A General Framework For VLSI Tool Parameter Optimization with Deep Reinforcement Learning

Anthony Agnesina, Sai Pentapati, Sung Kyu Lim
Georgia Institute of Technology
agnesina@gatech.edu

Abstract

Electronic design automation (EDA) tools and flows have steadily increased in complexity over the years, with modern tools offering more than 10,000 parameter settings, rendering the optimum tuning of such tools possible for only expert users. Automating this parameter setting for power-performance-area optimization would democratize modern EDA tools and VLSI physical design. In this paper, we present a general way of casting the parameter optimization problem into a reinforcement learning task. The resulting agent is then assigned to optimize a 2D VLSI placement step with proof-of-concept results. We conclude with a discussion of our ongoing work and how the methodology can be applied to 3D partitioning.

1 Introduction

Assessing power-performance-area (PPA) for physical design (PD) of increasingly large design spaces, ranges from hours to days, such that only a small portion of the possible parameters sets can be explored economically, resulting in designs that are, by definition, suboptimal.

Inspired by the recent work from Google [1] demonstrating the success of deep reinforcement learning (RL) applied to chip macro placement, we propose to use deep RL to automatize the search of optimal PD tool parameters and, because the PD flow consists of many stages performed sequentially, optimize each intermediate stage individually.

2 Parameter Optimization as an RL Problem

We believe PPA optimization can be reduced to an RL problem using a general RL framework by making the correspondences to the four key RL components described in Table 1.

Problem Definition Given a netlist n, we wish to find $p^* = \arg\max_{p \in \mathcal{P}} \mathrm{PPA}_{tool}(n, p)$ where $\mathcal{P}$ is the space of all parameter sets. The black-box function PPA_{tool} is not available to the learner and can only be evaluated at a query point (n, p), where queries are very expensive.

A Sequential Approach Let $s_t = (n, p_t)$ be the state at timestep t ($\equiv$ query). An action a_t turns p_t into p_{t+1}. The environment is then queried — the commercial tool is ran with parameters p_{t+1} for netlist n — and returns the reward R_t, a stochastic evaluation of $\mathrm{PPA}_{tool}(n, p_{t+1})$. The new state is $s_{t+1} = (n, p_{t+1})$. After L evaluations, we output our best guess $p^{\text{Guess}} = p(L)$, which could be different from p_L. This procedure is shown in Figure 1.

Solving this problem reduces to learning a sequential optimizer represented as an optimal policy π^* prescribing the optimal action to take in a specific state. The optimizer should learn the optimization

Workshop on Machine Learning for Systems, 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

RL	PD
environment	commercial EDA tool black-box
state	each PD parameter is tuned to a specific setting, plus netlist information
action	change in a deterministic manner the setting of a subset of parameters
reward	large if the gap between desired vs. achieved PPA is reduced

Table 1: PD Parameter Optimization translated into an RL problem.

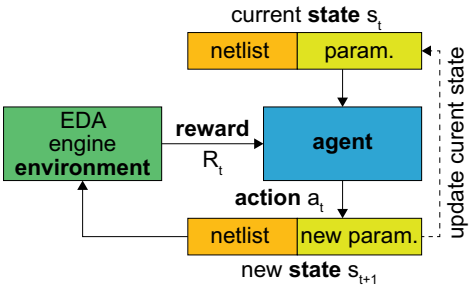

Figure 1: RL agent-environment interaction in the proposed methodology.

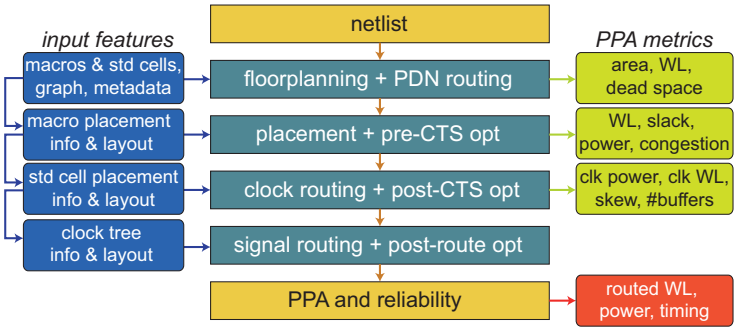

Figure 2: Our proposed inputs (states) and outputs (rewards) for each step in the sequential PD flow.

process for all possible netlists $\mathcal{N}$. A policy π is good if it maximizes the following expectation:

$$\mathbb{E}_{n\sim\mathcal{N}}[\mathrm{PPA}_{tool}(n, p^{\mathrm{Guess}})] \tag{1}$$

where p^{Guess} is found on the trajectory following π.

3 A General RL Framework

The components in our RL framework consist of four parts: Environment, State, Actions, and Reward.

Environment A black-box EDA engine capable of solving large NP-complete problems.

States The articulation of features (tool parameters & netlist information) for a given step, as illustrated in Figure 2.

▪ **Tool Parameters** The actual PD tool parameters themselves, dependent on the given step.

▪ **Netlist Features** The set of characteristics required to transfer knowledge learned from previous optimizations for use with a new, unseen netlist:

- ***Meta-data information*** describes the setting of the PD flow and high-level information about the netlist such as the technology, floorplan, macros area and circuit type.
- ***Image features*** such as layouts of placement, routing, flip flop distribution, etc. The rationale is that images can capture hard to hand-engineer information about designs. Work from [2] has demonstrated the efficiency of using placement layouts to predict clock tree metrics.
- ***Graph features*** from connected directed graphs of VLSI circuits. To capture rich local and global properties of the netlist, we use:
  - ◇ HANDCRAFTED TOPOLOGICAL FEATURES We borrow concepts from graph theory to capture complex topology characteristics from the netlist such as strongly connected components, maximal clique, k-colorability or spectral characteristics.

◇ GRAPH NEURAL NETWORK (GNN) FEATURES To complement these features with a learned embedding of the graph, we use a graph neural network (GNN). We propose to use the recurrent equations for edges and nodes updates:

$$\begin{aligned} \text{Edge } e=(u,v): m_e^{(k+1)} &= \Phi(x_u^{(k)}, x_v^{(k)}, m_e^{(k)}) \\ \text{w. } m_e^{(0)} &= (\text{distance}(u,v), \text{HP/routed-WL}(e), ...) \end{aligned} \tag{2}$$

$$\begin{aligned} \text{Node } u: x_u^{(k+1)} &= \Psi(x_u^{(k)}, \rho(\{m_e^{(k+1)} : (v,u,e) \in E\})) \\ \text{w. } x_u^{(0)} &= (\text{location}(u), \text{area}(u), \text{type}(u), \text{delay}(u), \text{fanout}(u), ...) \end{aligned} \tag{3}$$

where Φ combines the edge feature with the features of its incident nodes and Ψ aggregates incoming nodes messages using the reduce function ρ. Once node/edge embeddings are obtained, a representation of the full graph is obtained as in :

$$\text{ENC}(G) = \text{CONCAT}\left(\text{READOUT}\left(\{x_u^{(k)}\}\right) | k = 0, 1, .., K\right) \tag{4}$$

which passes the graph isomorphism test if READOUT is a learnable MLP [3].

Actions A challenging step in the proposed general RL framework, our solution is to discretize the parameter space to form a finite set. If the space is discretized finely enough (the granularity requires knowledge the problem characteristics, or assumptions of a locally Lipschitz PPA function), we can define actions that make a subset of parameters change from one discrete value to another. Another solution is to define actions that modify the parameters in a precise fashion where the designer has a good understanding of the parameters effects.

Reward To capture all QoR metrics of interest for each step (see Figure 2), a reward function is adjusted to optimize the design for different trade-offs, by combining them into a single numerical value as a weighted product:

$$R(s_t) = \prod_{i=1}^{K} \alpha_i QoR_i \overset{\text{e.g.}}{\propto} \frac{1}{WL(s_t)} \cdot \frac{1}{WNS(s_t)+\epsilon} \cdot \frac{1}{Power(s_t)} \cdot \frac{1}{\#DRV(s_t)} \cdot .. \tag{5}$$

where more generally $QoR_i = f\left(\frac{QoR(\text{Desired})}{QoR(p_i)}\right)$ with an increasing function f chosen to squash QoRs into $[0, 1]$ to render values comparable. Typical examples include $f(x) \in \{\tanh(x),\ 1/(1+e^{-x}),\ ...\}$.

Agent Architecture A neural network takes the state as input and outputs both action probablities and an estimation of the value of the state, as shown in Figure 3. Our architecture includes a sequence to sequence model such as a Transformer/LSTM to model a sequential optimization process.

Answers to EDA Challenges

- ***Transfer Learning*** Because of the sparsity of data (there is no publicly available database of millions of netlists, placed designs or layouts), we propose to leverage transfer learning to extract meaningful features from the layouts without much additional training.
- ***Unsupervised GNN*** We propose to use GNN with unsupervised training to learn node/edges embeddings. The loss function favors similar embeddings for similar nodes. For example, GraphSAGE [4] uses a unsupervised loss function as the existence of a random walk between two nodes. Then as shown in Eq. (4), the full graph representation can be obtained using a shallow aggregation layer.
- ***Parallel Algorithms*** While the two methods above help define simpler networks by fixing a large part of the network, RL algorithms require many iterations over thousands of batches to learn a task well. A way to gather many samples during training is to use parallel versions of RL algorithms. As in [5], our agent learns from experiences of multiple Actors interacting in parallel with their own copy of the environment and latest copy of the network.

  To optimize $\mathbb{E}_{n\sim\mathcal{N}}[\text{PPA}(n, p^{\text{Guess}})]$ we scatter the netlists $\in \mathcal{N}$ over the different Actors.

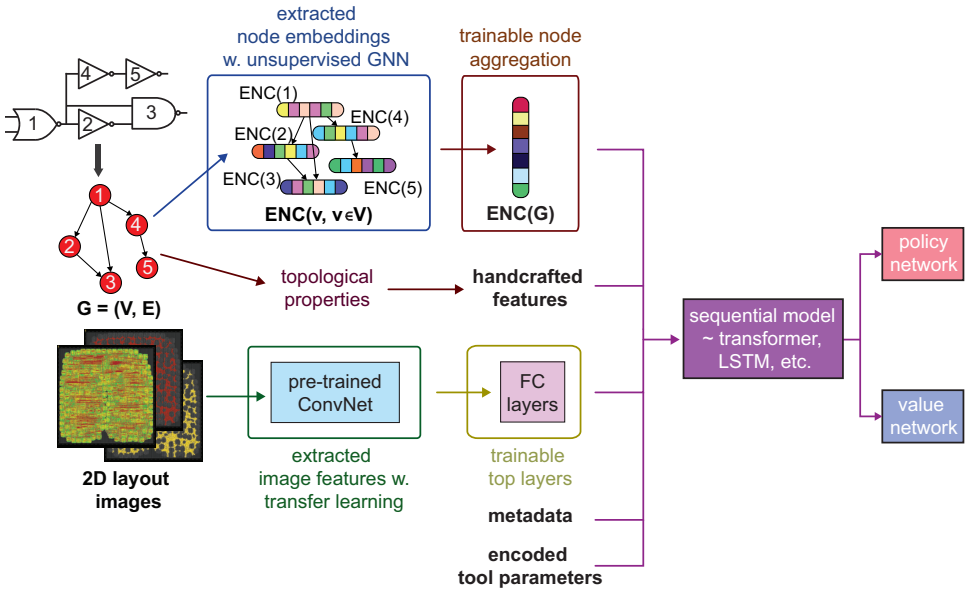

Figure 3: Our proposed general agent architecture. Netlist embeddings (graph & layout) and parameters are fed to the sequential body of the two heads of the network.

1. FLIP Booleans	7. DOWN Detailed
2. UP Integers	8. UP Global
3. DOWN Integers	9. DOWN Global
4. UP Efforts	10. INVERT-MIX Efforts
5. DOWN Efforts	11. DO NOTHING
6. UP Detailed	

Table 2: Our 11 actions.

Netlist	MAB	#iter.	RL	#iter.
LDPC	1.04 (−8.8%)	50	1.02 (−10.5%)	1
OpenPt	5.11 (−2.9%)	50	4.99 (−5.1%)	1
Netcard	4.45 (−8.8%)	50	4.34 (−11.1%)	1
Leon3	3.37 (−4.3%)	50	3.29 (−6.5%)	1

Table 3: Comparison on test netlists of best HPWL (in m) found (one iter. = one placement performed).

4 Case Study: Application to 2D Placement

We apply our framework to parameter optimization of the 2D placement step to target wirelength reduction, using *Cadence Innovus* as black-box [6]. The reward is defined as $-\text{HPWL}(p_t)$, where HPWL denotes the half-perimeter wirelength. Table 2 shows the actions we define based on domain knowledge of placement parameters. The Advantage Actor-Critic algorithm is used to train a similar network as in Figure 3, with the exception of layouts. The network is trained on 11 netlists using 16 parallel environments for $\sim$100 hours, corresponding to 14,400 placements. We test the agent optimizer on 4 unseen netlists. Results in Table 3 shows the RL agent improves over a state-of-the-art Multi-Armed-Bandit (MAB) auto-tuner. High quality wirelength and PPA is maintained after routing.

5 Ongoing Work: Application to 3D Partitioning

We use the RL framework to optimize the tier partitioning step in the state-of-the-art Pin-3D flow [7] for monolithic 3D IC design. In this flow, the design is first placed, routed and optimized on a 2D floorplan. The floorplan area is then scaled by $1/\sqrt{2}$ and a tier partitioning step distributes the cells on top and bottom dies. From the original partitioning scheme presented in [8] (bin-based with area constraints), we parametrize the step heavily and then tune it using RL.

Weighted FM Mincut As in objective-driven analytical placement methods, we decide on a *net-based* method that assigns weights to nets, so that the partitioner minimizes the total weighted cutsize, guided by metrics such as wirelength, timing and congestion. We use the FM [9] combinatorial algorithm to produce bisections with small edge-cut, and static weights computed before partitioning from the 2D routed design. Critical nets that are long, with small slacks or large delays should not

Name	Objective	Group	Type	Range
Unbalance (%)	max area unbalance per bin	bin	int	[1, 20]
# Rows	number of rows in tiled binning	bin	int	[1, 50]
# Columns	number of columns in tiled binning	bin	int	[1, 50]
α	mincut coefficient	mincut	float	$[-1, 1]$
β	wirelength coefficient	wirelength	float	$[-1, 1]$
a_0	fanout coefficient	wirelength	float	[0, 10]
a_1	squared fanout coefficient	wirelength	float	[0, 10]
γ	timing coefficient	timing	float	$[-1, 1]$
T	slack exponent	timing	int	[0, 15]
δ	congestion coefficient	congestion	float	$[-1, 1]$

Table 4: The ten 3D tier partitioning parameters we are targeting.

become 3D nets, i.e. cut, while nets traversing congested areas should go 3D to relieve congestion. If net weighting appears straightforward, it is hard to generate a good net weighting. We define weight value as the sum of individual metric weights:

$$w(e) = \alpha + \beta \cdot w_{WL}(e) + \gamma \cdot w_{Timing}(e) + \delta \cdot w_{Congestion}(e) \tag{6}$$

where:

$$w_{WL}(e) = w(e)_{HPWL} \cdot w(e)_{Fanout} = \frac{\text{HPWL}(e)}{\max \text{HPWL}} \cdot (1 + a_0 \text{fanout}(e) + a_1 \text{fanout}(e)^2) \tag{7}$$

$$w_{Timing}(e) = \frac{\text{Delay}(e)}{\max \text{Delay}} \cdot \left(1 - \frac{\text{Slack}(e)}{\max \text{Arrival Time}}\right)^T \tag{8}$$

$$w_{Congestion}(e) = \frac{1}{|\text{bins } b : e \in b|} \sum_{\text{bins } b : e \in b} \text{Cong}(b)\text{WL}(e \in b), \text{ w. Cong}(b) = \sum_{e \in b} \text{WL}(e)/\text{area}(b) \tag{9}$$

The integration of the fanout in the wirelength weight compensates for HPWL underestimating wirelength, while the congestion weight specifies how much a net participates in the congestion of the bins it traverses. The parameters that we tune are summarized in Table 4. We believe our method will prove to be superior to mincut-driven only approaches as it will adapt its partitioning focus (embodied by the parameter choices) based on the specificities of each 2D routed netlist.

Setting The environment is a combination of our custom tier partitioning engine and *Cadence Innovus* used to perform the remaining steps and extract PPA metrics. The state representation will use information of the 2D routed design such as cells locations and routed wirelength for the graph embedding, as well as layouts of placement, routing, and clock tree. To engineer a reward that captures correctly the final PPA needs, we need to perform 3D legalization and global routing after partitioning. Our reward will integrate the number of vias, WL, WNS, TNS, power and congestion overflow. Because no prior knowledge is known on the parameters presented in Table 4, we will discretize the search space as described in Section 3 to define actions.

6 Conclusions

We present a general parameter optimizer based on deep RL to generate a pre-set of improved parameter settings for each step in the physical design flow. We use a novel representation to formulate states, actions and rewards applied to tool parameter optimization. Our experimental results for the placement problem show our agent generalizes well to unseen netlists and consistently reduces wirelength compared with a state-of-the-art tool auto-tuner.

References

[1] Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E.M., Wang, S., Lee, Y., Johnson, E., Pathak, O., Bae, S., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Babu, A., Le, Q.V., Laudon, J., Ho, R., Carpenter, R., & Dean, J. (2020). Chip Placement with Deep Reinforcement Learning. *ArXiv, abs/2004.10746.*

[2] Lu, Y., Lee, J., Agnesina, A., Samadi, K., & Lim, S. (2019). GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization. *IEEE/ACM International Conference on Computer-Aided Design*.

[3] Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How Powerful are Graph Neural Networks? *ArXiv, abs/1810.00826.*

[4] Hamilton, W.L., Ying, Z., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. *NIPS.*

[5] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., & Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *ArXiv, abs/1802.01561.*

[6] Agnesina, A., Chang, K., & Lim, S. (2020). VLSI Placement Parameter Optimization using Deep Reinforcement Learning. *IEEE International Conference on Computer-Aided Design*

[7] Pentapati, S., Chang, K., Gerousis, V., Sengupta, R., & Lim, S. (2020). Pin-3D: A Physical Synthesis and Post-Layout Optimization Flow for Heterogeneous Monolithic 3D ICs. *IEEE International Conference on Computer-Aided Design*

[8] Panth, S., Samadi, K., Du, Y., & Lim, S. (2015). Placement-driven partitioning for congestion mitigation in monolithic 3-D IC designs. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*

[9] Fiduccia, C. M., & Mattheyses, R. M. (1982) A linear-time heuristic for improving network partitions. *Design Automation Conference* IEEE Press, 175–181.