$ 1e3).
8: **else**
9: FoM = 0.
10: **end if**
11: **end if**
12: **Return** FoM.

GHz $\pm$0.1%. It can be seen that the initial design does not fit into these specifications. If the process to optimize this design to these specifications is done manually, adjusting the physical design multiple times, correcting the W/L parameters, etc., it could take days, if not weeks to do so. Instead, the parameters that were chosen to describe the circuit are used to sample the design space using LHS sampling. The output is then recorded and further used to train the neural network with each needed specification for output. Another smaller set of LHS data is carefully created for the same design parameters, but is made sure that they are not the same as the training design set. This set will be used for verification of the neural network to make sure that the design is not overfitted and is accurate.

Both memetic and artificial bee colony algorithms which were introduced in Section IV are used to optimize the circuit to the above specifications. Since both algorithms are constructed to reach an optimal maximum, the figure of merit is transposed to: FoM = 1/(Power $\times$ 1000) since the power units are mW.

C. Comparing Other Algorithms

Figure 5 shows the results for both algorithms running independently to reach power minimization for MMDS constraints. The memetic algorithm reaches a more optimal result faster even though the ABC algorithm shows better results in the beginning. This is due to the ABC algorithm being purely stochastic, which does not enable the algorithm to always converge to the global maximum, even though it has a chance to do so with more iterations. The memetic algorithm starts off a little bit slow due to the learning process, but then reaches the result quite rapidly.

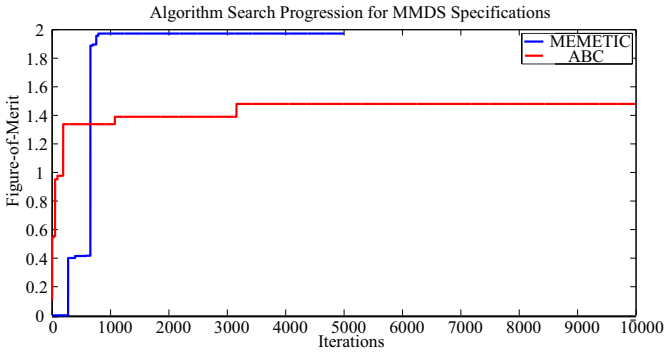


Fig. 5. Comparison of ABC versus MA for MMDS specifications.

Figure 6 shows the results as both algorithms are running independent optimizations trying to maximize the FoM for WiMax constraints. It can be observed that the MA reaches the optimum quite faster than the ABC algorithm. Table I shows

that both algorithms also converge on a very close solution to each other.

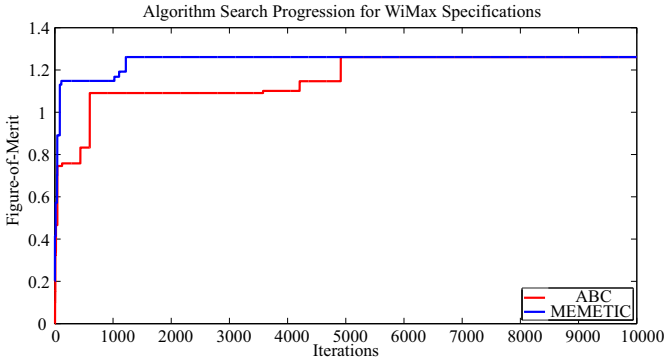


Fig. 6. Comparison of ABC versus MA for WiMax specifications.

Table II shows the final solution outputs for both optimization algorithms. MA has shown the best result for MMDS optimization by converging on a much better solution than the ABC algorithm. The WiMax specifications are very closely matched.

TABLE II
FINAL OPTIMIZATION RESULTS FOR THE PLL.

	MMDS		WiMax	
	Memetic	ABC	Memetic	ABC
Power	0.51 mW	0.68 mW	0.79 mW	0.79 mW
Locking Time	1.9 μs	1.58 μs	1.93 μs	1.92 μs
Frequency	2.702 GHz	2.703 GHz	2.502 GHz	2.502 GHz

As a final comparison, Table III shows the run time and convergence for both algorithms. The speedup was calculated for each in comparison to the ABC algorithm. Even though both algorithms complete the same amount of iterations the speedup comes from the number of accesses to the metamodels, which the ABC algorithm has to complete for every bee. Since the MA does not involve as many calculations at the global stage, this affects the calculation time.

TABLE III
ALGORITHMS COMPARISON FOR THE PLL.

Algorithm	Simulation Time	Convergence (iter.)		Speedup
		WiMax	MMDS	
ABC	$\approx$12 min	4914	3193	1$\times$
Memetic	$\approx$5 min	1221	825	2.4$\times$

VII. CONCLUSION AND FUTURE RESEARCH

This paper presented a metamodeling design flow using neural networks. It was shown, by the example of a 180 nm PLL, that the presented design approach was able to bring a circuit that was not within specification to two different design specifications. A new heuristic memetic algorithm was presented and was shown to work considerably better than a swarm based optimization algorithm (ABC). The MA has converged on the near-optimal result 2.4$\times$ faster than the ABC algorithm, while finding the same or better solution. The metamodel creation process, which involves sampling and SPICE simulations at a total of 130 points, took approximately 12.5 hours, while the optimization process only takes 5 minutes. This would be a considerable speedup over running the optimization on the SPICE netlist itself, or readjusting the layout manually.

REFERENCES

[1] G. Wolfe and R. Vemuri, "Extraction and Use of Neural Network Models in Automated Synthesis of Operational Amplifiers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, vol. 22, no. 2, pp. 198–212, February 2003.

[2] A. Agarwal, G. Wolfe, and R. Vemuri, "Accuracy Driven Performance macromodeling of Feasible Regions during Synthesis of Analog Circuits," in *Proc. Great Lakes Symp. VLSI, GLSVLSI*, 2005, pp. 482–487.

[3] C.-Y. Chao and L. Milor, "Performance Modeling of Analog Circuits Using Additive Regression Splines," in *Proceedings of the IEEE 1994 Custom Integrated Circuits Conference, 1994.*, May 1994, pp. 301–304.

[4] Y. Zhao, J. Gao, and X. Yang, "A Survey of Neural Network Ensembles," in *International Conference on Neural Networks and Brain*, vol. 1, October 2005, pp. 438–442.

[5] Y. Bontzios, M. Dimopoulos, and A. Hatzopoulos, "A Memetic Algorithm for Computing 3D Capacitance in Multiconductor VLSI Circuits," in *Proceedings of the IEEE 14th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, 2011, pp. 341–346.

[6] B. Liu, F. Fernandez, and G. Gielen, "An Accurate and Efficient Yield Optimization Method for Analog Circuits Based on Computing Budget Allocation and Memetic Search Technique," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition*, 2010, pp. 1106–1111.

[7] E. Armstrong, G. Grewal, S. Areibi, and G. Darlington, "An investigation of parallel memetic algorithms for vlsi circuit partitioning on multicore computers," in *Proceedings of the 23rd Canadian Conference on Electrical and Computer Engineering*, 2010, pp. 1–6.

[8] P. Subbaraj, K. Sivasundari, and P. Siva Kumar, "An Effective Memetic Algorithm for VLSI Partitioning Problem," in *Proceedings of the IET-UK International Conference on Information and Communication Technology in Electrical Sciences*, 2007, pp. 667–670.

[9] M. Tang and X. Yao, "A Memetic Algorithm for VLSI Floorplanning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 37, no. 1, pp. 62–69, feb. 2007.

[10] R. Dawkin, *The Selfish Gene*. Oxford, U.K.: Oxford Univ. Press, 1976.

[11] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Toward Memetic Algorithms," Technical Report Caltech Concurrent Computation Program. Pasadena:California Institute of Technology, 1989.

[12] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison Among Five Evolutionary-based Optimization Algorithms," *Advanced Engineering Informatics*, vol. 19, pp. 43–53, January 2005.

[13] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-W. Wong, "Classification of Adaptive Memetic Algorithms: A Comparative Study," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 1, pp. 141–152, February 2006.

[14] O. Garitselov, S. P. Mohanty, E. Kougianos, and P. Patra, "Bee Colony Inspired Metamodeling Based Fast Optimization of a Nano-CMOS PLL," in *Proceedings of the 2nd IEEE International Symposium on Electronic System Design (ISED)*, 2011.

[15] O. Garitselov, S. P. Mohanty, and E. Kougianos, "Fast-Accurate Non-Polynomial Metamodeling for nano-CMOS PLL Design Optimization," in *Proceedings of the 25th IEEE International Conference on VLSI Design*, 2012.