Reinforcement Learning for Electronic Design Automation: Case Studies and Perspectives

(Invited Paper)

Ahmet F. Budak*†, Zixuan Jiang*†, Keren Zhu†, Azalia Mirhoseini‡, Anna Goldie‡, and David Z. Pan†
† The University of Texas at Austin
‡ Google
{*ahmetfarukbudak, zixuan, keren.zhu*}*@utexas.edu*, {*azalia, agoldie*}*@google.com*, *dpan@ece.utexas.edu*

2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC) | 978-1-6654-2135-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASP-DAC52403.2022.9712578

***Abstract*—Reinforcement learning (RL) algorithms have recently seen rapid advancement and adoption in the field of electronic design automation (EDA) in both academia and industry. In this paper, we first give an overview of RL and its applications in EDA. In particular, we discuss three case studies: chip macro placement, analog transistor sizing, and logic synthesis. In collaboration with Google Brain, we develop a hybrid RL and analytical mixed-size placer and achieve better results with less training time on public and proprietary benchmarks. Working with Intel, we develop an RL-inspired optimizer for analog circuit sizing, combining the strengths of deep neural networks and reinforcement learning to achieve state-of-the-art black-box optimization results. We also apply RL to the popular logic synthesis framework ABC and obtain promising results. Through these case studies, we discuss the advantages, disadvantages, opportunities, and challenges of RL in EDA.**

I. Introduction

Reinforcement learning (RL) is a field of machine learning that studies agents interacting with exterior environments. In general, RL is an experience-driven paradigm of maximizing the cumulative reward by taking actions based on the observation of the environment. It has shown huge success and even demonstrated superhuman performance in well-defined environments, such as playing video games [1], chess, and Go [2]. Further, RL algorithms are applied in many real world applications, where environments are much more complicated.

In the past decade, RL algorithms have benefited from the representation extracted by deep learning. The deep reinforcement learning algorithms have proven to be effective in high-dimensional data with discrete or continuous action space with the help of deep learning.

Many optimization problems in EDA share the same nature. Therefore, RL algorithms have recently seen rapid advancement and adoption in electronic design automation (EDA) from both academia and industry [3], [4], [5]. In academia, we have seen RL algorithms used to tackle many sub-problems in EDA, including placement, sizing, logic synthesis, routing, etc. In industry, we notice that companies in EDA have devoted much effort in the direction of general artificial intelligence (AI). For instance, Synopsys and Cadence have developed and promoted their AI platforms DSO.ai [1] and Cerebrus [2].

This paper provides an overview of the RL algorithms and their applications in EDA, with the paper organization shown in Figure 1. We begin with an introduction to reinforcement learning algorithms. Building on that foundation, we analyze why RL has been gaining momentum in the field of EDA and discuss several applications. We demonstrate why and how RL algorithms are used in EDA with three case studies in chip placement, transistor sizing, and logic synthesis. In these three problems, we convert a difficult optimization problem into a control problem in the Markov decision process, which can be effectively solved by reinforcement learning algorithms. Specifically, RL agents make decisions on the decision variables step by step. The learning agent will evolve during the learning process with the assistance of other machine learning methods. We conclude this paper with discussions on the advantages, disadvantages, opportunities, and challenges of RL in the field of design automation.

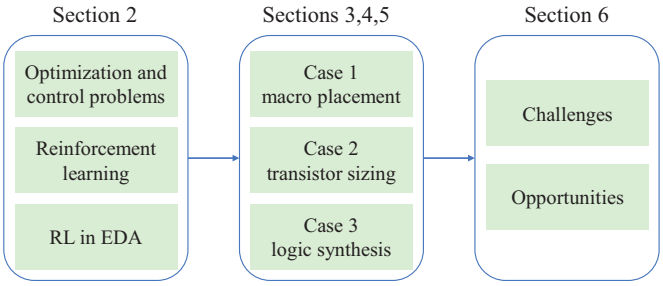

Fig. 1. The paper organization.

II. Reinforcement Learning in EDA

In this section, we give an overview of reinforcement learning algorithms and their applications in EDA.

A. Control problem and reinforcement learning

Reinforcement learning algorithms are usually proposed to tackle the control problem in Markov decision process (MDP). A Markov decision process is a 4-tuple (S, A, P, R), whose elements are defined as below.

- S and A are state and action spaces, respectively.
- P is the state transition function, $P(s, s', a) = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$ is the probability that action a in state s will lead to state s' at the next timestamp.
- R is the reward function. Specifically, $R(s, s', a)$ is the immediate reward obtained after moving from state s to state s' with action a.

At the t-th step of a Markov decision process, we observe the state S_t and reward R_t from the environment and decide

*The first two authors share equal contributions.
[1]Link to Synopsys DSO.ai
[2]Link to Cadence Cerebrus Intelligent Chip Explorer

978-1-6654-2135-5/22/$31.00 ©2022 IEEE

the action $A_t = \pi(s_t)$ based on the policy $\pi(s_t)$. We repeat the process of observation and decision by interacting with the environment. Our objective is to find the optimal policy function π^* to maximize the expectation of accumulated reward formulated as follows,

$$\mathbb{E}[\sum_{t=0}^{T} \gamma^t R_t]$$

where $\gamma \in [0, 1]$ is the discounting factor and T is the total number of steps of one episode. $T = \infty$ means that the MDP continues without termination.

Reinforcement learning algorithms can solve this control problem by learning from repeated trial and error. There are two major branches in reinforcement learning algorithms: value based and policy based methods. In value based methods, we attempt to model and calculate the value of each state $s \in S$ or each pair of state and action $(s, a) \in (S, A)$. In policy based methods, we parameterize the policy function π_θ and optimize the parameters θ. We refer the readers to [6] for more details.

In the past decade, machine learning, particularly deep learning, has achieved great success. Reinforcement learning has also benefited from the development of deep representation learning. Specifically, we can use deep neural networks to approximate the state space, the action space, and the policy function.

B. Optimization problem and reinforcement learning

Most of the problems in the field of electronic design automation can be formulated as follows:

$$\min_{x \in \mathcal{X}} f(x)$$

where f is the objective function, $\mathcal{X} \subseteq \mathbb{R}^n$ is the set of feasible solutions. Several types of optimization problems can be efficiently solved with existing algorithms. For example, linear programs where f is linear and $\mathcal{X}$ is a convex polytope can be solved with the simplex method or the interior point method.

However, many problems cannot be tackled efficiently with existing methods. Below, we list several reasons that these problems are challenging.

- The feasible solution set $\mathcal{X}$ is discrete. For example, some variables x_i should be integers.
- The objective function f is not convex. We may be stuck in a local minimum.
- In black box and derivative-free optimization [7], only zero-order information of the objective function f is accessible.

To address these challenges, the feasible solution set $\mathcal{X}$ is usually explored with heuristic methods, such as grid search, simulated annealing, or evolutionary algorithms.

Reinforcement learning has been leveraged to tackle these difficult optimization problems [8]. In a typical routine, an optimization problem can be converted into a control problem in MDP, such that the problem can be solved with reinforcement learning algorithms. The naive conversion is illustrated in Figure 2. In each step of the MDP, we make a decision on one element of the variable x_i. After n steps, we obtain one potential solution $\hat{x} \in \mathbb{R}^n$, which will be used to query the constraint and objective functions. Specifically, the reward in the first $(n-1)$ steps is 0. The reward of the final step is shown below:

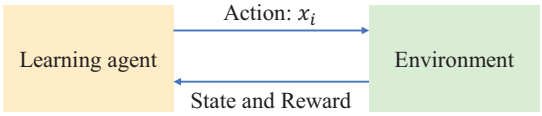

Fig. 2. Converting an optimization problem into a control problem in MDP. In each step, the agent determines one element of the variable x_i and interacts with the environment. The environment provides the state (or its approximation) and reward. The reward is 0 except the final reward as shown in Equation (1).

$$R_n = \begin{cases} -f(\hat{x}), & \text{if } \hat{x} \in \mathcal{X} \\ -\infty, & \text{otherwise} \end{cases} \tag{1}$$

If the RL algorithm finds the maximum accumulated reward, we have solved the equivalent optimization problem. In this routine, the state is all the decisions we have ever made so far. When making decision on x_{t+1}, we observe the state, which is the historical decisions $S_t = \{x_1, x_2, ..., x_t\}$, or its representation $f(S_t)$. The action space is the feasible set for the decision variable.

C. RL in EDA

There are many control problems and optimization problems that cannot be solved efficiently with existing algorithms in the field of electronic design automation. Human engineers in the design flow usually try to obtain acceptable solutions via trial and error. Specifically, designers have to try different settings many times to achieve design closure, which heavily relies on human knowledge and experience. Imitating the manual design process, RL algorithms can be leveraged to address these problems effectively.

In the following sections, we use three cases to investigate why and how RL algorithms are applied to different EDA problems.

III. Case 1: Chip Macro Placement

We show why and how we use RL algorithms to handle the macro placement in this section.

A. Why is RL a good fit for the placement problem?

In chip placement, we determine the locations of movable instances (e.g., macros and standard cells) in the physical layout. In the perspective of the optimization problem, the variables are the locations of movable instances. There are several constraints on this problem. For instance, no overlap is allowed between any two instances. Several instances should be located in a sub-region of the whole placement canvas. The objective function is usually a criterion to measure the performance of the chip design, such as the wirelength.

Prior to RL approaches, the state-of-the-art algorithm is analytical placement [9], which consists of three stages: global placement, legalization, and detailed placement. In global placement, we perform relaxation on the original constraints such that the optimization problem is solvable with existing

methods. Legalization recovers the original constraints, and detailed placement refines the solution.

There are usually two kinds of instances in the very-large-scale integrated (VLSI) circuits, macros and standard cells. The macros may have irregular shapes and sizes, while the standard cells follow the library specifications. Although the number of large macros (usually less than one thousand) is much smaller than that of standard cells (in the magnitude of millions), the location of large macros has a significant impact on the performance of circuit designs. The final solution is more sensitive to the tiny movement of a macro due to its large size and irregular shape.

RL is a promising approach to device placement optimization, the problem of efficiently partitioning large neural network models across multiple computing devices [10], [11], [12], [13], [14]. Given the similarity of the chip and device placement problems, it is natural to consider applying a similar approach to the chip placement problem. The chip placement problem, however, introduces several new challenges. For example, a chip netlist can have hundreds of macros and millions of standard cells, representing a search space of magnitude larger than the device placement problem. Furthermore, in device placement, the actual cost function is easy to quickly and accurately measure. In contrast, we care about the final results when the design closure is achieved for chip placement, which can take several days with commercial EDA tools.

In [15], Google proposed a deep RL approach to macro placement. Unlike prior approaches to this problem, the RL-based method improves as it solves more instances of the chip placement problem, and is capable of generalizing across chip blocks. Building on this work, [16] integrates DREAMPlace [17] into the loop of the RL algorithm to improve performance further and reduce the execution time.

B. How do we apply RL to the placement problem?

General framework. We convert the placement problem into an optimization problem, following the formulation in Section II-B. As proposed in [15], we place one macro at each time step. After placing all macros, we call the reward function as shown in Equation (1) to obtain the performance of this episode. We use a neural network to model the RL policy function. Given the reward signal at the end of each optimization episode, we use policy-based methods (such as PPO [18]) to update the parameters of the policy function such that it generates better placements over time.

Approximation. Due to the enormous size of the design space, we must apply approximations to make the problem tractable. Similar to many RL algorithms, we discretize the state and action space. Specifically, we only place macros on the grids of the placement canvas. Moreover, we only use RL algorithms to place macros. Standard cells are placed with the force-directed method or DREAMPlace [17] in the environment since the number of standard cells is large. We refer the readers to [15], [16] for more details regarding the approximations.

Representation learning. Placement takes in an abstract hypergraph representation (a netlist with devices as nodes and interconnections as hyperedges) and outputs a geometric representation (a layout). Information in the input hypergraph is critical for optimizing the placement, as reward functions such as wirelength and congestion are heavily impacted by the netlist information. We, therefore, take a graph convolution network to extract representations of the netlist graph structure, which can be used in downstream tasks. We use this graph neural network as the encoder for the policy and value networks of the RL agent. We pre-train the policy on a large set of training netlists. These learned graph representations enable the generalization to previously unseen circuits with fine-tuning.

IV. Case 2: Analog Transistor Sizing

In collaboration with Intel, we have developed an RL-inspired black-box optimization algorithm tailored for analog circuit sizing, which combined the strengths of deep neural networks and reinforcement learning to achieve state-of-the-art results [19].

A. Why do we use RL on the sizing problem?

In analog sizing, design parameters are configured in a such a way that the resulting design satisfies the predetermined performance requirements. Typically, circuit instances such as transistors, resistors, capacitors, etc. are sized to tune the performance. We formulate analog circuit sizing task as a constrained optimization problem succinctly as below.

$$\begin{aligned} &\text{minimize} \;\; f_0(\mathbf{x}) \\ &\text{subject to} \;\; f_i(\mathbf{x}) \leq 0 \quad \text{for } i = 1, \ldots, m \end{aligned} \tag{2}$$

where $\mathbf{x} \in \mathbb{D}^d$ is the design parameter vector, and d is the number of design variables of the sizing task. $f_0(\mathbf{x})$ is the objective performance metric we aim to minimize. Without loss of generality, we denote i^{th} constraint by $f_i(\mathbf{x})$. In general, a weighted summation of the objective and constraints is used as the Figure of Merit (FoM) value to assess the overall quality of a design.

The majority of the proposed algorithms are simulation-based optimization methods in the community. Since the simulations for analog circuits can be very time-consuming, efficient optimization methods are sought. This led to an increased focus on using surrogate models for better utilization of the existing samples. Hybrid evolutionary or derivatives of Bayesian optimization have successfully adapted to the area. However, such algorithms suffer from scalability issues since they are limited to a small number of parameters and typically has high modeling cost.

Then learning-based algorithms, such as RL, are applied in the field. When applying RL methods in the sizing problem, we are motivated by the idea that the state of the design is represented by the values (sizes) of the circuit parameters, and changing the values of these parameters can be viewed as taking actions in the MDP space. Furthermore, the resulting circuit FoM corresponds to the return obtained from taking that action. In this perspective, we can train an agent that learns how to change design parameters to improve the performance metrics, which is the goal for the analog sizing task.

B. How do we apply RL on the sizing problem?

General Framework. In order to solve the zero-order analog sizing optimization problem, we developed an RL-inspired framework, called DNN-Opt [19]. It comprises a two-stage deep neural network architecture that interacts with a circuit simulator during the optimization process. A critic-network is used as a proxy to circuit simulator to predict any new design point's performance. The actor-network uses the predictions from the critic-network to find a potential good design when the optimization flow is iterated. In this way, we efficiently explore the design space to find good designs. Besides, the actions are further regularized by adopting a population control scheme.

The two-stage network architecture of our work borrows its structure from the Deep Deterministic Policy Gradient (DDPG) algorithm [20], which is an RL actor-critic algorithm [21] developed for continuous action spaces. However, we do not apply the DDPG algorithm directly due to efficiency considerations. Instead, we adapt it with significant modifications and tailor it for the analog circuit sizing problem.

Adaptation for Sizing Task. In our work, we represented the state of a design by the optimization parameters (circuit design variables), and each action corresponds to a change in the optimization parameters vector. In expectation, the actions are taken to change design variables to result in better FoM circuits.

Critic-Network: Originally, a critic-network can approximate the return for a state-action pair. In our case, we modify its role and use it as an approximator of the costly SPICE simulator. At the output layer of critic-network, each node approximates a spec value. We later use this layer to calculate an FoM approximation which can be interpreted as the return value. Since we want the critic to predict spec values, the following Mean Squared Error loss function is used for training.

$$L\left(\theta^{Q}\right)=\frac{1}{N_b(m+1)}\sum_{k=1}^{N_b}\sum_{l=1}^{m+1}\left(Q(\mathbf{x}_k,\Delta\mathbf{x}_k)^l-f(\mathbf{x}_k+\Delta\mathbf{x}_k)^l\right)^2 \quad (3)$$

where $Q(\mathbf{x}_k,\Delta\mathbf{x}_k)^l$ is the critic-network's approximation for k^{th} sample's l^{th} performance and $f(\mathbf{x}_k+\Delta\mathbf{x}_k)^l$ is the SPICE simulated value for the same design-performance pair.

Actor-Network: An actor-network, in general, is utilized to determine the action for a given state. In DNN-Opt, it provides the change in the design parameter vector for a given design. To explore the design space and find potential good designs, we train actor-network based on the predictions of critic-network. Therefore, training of actor-network is done after critic network is trained and its hyperparameters are fixed. We use the circuit performance vector and custom normalization coefficients to define an FoM function, $g(\cdot)$, in the following form.

$$g\left[f(\mathbf{x})\right]=w_0\times f_0(\mathbf{x})+\sum_{i=1}^{m}\min\left(1,\max(0,w_i\times f_i(\mathbf{x}))\right) \quad (4)$$

where w_i is the weighting factor. Note, a $\max(\cdot)$ clipping used for equating designs after constraint are met and $\min(\cdot)$ clipping is used for practical purposes to prevent single constraint violation to dominate $g(\cdot)$ value. In the FoM expression, we can replace the real simulation values $f(\cdot)$ with the critic-network predictions $Q(\mathbf{x},\Delta\mathbf{x})$ in order to quantify the quality of any point in design space. In this context, for a batch-size of N_b samples, the following loss function is used to train actor-network parameters.

$$L\left(\theta^{\mu}\right)=\frac{1}{N_b}\sum_{k=1}^{N_b}\left(g\left[Q(\mathbf{x}_k,\mu(\mathbf{x}_k\mid\theta^{\mu}))\right]+\|\lambda*\mathrm{viol}_k\|_2\right) \quad (5)$$

where $\mu(\mathbf{x}_k\mid\theta^{\mu})$ is proposed parameter change vector $\Delta\mathbf{x}_k$ by the actor-network. The statistics of a population based on the 'elite" solutions is used to restrict the search space. We penalize the actor-network proposals resulting in designs outside this restricted region which is reflected at the second part of the loss function. This regularization mimics the population based search mechanism where the search space is restricted based on the elite solutions and hyperparameters of the optimization algorithm (e.g., evolutionary).

V. Case 3: Optimization in Logic Synthesis

In this case study, We show why and how to apply RL for finding the sequence of optimization operations in logic synthesis problem [22].

A. Why do we use RL on the logic synthesis problem?

One of the core problems in logic synthesis is how to optimize a Boolean logic without altering its function. A typical methodology to this problem is to represent the logic as a logic graph, such as an and-inverter graph (AIG), and then perform graph operation to optimize the graph size and logic depth. Those graph operations are sub-graph optimization steps that alter the topology of a logic graph with an equivalent logic representation. In practice, a sequence of different graph operations are applied on the logic graph to find solutions. However, finding the best optimization step receipt is rarely studied.

Typical sub-graph optimization steps replace the sub-graphs with equivalent logic. The outcome of a certain operation is deterministically based on the current graph structure. Therefore, the logic optimization process can be viewed as a dynamic decision making process. Given an initial logic graph, we operate a certain number of sub-graph optimization steps, and our objective is the outcome after these steps. We observe a current logic graph at each step and choose a graph operation from candidate actions. This procedure is well aligned with a typical MDP formulation.

B. How do we apply RL to the logic synthesis problem?

General framework. We employ a well-adopted open-source academic logic synthesis framework ABC [23] to build our environment. The targeting Boolean logic is first converted in AIG representation. Then, for each episode, we apply a fixed number of graph operations on the AIGs to find the lowest number of nodes and logic depth of the final AIG. At each step, the agent obverses the current AIG and freely choose one optimization operation from five actions, balance, rewrite, refactor, rewrite with zero-cost replacement, and refactor with zero-cost replacement. All of the five actions are implemented in the ABC framework and are deterministic. We formulate the rewards as the improvement on the number of nodes and logic depth, which is easily calculated by

comparing the current and initial logic graphs. The MDP state, on the other hand, is more challenging to represent. Intuitively, the state is the current AIG. However, the graph-structural data cannot be directly embedded into a vector representation for the RL agent. On the one hand, we use graph statistics and action history as part of state representation. On the other hand, we also apply a graph neural network to obtain graph embedding as part of the state vectors.

Graph Embedding. We use a graph convolutional network to obtain a graph embedding for an AIG network. We first represent the AIG as a directed graph. We use an one-hot vector as the initial node features to distinguish the node type. The graph then passes four graph convolutions as shown in the Equation 6.

$$h_i^{(k)} = \sigma\Big(\sum_{j \in N(i)} \frac{1}{c_{ij}} h_u^{(k-1)} W^{(k-1)} + b^{(k-1)}\Big), \qquad (6)$$

where $h^{(k)}$ is the k^{th} layer output, $\sigma(\cdot)$ is the activation function, $N(i)$ represents the neighbors of node i, W is a weight matrix and b is the bias. After the graph convolutions, each node has a new node embedding. Then we compute the mean of the node embeddings and obtain the final graph embedding. The resulting vector is concatenated with other state features.

With the MDP formulation above, we apply REINFORCE [24] algorithm in our RL agent. For each circuit, the RL agent searches operation receipts. We observe the performance of the RL agent improve over the episode and outperform existing operation sequence heuristics at the cost of search time.

VI. Future Directions

So far, we have discussed why and how we can use reinforcement learning for electronic design automation through three case studies. In this section, we delve into the nature of reinforcement learning and propose some future research directions of RL in EDA.

A. Challenges

High Cost on learning materials and computation. Reinforcement learning algorithms usually require extensive exploration before the learning agent can leverage the acquired knowledge. Evaluating the reward function in EDA problems requires slow and often expensive EDA tools. Since RL needs many iterations to converge, calling an EDA tool in the loop of RL training would significantly slow down the training. This problem can be mitigated by running multiple parallel instances of EDA tools to collect data in large batches. However, commercial EDA licenses are expensive, and open-source alternatives are typically unreliable for most EDA tasks.

Another related challenge is that most real-world benchmarks are not public or widely accessible. For example, public benchmarks for the chip placement problem are outdated and do not reflect modern hardware and technology node sizes. Moreover, it is usually expensive to generate and process such datasets. For example, it typically takes several hours to run the complete design or simulation flow.

The inherent complexity of EDA problems exacerbates this issue. Namely, we need more training data and computational resources in order to be able to handle more complex problems. This high cost often necessitates approximations to the state and action spaces and the reward function.

The computational cost of training RL algorithms can also be substantial, and most successfully deployed RL policies were trained on distributed platforms. Although this cost can be easily justified when developing commercial hardware, it may be prohibitively expensive for academic researchers.

Generalization and Transferability. The ability to generalize and transfer knowledge are critical properties of an algorithm. For example, with the simplex method or the interior point method, we can solve linear programs. Thus, once a problem can be formulated as a linear program, we can solve it effectively due to the generality of the above algorithms.

Given that an RL agent achieves high performance on a particular problem instance, we wonder how much the RL agent's knowledge transfers to other related problems, allowing it to generalize to new previously unseen examples quickly.

However, generality and transferability are known concerns of the RL algorithms. The RL agent and the representation learned by deep neural networks usually cannot extrapolate effectively. If we intend to tackle different benchmarks, we have to conduct fine-tuning or learning from scratch.

In [15], it is shown that as the policy is trained on a larger set of chip netlists, its performance on unseen netlists improves. This means that the policy is able to transfer the knowledge it acquired by placing previous chips and applying it effectively to new chips.

Interpretability. Interpretability is also a known challenge for the current reinforcement learning algorithms. The theoretical analysis and explanation regarding deep learning are currently insufficient. Hence, we cannot interpret the reinforcement learning algorithms thoroughly, which means the learned RL agent is essentially a black box for us. While we may not know why an RL agent makes a given decision, we can observe that that decision yields a better solution.

Human experts usually know what they have learned from particular experiments, but they cannot always explain their decisions or summarize their experiences. If we could understand and interpret deep reinforcement learning, we could not only solve problems human experts cannot handle, but also obtain insights, which help us understand these problems and their solutions.

Optimality. When we tackle optimization problems with reinforcement learning algorithms, there is no theoretical guarantee that optimal results are achieved. We do not even know the gap between the obtained solutions and the optimal ones. Hence, we are unaware of the solution quality compared with the optimal solution.

B. Opportunities

Learning capacity. With deep learning, we can enlarge the model capacity and extract better representations from large amounts of data, which can then be used in downstream tasks. For example, we can train on a large corpus of chip netlists, derive expressive embeddings for these chips, and use these

representations to guide decision-making in various optimization tasks, such as placement, block shaping, pin assignment, and routing. In addition, RL can help with tackling function approximation problems in high dimensions. Analytical solutions such as linear regression or Gaussian processes suffer from the curse of dimensionality, whereas deep RL has more modeling capacity with the high dimensional feature spaces. In [19], RL demonstrates strong results for tuning large number of circuit design variables.

Prior knowledge. When using reinforcement learning to solve a specific problem, we can integrate the prior knowledge to reduce the complexity of the problem. For example, the AlphaGo [25] learns from extensive training from human and computer play, which consists of the knowledge from human players. On the contrary, it is also possible to achieve better results without prior knowledge. The AlphaGo Zero [2] surpasses all the old versions of AlphaGo with playing games against itself.

When we apply RL algorithms, we can integrate our prior knowledge regarding the related problems. This prior knowledge will guide the RL agent to explore the environment. We can also discard the knowledge and allow the RL agent to discover the whole design space. In this way, we may mitigate the issue induced by our prejudice and bias.

Active community. Many researchers are active in the machine learning community. Reinforcement learning has received more and more attention in terms of both its underlying theory and its practical applications. The abundance of open-sourced algorithms allows peers to contribute faster development of the field. Furthermore, a successful application in one area is easily reflected in the areas with similar problem formulation.

VII. Conclusion

In this paper, we analyze why and how RL algorithms are applied in the field of EDA. We also demonstrate its potential by presenting three successful case studies in macro placement, transistor sizing, and logic synthesis. We further discuss what challenges and opportunities arise from deploying such algorithms in EDA problems. We believe reinforcement learning, combined with other AI and analytical/heuristic algorithms can reshape and even revolutionize the field of electronic design automation.

Acknowledgement

This work is supported in part by NSF under grants 1704758, 1718570, and 2112665.

References

[1] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 2019.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[3] H. Ren, S. Godil, B. Khailany, R. Kirby, H. Liao, S. Nath, J. Raiman, and R. Roy, "Optimizing vlsi implementation with reinforcement learning," in *2021 International Conference On Computer-Aided Design (ICCAD)*, IEEE/ACM, 2021.

[4] M. Rapp, H. Amrouch, Y. Lin, B. Yu, D. Z. Pan, M. Wolf, and J. Henkel, "Mlcad: A survey of research in machine learning for cad keynote paper," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[5] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, X. Ning, Y. Ma, H. Yang, B. Yu, H. Yang, and Y. Wang, "Machine learning for electronic design automation: A survey," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, June 2021.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[7] C. Audet and M. Kokkolaras, "Blackbox and derivative-free optimization: theory, algorithms and applications," 2016.

[8] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, p. 105400, 2021.

[9] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "RePlAce: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2019.

[10] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 2430–2439, PMLR, 06–11 Aug 2017.

[11] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A Hierarchical Model for Device Placement," *International Conference on Learning Representations*, 2018.

[12] R. Addanki, S. Bojja Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh, "Learning Generalizable Device Placement Algorithms for Distributed Machine Learning," in *Advances in Neural Information Processing Systems*, 2019.

[13] Y. Zhou, S. Roy, A. Abdolrashidi, D. Wong, P. C. Ma, Q. Xu, H. Liu, M. P. Phothilimtha, S. Wang, A. Goldie, A. Mirhoseini, and J. Laudon, "Transferable graph optimizers for ML compilers," *CoRR*, vol. abs/2010.12438, 2020.

[14] A. Goldie and A. Mirhoseini, "Placement optimization with deep reinforcement learning," in *Proceedings of the 2020 International Symposium on Physical Design*, ISPD '20, (New York, NY, USA), p. 3–7, Association for Computing Machinery, 2020.

[15] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, *et al.*, "A Graph Placement Methodology for Fast Chip Design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[16] Z. Jiang, E. Songhori, S. Wang, A. Goldie, A. Mirhoseini, J. Jiang, Y.-J. Lee, and D. Z. Pan, "Delving into macro placement with reinforcement learning," in *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–3, IEEE, 2021.

[17] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 748–761, 2021.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[19] A. F. Budak, P. Bhansali, B. Liu, N. Sun, D. Z. Pan, and C. V. Kashyap, "DNN-Opt an rl inspired optimization for analog circuit sizing using deep neural networks," in *Proceedings of the 58th ACM/EDAC/IEEE Design Automation Conference*, DAC '21, 2021.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *ICLR*, 2016.

[21] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *SIAM Journal on Control and Optimization*, MIT Press, 2000.

[22] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *2020 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, 2020.

[23] A. Mishchenko, "ABC: A system for sequential synthesis and verification."

[24] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.