Techniques for CAD Tool Parameter Auto-tuning in Physical Synthesis: A Survey (Invited Paper)

Hao Geng, Tinghuan Chen, Qi Sun, Bei Yu
Department of Computer Science and Engineering
The Chinese University of Hong Kong, NT, Hong Kong, China
{hgeng,thchen,qsun,byu}@cse.cuhk.edu.hk

Abstract

As the technology node of integrated circuits rapidly goes beyond 5nm, synthesis-centric modern very large-scale integration (VLSI) design flow is facing ever-increasing design complexity and suffering the pressure of time-to-market. During the past decades, synthesis tools have become progressively sophisticated and offer countless tunable parameters that can significantly influence design quality. Nevertheless, owing to the time-consuming tool evaluation plus a limitation to one possible parameter combination per synthesis run, manually searching for optimal configurations of numerous parameters proves to be elusive. What's worse, tiny perturbations to these parameters can result in very large variations in the Quality-of-Results (QoR). Therefore, automatic tool parameter tuning to reduce human cost and tool evaluation cost is in demand. Machine-learning techniques provide chances to enable the auto-tuning process of tool parameters. In this paper, we will survey the recent pace of progress on advanced parameter auto-tuning flows of physical synthesis tools. We sincerely expect this survey can enlighten the future development of parameter auto-tuning methodologies.

I Introduction

In the synthesis-centric modern VLSI design flow, IC designers first specify the functionality of a circuit and then leverage the synthesis-centric flow to accomplish the circuit through a chain of steps coarsely including logic synthesis, physical synthesis, and mask synthesis. Quite a few steps in the flow need to search for the solution to large-scale NP-hard problems. Under advanced technology nodes, high-quality design entails designers' substantial attempts, huge costs on time and other resources. To combat such design complexities and be flexible to meet various types of design requirements, computer-aided design (CAD) tools incorporated with an ocean of sophisticated algorithms have been developed to aid and speed the closure of an integrated circuit (IC) design. However, it is a double-edged sword. Numerous tunable parameters with different data types are exposed as hints for human beings and impact the QoR of the tool outcome. The huge tool parameter space coupled with multiple design objectives makes the exhaustively manual tuning virtually impossible for novice or even experienced engineers. Even worse, synthesis runs may take several hours or even days. Consequently, both academia and the industry have started to place great emphasis on tool parameter auto-tuning.

Design space exploration techniques that aim at navigating for optimal solutions (values, designs, or configurations) in large design space have been heavily harnessed in our community [1–9]. To a certain extent, the parameter tuning problem which can be treated as parameter space searching is analogous to design space exploration (DSE) [1]. Consequently, many optimization frameworks developed under the DSE problem formulation can be employed as reference.

Typically, the tool parameter auto-tuning problem relates to optimizing multiple QoR metrics concurrently. Distressingly, these QoR metrics (e.g. delay, power, and area) are conventionally in conflict or coupled. The trade-offs are inevitably taken into account. Besides, modeling the whole tuning process in an explicit mathematical expression seems to be arduous or even impossible. To put it from another angle, it belongs to "black-box" optimizations. As we know, the objective functions have explicit formulations in "white-box" optimization problems, which can be directly addressed by numerous numerical methods. The contrary is the "black-box" optimization case. The surrogate function is proposed as an alternative to be optimized in the "black-box" problem.

Over the past decade, a large amount of pioneered works [10–17] concentrate on parameter auto-tuning techniques in the CAD field. For example, a cutting-edge tool auto-tuner [11] based on the multi-armed bandit (MAB) is used in FPGA and high-level synthesis (HLS) compilations [14, 16]. However, these techniques highly depend on heuristics and lack generality. With more heuristics deployed to meet capacity and turnaround time requirements, tools themselves including such tuning process become unpredictable, especially when driven to their limits [18]. Machine learning (including deep learning and reinforcement learning) techniques which have gained great success in other domains such as computer vision and natural language processing pervade our community [19–31].

Powered by machine learning techniques, quite a few parameter auto-tuning works [10, 32–38] behave excellent performance.

As is observed in previous arts, CAD tool parameter auto-tuning has the potential to be widely exploited in modern synthesis-centric design flow including three synthesis stages: high-level synthesis (HLS), logic synthesis, and physical synthesis. HLS optimizes register transfer level (RTL) implementations with parameters such as loop manipulation, function inlining, memory configuration and etc. Under constraints such as clock period, logic synthesis optimizes the QoR metrics like area and power of a netlist via restructuring Boolean logic and mapping it to standard cells. Physical synthesis flow, which covers partitioning, floorplanning, placement, clock tree synthesis, and routing through CAD tools, determines the layout performance. It plays an essential role in advanced circuit design. It is worth pointing out that nowadays, a physical synthesis tool has over ten thousand command-option combinations. The status quo is that such complicated tools which are difficult to fathom lead to unpredictable outcomes. A gap emerges between physical synthesis tools and IC designers' methodologies. The auto-tuning of physical synthesis tools which feed predictions and guidance into the physical synthesis flow without human intervention is imperative. In this survey, we will focus on tool parameter auto-tuning in physical synthesis.

The rest of the paper is organized as follows. Section II will give an overview of the parameter auto-tuning of the physical synthesis tools. Section III and Section IV will introduce the heuristic-based works and machine learning method-based frameworks, respectively. Section V will conclude the whole paper and discuss possible future directions as well.

II Problem Descriptions

For a better understanding, we draw a visualization of a parameter auto-tuning of physical synthesis tool in Fig. 1. Given the gate-level netlist of a design with other files like timing constraints, technology libraries, the parameter auto-tuning of a physical synthesis tool is expected to automatically find out the optimal parameter configurations so that the post-P&R performance of a design can achieve or even beyond the expectation. During tuning, the auto-tuning module (i.e., tuning engine in Fig. 1) will obtain some latent information from the physical design tool's output (e.g., tool log files and reports containing QoR metrics values) to guide the tuning process. The resource like runtime cost by the auto-tuning process should be minimized or within the rational budget. Besides, the QoR metrics to be optimized may be one (e.g., timing) or multiple (e.g., area vs. power vs. timing), which relates to the customer's preference. To be more clear, we also list some common challenges existing in parameter auto-tuning of physical design tools. 1) A vast number of tool

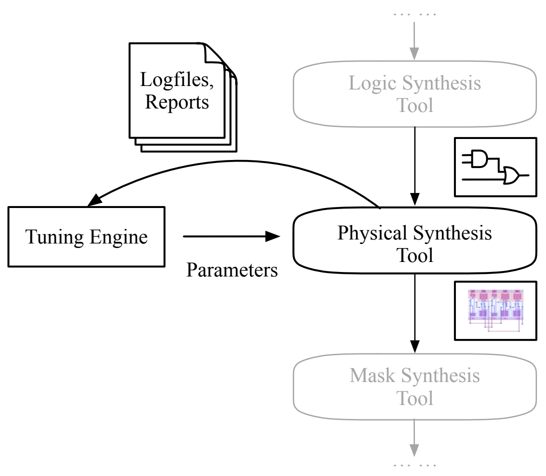


Fig. 1.: The parameter auto-tuning of a physical synthesis tool.

parameters may need to be tuned. 2) Multiple QoR metrics (e.g., area, power, and delay) to be optimized may be in conflict. 3) "Black-box" parameter-to-performance mappings make researchers hard to write down the explicit function expressions. 4) The tool evaluation is time-consuming.

III Heuristic Method-based Physical Synthesis Tool Parameter Auto-tuning

Some prior arts avail heuristic methods to predict tuning of tool executions. Such methods can provide some quick and relatively cheap feedback as well as good suggestions on parameter configuration to IC engineers. Nevertheless, it requires much experience, domain knowledge, and even some tricks to apply the heuristics viably. Besides, heuristic-based frameworks may be stuck in some sub-optimal parameter configurations so that the post-P&R performance of a design is affected.

SynTunSys (STS) in [13, 39] is the first self-evolving and autonomous system for tuning the input parameters of logic and physical synthesis tools. Ziegler *et al.* further enhance the STS tuning framework in [13] by integrating an adaptive online learning method and developing the STS Scheduler (STSS) to tune multiple macros. The main architecture of STS is shown in Fig. 2. It is composed of one main tuning loop and one background archiving loop. In the main loop, the framework embraces running multiple synthesis jobs in parallel, monitoring the jobs in flight, analyzing the costs of the jobs, and a decision engine for the next jobs per iteration. The archiving loop is used to record the results of all runs from all macros, users, and projects.

To reduce the huge searching space, the authors combine several specific valued synthesis parameters which target a singular action and then create a new boolean variable called the primitive (see Fig. 3). With creating

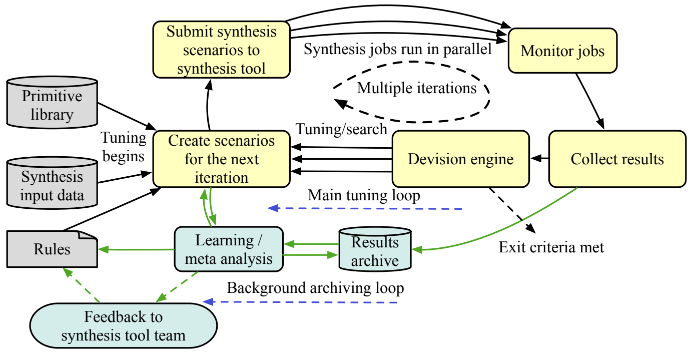


Fig. 2.: The framework of the SynTunSys [39].

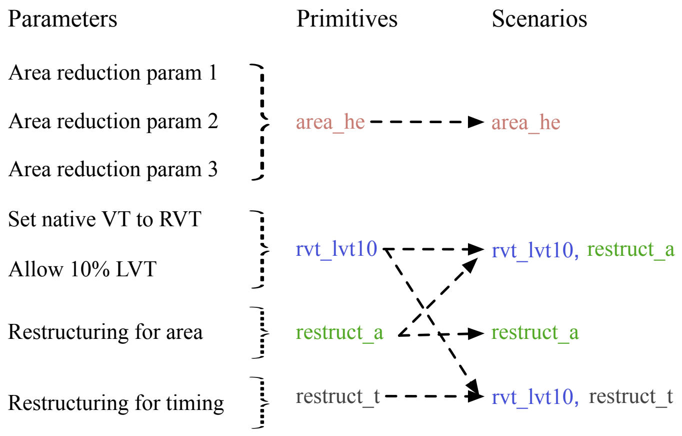


Fig. 3.: The illustration of the interaction of parameters, primitives, and scenarios in the SynTunSys [39].

scenarios consisting of one or more primitives, the searching space is further reduced. Two kinds of decision diagrams are developed in STS, as shown in Fig. 4. One is a base algorithm, namely, a sensitivity test followed by iterative combinations of scenarios. Another enhanced version is to choose a certain number of scenarios in each iteration as parallel synthesis jobs and dynamically adapt to these scenarios that can return lower costs with higher probability. It is worth mentioning that the cost of STS is a linear summation of multiple weighted QoR metrics. The experimental results of applying STS to the IBM z13 22nm high-performance server processor demonstrate a 36% average improvement in total negative slack and a 7% power reduction.

IV Machine Learning Method-based Physical Synthesis Tool Parameter Auto-tuning

In this section, we survey the recent progress on the machine learning method-based parameter auto-tuning frameworks. This kind of method uses either a one-time effort training method or iterative optimization scheme (e.g., Bayesian optimization, active learning), where machine learning models act as a regressor or surrogate model of the associated flow.

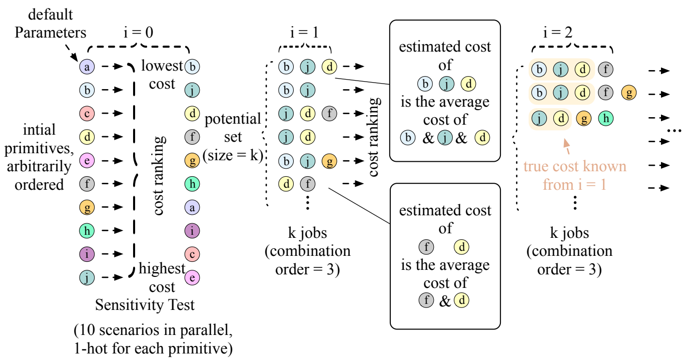

default Parameters
i = 0
lowest cost
intial primitives, arbitrarily ordered
cost ranking
potential set (size = k)
highest cost
Sensitivity Test
(10 scenarios in parallel, 1-hot for each primitive)
i = 1
cost ranking
k jobs (combination order = 3)
estimated cost of b j d is the average cost of b & j & d
estimated cost of f d is the average cost of f & d
i = 2
true cost known from i = 1
k jobs (combination order = 3)

Fig. 4.: The learning decision algorithm of the SynTunSys [39].

A One-time Effort Training Framework

In a one-time effort training framework, the parameter tuning issue is cast into a classical machine learning problem. During training, certain machine learning models are calibrated by training data (e.g., feature representations of tool parameters with golden QoR metric values). Then the performance is tested on the testing dataset like unseen netlists.

[36] is inspired by the solution to matrix factorization issue in collaborative recommender systems, and spans the problem to the high-dimension space via using tensor decomposition and an artificial neural network. It considers parameter auto-tuning of CAD tools for logic synthesis and physical design (LSPD). The whole recommender system-based tuning framework is displayed in Fig. 5. In the off-line training stage, a neural network model for collaborative filtering is trained via QoR results in the archive to make QoR metric predictions. After the model is calibrated, the recommendation for the target macro with a pre-defined QoR cost function is performed in an online fashion. This one-time effort framework exploits the single-layer perceptron network. In light of the over-parameterized regime, neural networks have the advantage of high accuracy but require more data than non-parameterized models like Gaussian process.

Agnesina *et al.* [37] optimize the placement parameters via a deep reinforcement learning framework (see Fig. 6) fed with a mixture of handcrafted features covering graph topology theory along and graph embeddings extracted from the netlist via graph neural networks (GNNs) [40]. In [37], an RL agent is constructed to tune the parameters of the placement tool automatically, aiming at minimizing wirelength. The agent learns from interacting with the placement tool over a number of discrete time steps. At each time step, the agent receives a state and chooses an action from a set of possible actions regarding its policy mapping states to actions. In return, the agent receives a reward and transitions to the next state. This process continues until the agent meets a terminal state after which the process restarts. A state vector in [37]

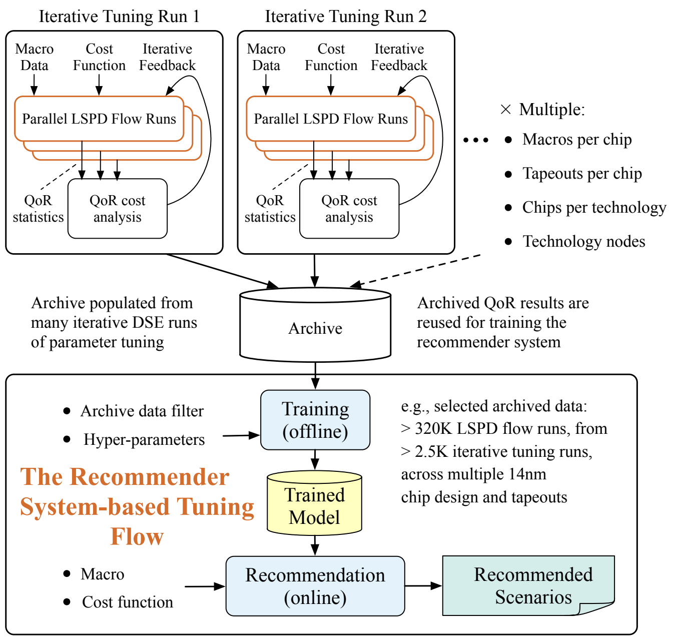


Fig. 5.: The sketch of the framework proposed in [36].

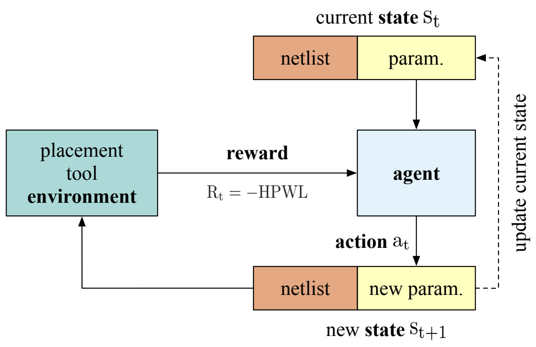


Fig. 6.: Deep reinforcement learning-based parameter tuning of a placement tool [37].

consists of manually designed topological graph features (e.g., average degree, maximum clique and so on), features from netlist extracting by Graph Neural Network, and tool parameters. The experimental results on OpenCores, ISPD2012 contest, and two RISC-V single cores benchmarks demonstrate the method generalizes well to unseen netlists. But the defect also exists. The deep reinforcement learning framework (i.e., A2C) utilized is hard to train and reproduce, and it is hungry for oceans of training data.

B Iterative Refinement Framework

The iterative optimization scheme often has several key features: initialization, modeling, sampling. During initialization, a machine learning model has been initialized by a small amount of data. In modeling, it captures domain knowledge about the regularity of the design space by using the learning models like XGBoost [41], Gaus-

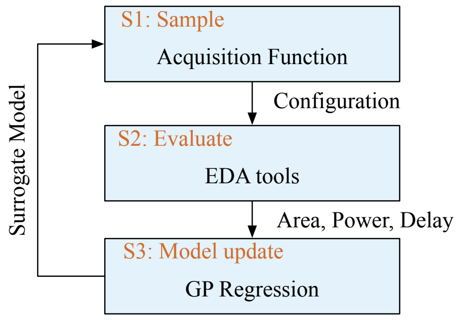
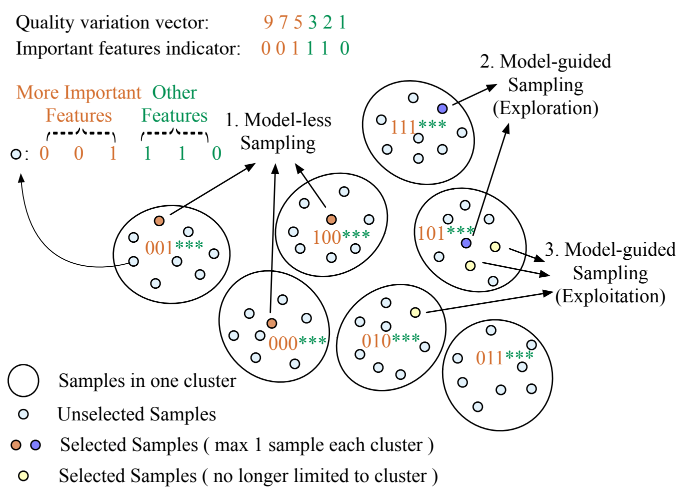


Fig. 7.: The diagram of the workflow proposed in [34].

sian process (GP) models. The learning model will be used to predict QoR metric values for parameter configurations that have not been evaluated by the synthesis tool. The sampling step is performed to select points for the fine-tuning of learning models. During sampling, based on the predictions (sometimes with predictive uncertainties), some Pareto-optimal driven frameworks [4,7] do additional work, namely, classification on the inputs. They iteratively discard points that are either redundant or suboptimal, and it terminates when no more points can be removed. With high probability, the remaining points define an optimal or Pareto set of the given parameter space.

A state-of-the-art Bayesian optimization technique with weighted and summed cost function is exploited to handle the multiple objective problems in the CAD tool (the Synopsys IC Complier tool) parameter auto-tuning [34]. Fig. 7 illustrates the corresponding flow of the applied Bayesian optimization algorithm. In each iteration, it performs a 3–step process and provides one sample point. It starts with optimizing the acquisition function, which returns a set of parameter configurations. The acquisition function is designed to guide the selection of the next most potential candidate. In this framework, authors try several famous acquisition functions like the lower confidence bound (LCB), the probability of improvement (POI), and expected improvement (EI). For the multiple QoR metrics to be optimized, they are scaled and then summed as a single one utilized in the acquisition function (e.g., Gaussian process upper confidence bound). The experimental results of 64-bit adder case show that their proposed Bayesian optimization framework has significant superiority to the genetic algorithm as well as industrial settings. Despite the fact that the single-objective Bayesian optimization framework is modified by the weighted and summed cost function, the defect that the weight-sum trick is not fully appropriate for handling the multi-objective problem still exists.

The prior art [38] exploits feature importance to guide the sampling shown in Fig. 8 and utilize an ensemble boosting tree-based regressor, XGBoost, as the learning model. In this work, the concept of "important" features is that these features can influence the final QoR metric



Quality variation vector: 9 7 5 3 2 1
Important features indicator: 0 0 1 1 1 0
More Important Features
Other Features
o: 0 0 1 1 1 0
1. Model-less Sampling
2. Model-guided Sampling (Exploration)
3. Model-guided Sampling (Exploitation)
001***
100***
111***
101***
000***
010***
011***
Samples in one cluster
Unselected Samples
Selected Samples (max 1 sample each cluster)
Selected Samples (no longer limited to cluster)

Fig. 8.: An example of sampling by clustering [38].

quality. When evaluating the QoR metric quality variation caused by changing the value of one feature, values of the rest are the same. The larger the variation, the more important the feature. Based on the rules, then samples with the same level for important features are grouped. To navigate the whole parameter space within acceptable cost, a model-less sampling scheme that randomly selects one sample from a subset of clusters is performed. These samples act as the initial dataset.

When it turns to the model-guided sampling in this iterative refinement framework, the balancing between "exploration" and "exploitation" becomes crucial. Here, "Exploration" means only obtaining new samples from unvisited clusters, while "exploitation" also considers sampling from promising explored clusters. At the beginning of the model refinement phase, the framework focuses on exploration. The reason is that the number of potential optimal parameter configurations is relatively small and many clusters have not been explored. After several iterations, the focus will be shifted to exploit previously visited clusters. The XGBoost model is only trained by the potential optimal parameter configuration. Thereby, a good trade-off between exploitation and exploration is achieved.

The whole framework is built upon the active learning strategy for iterative refinement. In addition, the weight-sum trick is also employed in this method. Although this kind of method is classical and commonly used, yet limitations emerge.

As is observed, the trick of weighted-sum cost function is utilized in many works [13,34,38,39] to handle multiple QoR metrics. Especially, most of them directly design a linear summation form to combine QoR metrics with trade-off coefficients. Nevertheless, the linear weighted-sum technique transforms the optimization problem into a single objective, which is not totally equivalent to the original multi-objective problem since the extra weighting coefficients could be arbitrary [42]. The final solutions still depend on these coefficients which cannot be easily and optimally chosen. Even worse, it works only for convex problem formulations, but synthesis-derived Pareto sets can be non-convex [1], and may contribute to their performance degradation. Pareto-optimal driven approaches [4, 7] and multi-objective Bayesian optimization [43] may be the potential solution to multi-objective optimization in parameter auto-tuning issue.

V Conclusion

In this paper, we have surveyed the recent line of arts in techniques of physical synthesis tool parameter auto-tuning. Heuristics and machine learning techniques have been applied in these works. Specially, one-time effort training framework and iterative refinement optimization are two kinds of auto-tuning schemes to utilize machine learning models.

In the future, we believe the following aspects are worth studying. 1) Parameter space auto-pruning which significantly reduces tuning runtime overhead is substantial. 2) In iterative refinement frameworks like Bayesian optimization, the Gaussian process is exploited. But it has limited fitting ability for the very complicated model and is computationally costly due to cubical scaling concerning the number of data-points. On the other hand, neural networks as a powerful regressor cannot be directly integrated into the Bayesian optimization framework due to a lack of predictive uncertainty. One potential solution to this dilemma is using neural processes [44] which have lower model complexity and provide estimation uncertainty. 3) Transferring auto-tuning frameworks between different technology nodes and similar designs would be beneficial to industries. 4) One recent work [45] integrates several CAD tools into a unified and standardized system where machine learning models can give feedback to the design flow to guide parameter auto-tuning of tools. It is convincing that collectively considering parameter auto-tuning of multiple tools exploited in the whole design flow has great prospects.

References

[1] H.-Y. Liu, I. Diakonikolas, M. Petracca, and L. Carloni, "Supervised design space exploration by compositional approximation of Pareto sets," in *Proc. DAC*, 2011, pp. 399–404.

[2] S. Roy, Y. Ma, J. Miao, and B. Yu, "A learning bridge from architectural synthesis to physical design for exploring power efficient high-performance adders," in *Proc. ISLPED*, 2017, pp. 1–6.

[3] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *Proc. ICML*, 2018, pp. 3306–3314.

[4] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A Pareto driven machine learning approach," *IEEE TCAD*, vol. 38, no. 12, pp. 2298–2311, 2019.

[5] Q. Sun, C. Bai, H. Geng, and B. Yu, "Deep neural network hardware deployment optimization via advanced active learning," in *Proc. DATE*, 2021, pp. 1510–1515.

[6] Q. Sun, T. Chen, S. Liu, J. Miao, J. Chen, H. Yu, and B. Yu, "Correlated multi-objective multi-fidelity optimization for HLS directives design," in *Proc. DATE*, 2021, pp. 46–51.

[7] H. Geng, Y. Ma, Q. Xu, J. Miao, S. Roy, and B. Yu, "High-speed adder design space exploration via graph neural processes," *IEEE TCAD*, 2021.

[8] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework," in *Proc. ICCAD*, 2021, pp. 1–9.

[9] Q. Sun, C. Bai, T. Chen, H. Geng, X. Zhang, Y. Bai, and B. Yu, "Fast and efficient DNN deployment via deep Gaussian transfer learning," in *Proc. ICCV*, 2021, pp. 5380–5390.

[10] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proc. DAC*, 2013, pp. 1–7.

[11] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "OpenTuner: An extensible framework for program autotuning," in *Proc. PACT*, 2014, pp. 303–316.

[12] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, and L. P. Carloni, "A synthesis-parameter tuning system for autonomous design-space exploration," in *Proc. DATE.* IEEE, 2016, pp. 1148–1151.

[13] M. M. Ziegler, H.-Y. Liu, and L. P. Carloni, "Scalable autotuning of synthesis parameters for optimizing high-performance processors," in *Proc. ISLPED*, 2016, pp. 180–185.

[14] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A parallel bandit-based approach for autotuning FPGA compilation," in *Proc. FPGA*, 2017, pp. 157–166.

[15] A. B. Kahng, S. Kumar, and T. Shah, "A no-human-in-the-loop methodology toward optimal utilization of EDA tools and flows," *DAC work in progress poster*, 2018.

[16] C. H. Yu, P. Wei, M. Grossman, P. Zhang, V. Sarker, and J. Cong, "S2FA: An accelerator automation framework for heterogeneous computing in datacenters," in *Proc. DAC*, 2018, pp. 1–6.

[17] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, J. Kwon, and L. P. Carloni, "SynTunSys: A synthesis parameter autotuning system for optimizing high-performance processors," in *Machine Learning in VLSI Computer-Aided Design.* Springer, 2019, pp. 539–570.

[18] A. B. Kahng, "Reducing time and effort in ic implementation: a roadmap of challenges and solutions," in *Proc. DAC*, 2018, pp. 1–6.

[19] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," in *Proc. DAC*, 2017, pp. 62:1–62:6.

[20] H. Yang, S. Li, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," in *Proc. DAC*, 2018, pp. 131:1–131:6.

[21] C. Zhuo, K. Unda, Y. Shi, and W.-K. Shih, "From layout to system: Early stage power delivery and architecture co-exploration," *IEEE TCAD*, vol. 38, no. 7, pp. 1291–1304, 2018.

[22] Y. Cao, T. Shen, L. Zhang, X. Yin, and C. Zhuo, "An efficient and flexible learning framework for dynamic power and thermal co-management," in *Proc. MLCAD*, 2020, pp. 117–122.

[23] T. Chen, B. Lin, H. Geng, S. Hu, and B. Yu, "Leveraging spatial correlation for sensor drift calibration in smart building," *IEEE TCAD*, 2020.

[24] H. Geng, W. Zhong, H. Yang, Y. Ma, J. Mitra, and B. Yu, "SRAF insertion via supervised dictionary learning," *IEEE TCAD*, vol. 39, no. 10, pp. 2849–2859, 2020.

[25] W. Zhong, S. Hu, Y. Ma, H. Yang, X. Ma, and B. Yu, "Deep learning-driven simultaneous layout decomposition and mask optimization," *IEEE TCAD*, 2021.

[26] W. Li, Y. Qu, G. Chen, Y. Ma, and B. Yu, "TreeNet: Deep point cloud embedding for routing tree construction," in *Proc. ASPDAC*, 2021, pp. 164–169.

[27] H. Geng, H. Yang, L. Zhang, J. Miao, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," *IEEE TCAD*, 2021.

[28] T. Chen, Q. Sun, C. Zhan, C. Liu, H. Yu, and B. Yu, "Deep H-GCN: Fast analog IC aging-induced degradation estimation," *IEEE TCAD*, 2021.

[29] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. ICCAD*, 2021, pp. 1–9.

[30] R. Chen, S. Hu, Z. Chen, S. Zhu, B. Yu, P. Li, C. Chen, Y. Huang, and J. Hao, "A unified framework for layout pattern analysis with deep causal estimation," in *Proc. ICCAD*, 2021, pp. 1–9.

[31] H. Geng, F. Yang, X. Zeng, and B. Yu, "When wafer failure pattern classification meets few-shot learning and self-supervised learning," in *Proc. ICCAD*, 2021, pp. 1–8.

[32] N. Kapre, H. Ng, K. Teo, and J. Naude, "InTime: A machine learning approach for efficient selection of FPGA CAD tool parameters," in *Proc. FPGA*, 2015, pp. 23–26.

[33] Q. Yanghua, H. Ng, and N. Kapre, "Boosting convergence of timing closure using feature selection in a learning-driven approach," in *Proc. FPL.* IEEE, 2016, pp. 1–9.

[34] Y. Ma, Z. Yu, and B. Yu, "CAD tool design space exploration via Bayesian optimization," in *Proc. MLCAD*, 2019, pp. 1–6.

[35] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "LAMDA: Learning-assisted multi-stage autotuning for FPGA design closure," in *Proc. FCCM*, 2019, pp. 74–77.

[36] J. Kwon, M. M. Ziegler, and L. P. Carloni, "A learning-based recommender system for autotuning design flows of industrial high-performance processors," in *Proc. DAC*, 2019, pp. 1–6.

[37] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *Proc. ICCAD*, 2020, pp. 1–9.

[38] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *Proc. ASP-DAC*, 2020, pp. 19–25.

[39] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, and L. P. Carloni, "A synthesis-parameter tuning system for autonomous design-space exploration," in *Proc. DATE*, 2016, pp. 1148–1151.

[40] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE TNNLS*, vol. 32, no. 1, pp. 4–24, 2020.

[41] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. KDD*, 2016, pp. 785–794.

[42] X.-S. Yang, *Nature-Inspired Optimization Algorithms.* NLD: Elsevier Science Publishers B. V., 2014.

[43] D. Hernández-Lobato, J. Hernández-Lobato, A. Shah, and R. Adams, "Predictive entropy search for multi-objective Bayesian optimization," in *Proc. ICML*, 2016, pp. 1492–1501.

[44] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh, "Neural processes," in *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[45] S. Hashemi, C. Ho, A. Kahng, H. Liu, and S. Reda, "METRICS 2.0: A machine-learning based optimization system for IC design," in *Workshop on Open-Source EDA Technology*, 2018, p. 21.