CAMO: Correlation-Aware Mask Optimization with Modulated Reinforcement Learning

Xiaoxiao Liang
HKUST(GZ)

Haoyu Yang
NVIDIA

Kang Liu
HUST

Bei Yu
CUHK

Yuzhe Ma
HKUST(GZ)

Abstract

Optical proximity correction (OPC) is a vital step to ensure printability in modern VLSI manufacturing. Various OPC approaches based on machine learning have been proposed to pursue performance and efficiency, which are typically data-driven and hardly involve any particular considerations of the OPC problem, leading to potential performance or efficiency bottlenecks. In this paper, we propose CAMO, a reinforcement learning-based OPC system that specifically integrates important principles of the OPC problem. CAMO explicitly involves the spatial correlation among the movements of neighboring segments and an OPC-inspired modulation for movement action selection. Experiments are conducted on both via layer patterns and metal layer patterns. The results demonstrate that CAMO outperforms state-of-the-art OPC engines from both academia and industry.

ACM Reference Format:
Xiaoxiao Liang, Haoyu Yang, Kang Liu, Bei Yu, and Yuzhe Ma. 2024. CAMO: Correlation-Aware Mask Optimization with Modulated Reinforcement Learning. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3656254

1 Introduction

The scaling down of the modern VLSI system has aggravated the diffraction effects in manufacturing and greatly challenged the yield. Optical proximity correction (OPC) aims to compensate for the lithography proximity effects by making corrections to the patterns on a mask. It has been a long-term critical problem in chip manufacturing. Therefore, various solutions have been proposed from both industry and academia [1, 2, 3, 4, 5, 6], which leveraged empirical experience, iterative local search, or numerical optimization to generate the final mask. These methods have become the foundation for modern mainstream OPC flow in commercial tools. However, due to the inherent complexity of the lithography process, achieving the desired solution remains challenging.

Recently, the rise of machine learning (ML) techniques has provided new opportunities and solutions for OPC problems. Existing ML-based OPC techniques could be broadly classified into three categories: regression models [7, 8], generative OPC models [9, 10, 11], and reinforcement learning (RL)-based OPC [12]. Note that although A2ILT [13] also adopts RL in the OPC problem, essentially it belongs to the inverse lithography technique but leverages RL to guide the gradient calculation. Generative OPC mainly utilizes generative models such as generative adversarial networks (GANs). For instance, conditional GANs have been extensively used in previous works [9, 10, 11], in which the OPC problem is treated as an image generation task. The regression-based and RL-based OPC share a similar principle, where a model is trained to determine the movement of the edge segments in a pattern. The regression-based OPC methods [8, 7] train the model purely based on the supervision data, while the RL-based OPC [12] maintains an agent in the form of a neural network, which analyzes the layout geometry and determines how the patterns should be corrected based on the obtained reward.

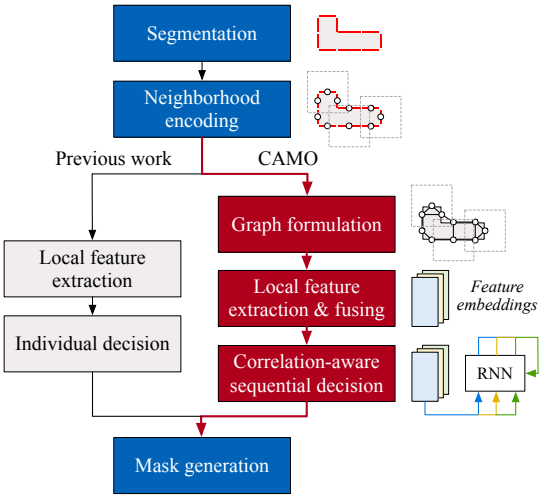

Figure 1: The comparison of the inference flow among CAMO and previous regression-based and RL-based OPC.

However, there are several issues with the aforementioned ML-OPC techniques. Generative learning-based [10] and regression-based OPC approaches [8, 7] utilize a given dataset that contains designs and optimized masks from other OPC engines as the supervision, hence their performance may be bounded by the quality of the dataset [12]. Although RL-based OPC can intrinsically perform exploration to search for optimal masks, the training is usually very inefficient due to the huge action space. Besides, previous ML-based OPC approaches are typically data-driven, while some OPC-specific principles are hardly involved in the methods, which may lead to potential bottlenecks in performance or efficiency.

The lithography principles suggest that the closely located segments are spatially *correlated*, i.e., the printed contour in a region is determined by the movements of a set of neighborhood segments. In

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656254

regression-based OPC [8, 7] and RL-based OPC [12], the prediction for each possible movement depends solely on the segment's local features, while lacking coordination among the actual movements of the neighborhood segments, which leads to fluctuation of the mask quality. Similar concerns regarding the spatial correlation in OPC were mentioned in Awad *et al.* [4], which models the regional light intensity with respect to adjacent segment pairs, such that the adjacent segments' movement can be coordinated.

In this paper, we propose a spatial correlation-aware OPC framework, namely CAMO, that processes multiple segments by capturing their local correlation. CAMO performs OPC by moving the edge segments and adopting a policy gradient-based RL paradigm. Moreover, several attempts have been made in CAMO to capture the spatial correlation in the OPC problem. Firstly, different from RL-OPC [12], CAMO encodes the layout information into a graph, where each graph node represents a segment on the pattern boundary, and the graph edge is determined by the spatial proximity between the segments. CAMO then utilizes a graph neural network (GNN) for node embedding generation, which allows the node features to fuse along the graph edges and captures more intensive local information. Secondly, taking advantage of a recurrent neural network (RNN) in sequential modeling, we employ an RNN to sequentially analyze the node embeddings and make decisions for segment movement on the fly, such that the spatial correlation can be captured and the movements among neighborhood segments can be coordinated to achieve superior mask quality.

Considering the potential huge solution space of the OPC problem, a purely data-driven learning scheme may not be efficient enough to scale to layers with complex patterns, e.g., metal layers, and hence many previous works fail to demonstrate the effectiveness on those layers [10, 12]. In CAMO, we address the efficiency and generalization issues by incorporating the domain knowledge of OPC. More specifically, an OPC-inspired *modulator* is proposed to modulate the RL agent output and boost the optimization. Therefore, our RL model can be trained much more easily while generating desired masks.

The technical contributions of this paper can be summarized as follows:

- An RL-based OPC framework is built for modern mask optimization;
- The spatial correlation is well-considered, including a GNN-based feature fusion and an RNN module in the policy structure to capture the spatial correlation when handling multiple segments;
- We propose an OPC-inspired modulation module to guide the training in RL to achieve more stable and more efficient training;
- Experiments are conducted on both via layer patterns and metal layer patterns, where CAMO outperforms state-of-the-art OPC techniques from academia and industry commercial tools.

2 Preliminaries

2.1 Reinforcement Learning

Reinforcement learning (RL) is a paradigm of how intelligent agents optimize a defined objective function by engaging in a sequence of interactions with their environment. Specifically, the interaction sequence involves a set of optimization instances denoted as *state* s. In each step, with a decision model namely policy π maintained, the agent analyzes the current state s and selects over a set of available actions A. Triggered by the selected action a, the state then transits to the next state s', on which the environment performs evaluation with the *reward* $r(s, a)$ generated. The decision and environmental interaction process is iteratively performed, which forms a Markov chain:

$$s_0 \xrightarrow{a_0} (s_1, r_1) \xrightarrow{a_1} \dots \xrightarrow{a_{T-1}} (s_T, r_T). \tag{1}$$

The continuous update ends when the final state is reached or the maximum number of steps allowed T is consumed.

The objective of the RL process is to maximize the accumulative reward over the state-updating trajectory, formulated as follows:

$$\begin{aligned} \max_{\theta} J(\theta) &= \sum\nolimits_{\tau} R(\tau) p(\tau|\theta), \\ R(\tau) &= \sum\nolimits_{t} \gamma^{t-1} r_t, \end{aligned} \tag{2}$$

where τ denotes a trajectory as depicted in Equation (1), $p(\tau|\theta)$ denotes the probability that τ occurs when the policy is parameterized by θ, and γ is the discount factor.

2.2 Policy Gradient

Policy gradient is an RL algorithm that tunes the policy parameters to increase the likelihood of taking actions that result in better accumulative rewards [14]. In the implementation, the gradient of the expected accumulative reward $J(\theta)$ w.r.t. the policy parameters θ is computed, which then serves as an estimate of the policy parameters updating direction.

3 Method

3.1 RL Environment

To build the RL framework, the key elements for RL are first illustrated, including state, action, and reward formulation.

- State (s): The state refers to the layout geometrical information, including the up-to-date mask and the target patterns.
- Action (a): Since CAMO updates the mask by batches of segments, an action a indicates the directions and strides of the movements for each segment. The action space contains valid movements $\{m_1, m_2, m_3, m_4, m_5\} = \{-2nm, -1nm, 0, 1nm, 2nm\}$, where negative values and positive values correspond to inward movement and outward movement, respectively. Hence, the dimension of the action space A in each RL step becomes 5^n, where n is the number of segments.
- Reward (r): Upon action, action a is applied to update the mask, and a reward r is generated by the environment which represents the improvement of mask quality and robustness. In this context, the reward is formulated by edge placement error (EPE) and process variation band (PV band) improvement. Similar to [12], the reward r_t is formulated as follows:
$$r_t = \frac{|EPE_t| - |EPE_{t+1}|}{|EPE_t| + \varepsilon} + \beta \frac{PVB_t - PVB_{t+1}}{PVB_t}, \tag{3}$$
where $|EPE_t|$ and PVB_t respectively indicate the EPE value and the PV band of the entire layout in step t, ε is a small constant, and β is a factor that adjusts the relative importance of EPE and PV band improvements.
- Policy (π): the policy π refers to the decision model, which is in the form of a deep neural network in CAMO and is parameterized by θ.

In each optimizing step t, the local features of each segment s_t are first encoded and fed into the policy network. The policy work

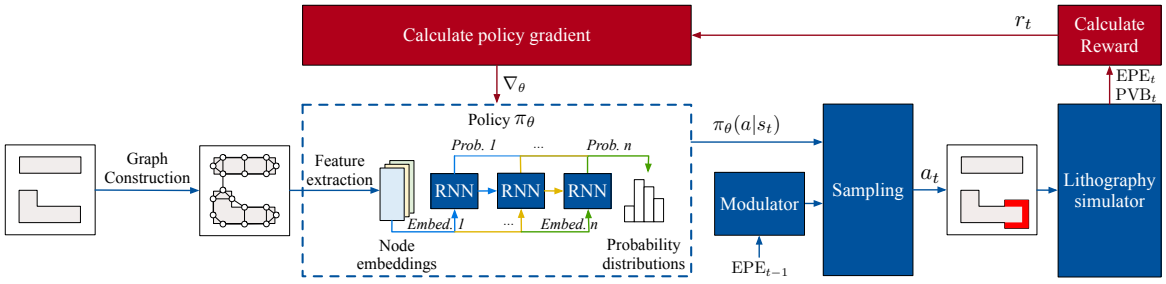

Figure 2: Overall framework of CAMO.

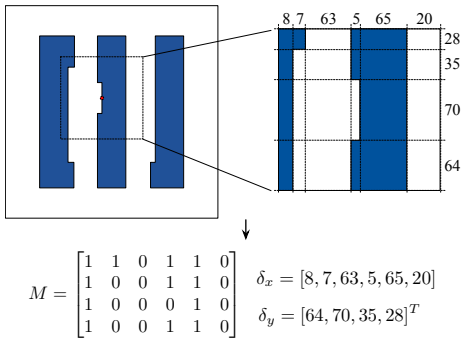

Figure 3: Illustration of squish pattern encoding. Given a control point, its neighboring geometries are encoded in M and the distance information is revealed in δ_x and δ_x.

then selects proper movements a_t and commits to the mask, which is updated and evaluated by lithography simulation. The reward r_t is then obtained by the EPE and PV band reduction upon the mask updating, and the policy network parameters θ are updated. With the updated mask, the next RL step is launched, and the iterative procedure forms the trajectory as depicted in Equation (1).

Our RL framework is revealed in Figure 2. In the training stage, given a set of target designs, the policy parameters are updated by performing epochs of mask correction and policy gradient computing. The policy is then tested on a set of unseen designs. Both stages consist of the following steps: (1) Formulate the segmented target pattern into a graph; (2) Generate node embeddings by aggregating the features of neighborhood segments in the mask; (3) Feed the node embeddings into the policy network, which selects an action a_t for each corresponding segment; (4) Commit the specified movement to segments, and perform lithography simulation to get the reward r_t; (5) Move to the next iteration, repeat (2)-(4), and additionally update the policy parameters θ upon generating the reward during the training stage.

3.2 CAMO Architecture

Graph Construction. The first step in CAMO is to construct a graph based on the given target design pattern. Following conventional steps in OPC, the pattern boundary fragmentation is first performed on the given pattern. For via patterns, the edges are regarded as segments and no fragmentation is needed. For metal patterns, the edges are evenly split into small segments according to the measure points such that the measure points are at the center of the segments. The remainder is evenly absorbed by line ends. Taking the midpoints of each segment as the control points, the input layout is initially encoded into an undirected graph $G(\mathcal{V}, \mathcal{E})$, including node set $\mathcal{V}$ and graph edge set $\mathcal{E}$.

Each node $v \in \mathcal{V}$ corresponds to the segment, and the total number of nodes is fixed since we adopt a consistent fragmentation strategy throughout the OPC process. The graph edge set E represents the geometry proximity among the segments, which is determined by the distance between control points. If the distance between two control points is closer than a threshold, an edge is added to connect the corresponding nodes. During the OPC process, the node features are renewed when the mask is updated, and the edge set will remain unchanged all the time.

Correlation-aware Policy Network Structure. In CAMO, the policy structure is a neural network consisting of a GNN-based feature extractor and an RNN module, which are respectively in charge of analyzing the local features of each segment and determining movements.

The node features are obtained by first placing a window centering at each control point and then extracting the geometry information in the window. Specifically, since the geometries on the layout are usually sparse, directly converting the neighborhood into an image by pixels may result in redundancy in image processing. Hence, we encode the neighborhood of each control point into squish pattern [15], as depicted in Figure 3. The window is first converted into a grid by placing scanlines at the edges of the sampled geometries, as well as two vectors $\boldsymbol{\delta_x}$ and $\boldsymbol{\delta_y}$ indicating the horizontal and vertical grid spacings by nanometers respectively. Subsequently, this grid is converted into a matrix $\boldsymbol{M}$, where entries corresponding to grid spaces containing geometry are marked as 1, while all others are set to 0. However, the policy in the form of a neural network necessitates a consistent input dimension, while the size of $\boldsymbol{M}$ varies depending on the layout geometries within the window. To address this inconsistency, the resulting $(\boldsymbol{M}, \boldsymbol{\delta_x}, \boldsymbol{\delta_y})$ is further converted into the adaptive squish pattern [15] to formulate a tensor $\mathcal{T} \in \mathbb{R}^{d_x \times d_y \times 3}$, where d_x and d_y denote the desired input size. This adaptive squish pattern is applied in previous RL-OPC [12]. Different from [12], we additionally formulate another $d_x \times d_y \times 3$ tensor in CAMO. The identical squish pattern encoding strategy is applied, and we place additional scanlines in both directions at the edge of the target patterns in encoding $\boldsymbol{M}$ to highlight the edge movements. By concatenating the two tensors, the dimension of our input node features becomes $\mathcal{T}_v \in \mathbb{R}^{d_x \times d_y \times 6}$.

In the feature extractor, the node features are fused along the graph edges to capture the geometry information and produce the node embeddings. One example of the features fusing is GraphSAGE [16], which involves multi-level sampling among nodes and

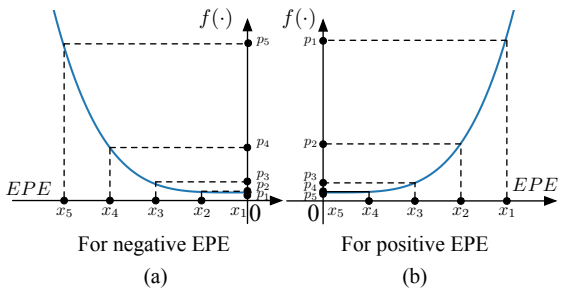

Figure 4: Illustration on the projection function. Given the EPE value of a segment, five points are evenly sampled from region $[0, EPE]$, and are then projected through $f(\cdot)$.

aggregating among sampled node features. The feature fusing on node v is formulated as follows:

$$\begin{aligned} \mathcal{G}_v^{(k)} &= \text{AGGREGATE}^{(k)}\left(\left\{\mathcal{T}_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), \\ \mathcal{T}_v^{(k)} &= \text{COMBINE}^{(k)}\left(\mathcal{T}_v^{(k-1)}, \mathcal{G}_v^{(k)}\right), \end{aligned} \tag{4}$$

where k refers to the level of sampling, $\mathcal{H}_v$ refers to the node feature of node v, $\mathcal{N}(v)$ indicates the neighbors set of node v. AGGREGATE$(\cdot)$ is the aggregation function which is usually averaging, and COMBINE$(\cdot)$ is usually concatenating or summing.

The RNN module then sequentially processes the embeddings and recurrently records the historical contexts for future reference by maintaining a *hidden state* $\boldsymbol{h}$. The forward process of a standard recurrent unit at step t is formulated as follows:

$$\begin{aligned} \boldsymbol{h}^{(t)} &= \phi(\boldsymbol{U}\mathcal{T}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}), \\ \boldsymbol{o}^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{c}, \end{aligned} \tag{5}$$

where $\mathcal{T}^{(t)}$ is the embedding of the t-th node from the GNN, $\phi(\cdot)$ is the activation function; $\boldsymbol{U}, \boldsymbol{W}$ and $\boldsymbol{V}$ are the weight parameters, $\boldsymbol{b}$ and $\boldsymbol{c}$ are bias, and $\boldsymbol{o}^{(t)}$ is the output for $\mathcal{T}^{(t)}$. Through the recurrent unit, the information and movements of previous segments are considered in processing the upcoming ones. For each embedding, the policy outputs the estimated probability distribution of selecting the five possible movements for each segment.

OPC-Inspired Modulator. Despite the action space is finite, it grows exponentially with the number of segments. Especially when handling the metal layer patterns, a great diversity of patterns leads to an extremely large search space, making the policy hard to converge. Hence, a purely data-driven approach may take tremendous time and a huge amount of data to train an agent.

In CAMO, we propose to combine the domain knowledge from the lithography principle and OPC problem to improve effectiveness and efficiency by designing a *modulator*. The modulator adjusts the likelihood of each movement being selected according to the EPE value. It is represented by a vector $\boldsymbol{p} = [p_1, p_2, p_3, p_4, p_5]$, indicating the *preference* of the five possible movements. Recall that EPE refers to the displacement between the printed contour and the target pattern and indicates the extent of light intensity overflowing or lacking at the target edges. Therefore, a modulator should satisfy the following properties:

- A large inner EPE may prefer an outward movement and discourage an inner movement of the segment, and vice versa. Thus the preferences should be more distinct.

Algorithm 1 CAMO Training

Require: θ_0: initial policy parameters; T: number of training iterations; M_0: initial mask;
Ensure: θ: trained policy parameters

```
Construct graph G(V, E) based on M_0
(EPE_0, PVB_0) ← Launching lithography simulation on M_0
for t ← 1 to T do
    s_t ← Node feature encoding based on M_{t-1} and G(V, E)
    p̂ ← Get the modulation vector based on EPE_{t-1}        ▷ Not needed in Phase 1
    a_t, π_θ(a_t|s_t) ← Sample from p̂ ⊙ π_θ(a|s_t)          ▷ Use labeled data in Phase 1 instead of sampling
    M_t ← Take actions a_t on M_{t-1}
    EPE_t, PVB_t ← Lithography simulation on M_t
    r_t ← Calculate reward by Equation (3)
    Update θ by Equation (7)
end for
```

- When EPE is small, the preferences should not be significantly biased.

From a function's perspective, it should be flat when EPE is small and becomes sharp as EPE increases.

Motivated by the above insights, we utilize a polynomial function $f(\cdot) = kx^n + b$, where k, n, and b are positive hyper-parameters, and n should be even. Then we obtain a preference vector based on this polynomial function. As shown in Figure 4, given a signed EPE value EPE, we first evenly sample five points $\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5]$ from the interval $[0, EPE]$, and guarantee that $x_1 > x_2 > x_3 > x_4 > x_5$.

The sampled points serve as inputs for the projection function $f(\cdot)$ and formulate a vector $\boldsymbol{p} = f(\boldsymbol{x})$. Then the vector is normalized by the $softmax$ function as $\widehat{\boldsymbol{p}} = softmax(\boldsymbol{p}) = [\widehat{p_1}, \ldots, \widehat{p_5}]$. After that, the vector $[\widehat{p_1}, \ldots, \widehat{p_5}]$ is the modulated preference of the corresponding movements $[m_1, \ldots, m_5]$. It can be verified that the proposed modulators satisfy the aforementioned properties.

Decision. During the inference stage, the policy selects the movement according to the modulated probability, which can be formulated by:

$$a_t = \arg\max_a \widehat{\boldsymbol{p}} \odot \pi_\theta(a|s_t), \tag{6}$$

where the modulated probability is obtained by element-wise multiplication between the output vector $\widehat{\boldsymbol{p}}$ and the probability distribution generated by the policy network. The concatenated decision a_t of n segments is then committed for mask updating and evaluation.

3.3 Agent Training

The early stage of policy training is often time-consuming due to the RL exploration in the huge search space. For boosting, we adopt a two-phase training paradigm. The training procedure is revealed in Algorithm 1.

In the first phase, the policy mimics the behaviors of other OPC engines. For each training case, we first collect the moving trajectories of each segment within a limited number of steps by another OPC engine, e.g., Calibre. In step t, after receiving the input graph with n nodes, the policy outputs the probabilities with the dimension of $n \times 5$, indicating the five possible movements to be selected for n segments. Following [14], the update rule is formulated as follows:

$$\begin{aligned} \theta_{t+1} &\leftarrow \theta_t + \alpha \nabla_\theta J(\theta_t), \\ \nabla_\theta J(\theta_t) &= \nabla_\theta r(s_t, a_t) \log \pi_\theta\left(a_t|s_t\right), \end{aligned} \tag{7}$$

where α indicates the learning rate, and $\pi_\theta(a|s_t)$ denotes the output probability distribution when the policy π_θ receives s_t as input. In this phase, the policy specifies the action a_t according to the collected trajectories and updates its parameters by the output $\pi_\theta(a_t|s_t)$.

The next phase adopts the same policy-updating rules except for the involvement of the modulator and the selection strategy of action a_t. The training procedure is revealed in Algorithm 1. As depicted in Line 5, the modulation vector is calculated. Besides, instead of specifying a_t, the policy individually samples the output probability distribution with five in length for each segment as revealed in Line 6. The n decisions are then concatenated. After mask updating, lithography simulation, and reward calculation, the policy parameters are updated as described in Line 10. Note that $\pi_\theta(a_t|s_t)$ utilized in Equation (7) is the original output of the policy that is independent of the modulator.

4 Experiments

In this section, we present experimental results on various patterns from the via layer and metal layer, and compare them with other OPC engines from academia and an industry commercial tool. CAMO is implemented with Python using PyTorch framework and runs on a CentOS-7 machine with an Intel i7-5930K 3.50GHz CPU and Nvidia GeForce RTX 3090 GPU. The Calibre-compatible lithography simulator and the corresponding Calibre OPC scripts are from an industry partner.

4.1 Experimental Setup

Dataset. In experiments on via patterns, the layouts are adopted from [17] which are $2\mu m \times 2\mu m$ clips, which contain a different number of $70nm \times 70nm$ via patterns. The training set contains 11 clips with the number of via patterns varying from 2 to 5, and the test set contains 13 clips with 2 to 6 via patterns. SRAFs are inserted by Calibre before CAMO launches and are included in squish pattern encoding as [12] did.

In experiments on metal patterns, the dataset comprises $1500nm \times 1500nm$ clips from two categories: clips from a generated GDSII layout, and clips with regular metal patterns. The layout is generated by OpenROAD [18] using standard cells from NanGate $45nm$ PDK, and the clips are randomly sampled from the metal layer.

EPE Measure Points To achieve convincing comparisons, the selection of EPE measure points follows conventional strategies as in [19]. More specifically, for a pattern in the via layer, the center of each edge corresponds to a measure point. For a pattern in the metal layer, the EPE measure points are evenly placed on the edges along the primary direction with $60nm$ spacing, which is consistent with the evaluation of the commercial tool we use. The total number of measure points is recorded in the corresponding tables.

CAMO Setup. The node features in $G(\mathcal{V}, \mathcal{E})$ are sampled with the $500nm \times 500nm$ neighborhood centered at each control point, and are encoded into tensors sized $128 \times 128 \times 6$ for via layer patterns and $64 \times 64 \times 6$ for metal layer patterns. The threshold for determination E is $250nm$. Regarding the policy network structure, we adopt GraphSAGE [16] as our feature extractor. It is followed by an RNN with an input size of 256, 3 recurrent layers, a hidden state size of 64, and a 64×5 fully connected layer. The stochastic gradient descent optimizer is employed with the learning rate α set to 3×10^{-4}. In calculating the modulator in Section 3.2, the projection function is $f(x) = 0.02x^4 + 1$. In Equation (3), ϵ and β are set to 0.1 and 1 respectively. In the following sections, the policy π of CAMO mimics the five-step trajectories collected from Calibre on the training set for 500 epochs.

4.2 Experiments on Via Layer Patterns

In experiments on via layer, we compare with ML-OPC techniques and a commercial tool Calibre [20]. We choose a state-of-the-art generative OPC model DAMO [10] and an RL-based approach RL-OPC [12] as the baseline ML-OPC techniques. The performance of baselines is provided by the authors of [10, 12]. Similar to RL-OPC, we also adopt the early-exit mechanism, i.e., the optimization stops when the EPE per via is less than $4nm$. The maximum times of mask updating are set to 10, and the mask is also initialized by moving each edge outwards for $3nm$. DAMO is a generative model, and only a one-time inference is needed to generate a mask. Thus it is the fastest in terms of the runtime. However, the generative model cannot perform any exploration on the given new designs. Therefore the masks generated by DAMO lead to a substantially larger EPE than other methods. The results are revealed in Table 1, where CAMO outperforms all three baseline methods in the EPE and PV band. Notably, even compared with the commercial tool, CAMO can achieve an EPE reduction by 20%, as well as 1.32× speedup.

4.3 Experiments on Metal Layer Patterns

For experiments on metal layer patterns, both original RL-OPC and DAMO failed to report experimental settings as well as results in their work due to the difficulty in handling more complex patterns. Implementing DAMO on metal layer patterns requires significantly more dataset to train and it is still hard to converge, thus DAMO cannot be used for comparison on this experiment. Therefore, we implement RL-OPC [12] on the metal layer as the baseline. In our implementation, the primary settings follow the description of [12]. Besides, RL-OPC and CAMO adopt identical initial masks and the early exit mechanism, where the optimization stops when the average EPE per measure point is less than $1nm$. The maximum number of mask updating times is 15. In addition, we also adopt Calibre as the baseline. As revealed in Table 2, CAMO achieves a 10% reduction in EPE and 2% improvement on PV band compared to Calibre, as well as competitive runtime. The results of RL-OPC suggest that it is difficult to converge in this experiment, which may due to the huge solution space upon transferring to the metal layer.

4.4 Effectiveness of the Modulator

To verify the effectiveness of the proposed modulator, we compare the EPE trajectories during the optimization on two testcases with/without the modulator, as depicted in Figure 5. The fluctuating trajectories for both cases without the modulator suggest that the policy is hard to converge in the huge solution space. By contrast, the modulator effectively guides a more stable search in the solution space, with which the EPE curves converge and achieve at most $64nm$ and $60nm$ respectively for cases M2 and M4.

5 Conclusion

In this paper, we propose CAMO, a spatial correlation-aware OPC system using modulated reinforcement learning. Different from other methods that individually decide the edge movements only based on their local features, CAMO captures the spatial correlation among neighboring segments by graph-based local feature fusing

Table 1: OPC results comparison on via layer patterns in terms of EPE (nm), PV band (nm^2) and runtime (s).

Design	Via #	DAMO [10]			Calibre			RL-OPC [12]			Ours		
		EPE	PVB	RT	EPE	PVB	RT	EPE	PVB	RT	EPE	PVB	RT
V1	2	7	5822	0.57	8	5837	8.11	6	**5730**	11.12	**1**	5797	3.18
V2	2	8	5836	0.56	8	5834	7.79	8	5813	8.66	**5**	**5734**	3.23
V3	3	14	8565	0.55	11	8587	8.01	13	8594	11.50	**10**	**8470**	4.85
V4	3	14	8621	0.59	12	8771	8.12	14	8679	11.79	**10**	**8576**	4.66
V5	4	18	10615	0.58	15	10775	8.18	16	10772	7.23	**10**	**10503**	3.28
V6	4	20	10739	0.58	15	10763	8.37	19	10659	11.85	**15**	**10507**	6.55
V7	5	28	12993	0.56	23	12615	8.64	23	12485	12.77	**23**	**12097**	11.08
V8	5	26	13047	0.57	19	12784	8.25	24	12547	12.42	**19**	**12437**	6.49
V9	6	30	15497	0.56	24	15454	8.70	26	15414	12.30	**19**	**15186**	3.45
V10	6	35	15088	0.58	27	15064	8.57	33	14588	12.17	**26**	**14556**	11.31
V11	6	39	15516	0.59	27	15782	8.46	31	15538	12.42	**21**	**15333**	3.32
V12	6	36	15424	0.57	23	15686	8.74	24	15464	12.59	**23**	**15204**	11.28
V13	6	32	16970	0.57	23	17035	8.43	39	17440	12.78	**14**	**16712**	9.70
Sum	58	307	154733	7.43	235	154987	108.36	276	153723	149.6	**196**	**151112**	82.38
Ratio		1.57	1.02	0.10	1.20	1.03	1.32	1.41	1.02	1.96	**1.00**	**1.00**	**1.00**

Table 2: Comparison with other OPC engines on the metal layer, in terms of EPE (nm), PV band (nm^2), and runtime (s).

	Point #	Calibre			RL-OPC [12]			Ours		
		EPE	PVB	RT	EPE	PVB	RT	EPE	PVB	RT
M1	64	49	28728	**8.65**	104	29390	16.61	**44**	**27795**	8.73
M2	84	**61**	37386	8.72	117	39139	15.79	67	**36467**	**7.78**
M3	88	81	**39430**	8.46	137	41623	16.88	**59**	39451	**7.95**
M4	100	89	45741	**8.50**	252	46892	17.13	**60**	**44961**	9.41
M5	106	**66**	47220	**8.84**	336	47041	16.62	69	**46582**	11.05
M6	112	102	49887	8.78	355	51433	17.57	**78**	**49438**	**7.97**
M7	116	89	52584	**8.92**	325	50770	17.82	**83**	**49961**	14.26
M8	24	**20**	11014	8.51	32	10770	16.13	23	**10928**	**1.11**
M9	72	50	22531	8.72	197	22360	16.44	**42**	**22032**	**8.32**
M10	120	**91**	37546	**8.95**	263	36368	16.79	95	**36849**	11.79
Sum	886	698	372067	**87.05**	2118	375786	167.78	**620**	**364464**	88.37
Ratio		1.13	1.02	0.99	3.42	1.03	1.90	**1.00**	**1.00**	1.00

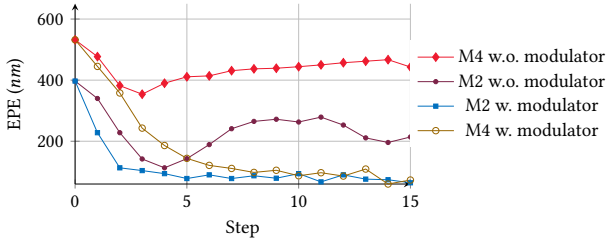

Figure 5: The EPE trajectories with / without modulator on M2 and M4 in Table 2.

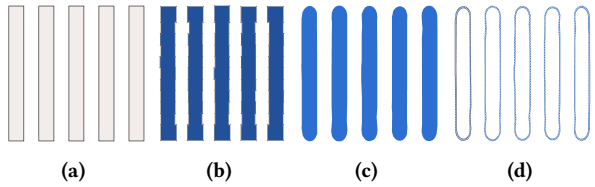

Figure 6: An example of OPC result visualization, including (a) the target pattern, (b) the mask pattern, (c) the printed contour, and (d) the PV band.

and RNN-based sequential decision. Besides, we design a modulator inspired by the intuition of OPC, which improves the effectiveness and efficiency of CAMO. Finally, CAMO outperforms state-of-the-art OPC engines from academic approaches and a commercial toolkit.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. 62204066, No. 62202190), Guangzhou Municipal Science and Technology Project (Municipal Key Laboratory Construction Project, Grant No.2023A03J0013), The Research Grants Council of Hong Kong SAR (Project No. CUHK14208021), and Hubei National Science Foundation (No. 2023AFB237).

References

[1] O. W. Otto, J. G. Garofalo, K. K. Low *et al.*, "Automated optical proximity correction: a rules-based approach," in *Proc. SPIE*, 1994, pp. 278–293.
[2] J. Kuang, W.-K. Chow, and E. F. Y. Young, "A robust approach for process variation aware mask optimization," in *Proc. DATE*, 2015, pp. 1591–1594.
[3] Y.-H. Su, Y.-C. Huang, L.-C. Tsai, Y.-W. Chang, and S. Banerjee, "Fast lithographic mask optimization considering process variation," *IEEE TCAD*, vol. 35, no. 8, pp. 1345–1357, 2016.
[4] A. Awad, A. Takahashi, S. Tanaka, and C. Kodama, "A fast process variation and pattern fidelity aware mask optimization algorithm," in *Proc. ICCAD*, 2014, pp. 238–245.
[5] A. Poonawala and P. Milanfar, "Mask design for optical microlithography–an inverse imaging problem," *IEEE TIP*, vol. 16, no. 3, pp. 774–788, 2007.
[6] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. DAC*, 2014, pp. 52:1–52:6.
[7] T. Matsunawa, B. Yu, and D. Z. Pan, "Optical proximity correction with hierarchical bayes model," in *Proc. SPIE*, vol. 9426, 2015.
[8] A. Gu and A. Zakhor, "Optical proximity correction with linear regression," *IEEE TSM*, vol. 21, no. 2, pp. 263–271, 2008.
[9] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Y. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE TCAD*, 2020.
[10] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "Damo: Deep agile mask optimization for full chip scale," in *Proc. ICCAD*, 2020, pp. 1–9.
[11] H.-C. Shao, C.-Y. Peng, J.-R. Wu, C.-W. Lin, S.-Y. Fang, P.-Y. Tsai, and Y.-H. Liu, "From ic layout to die photograph: A cnn-based data-driven approach," *IEEE TCAD*, vol. 40, no. 5, pp. 957–970, 2020.
[12] X. Liang, Y. Ouyang, H. Yang, B. Yu, and Y. Ma, "Rl-opc: Mask optimization with deep reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
[13] Q. Wang, B. Jiang, M. D. Wong, and E. F. Young, "A2-ilt: Gpu accelerated ilt with spatial attention mechanism," in *Proc. DAC*, 2022, pp. 967–972.
[14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.
[15] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Detecting multi-layer layout hotspots with adaptive squish patterns," in *Proc. ASPDAC*, 2019, pp. 299–304.
[16] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.
[17] K. Liu, H. Yang, Y. Ma, B. Tan, B. Yu, E. F. Young, R. Karri, and S. Garg, "Adversarial perturbation attacks on ml-based cad: A case study on cnn-based lithographic hotspot detection," *ACM TODAES*, vol. 25, no. 5, pp. 1–31, 2020.
[18] "The OpenROAD Project," https://theopenroadproject.org/.
[19] S. Banerjee, Z. Li, and S. R. Nassif, "ICCAD-2013 CAD contest in mask optimization and benchmark suite," in *Proc. ICCAD*, 2013, pp. 271–274.
[20] "Calibre Design Solutions," https://eda.sw.siemens.com/en-US/ic/calibre-design/.