Timing-Driven Detailed Placement with Unsupervised Graph Learning

Dhoui Lim
School of Electrical Engineering
Ulsan National Institute of Science and Technology
Ulsan, Rep. of Korea
ehdml7171@unist.ac.kr

Heechun Park
School of Electrical Engineering
Ulsan National Institute of Science and Technology
Ulsan, Rep. of Korea
h.park@unist.ac.kr

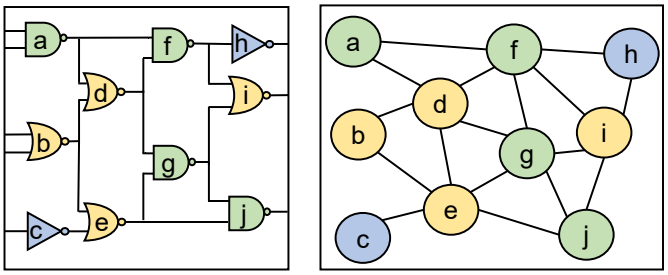
***Abstract*—Detailed placement is a crucial stage in VLSI design that starts from the global placement result to determine the final legal locations of each cell through fine-grained optimization. Traditional detailed placement methods focus on minimizing the half-perimeter wire length (HPWL) as in global placement. However, incorporating timing-driven placement becomes essential with the increasing complexity of VLSI designs and tighter performance constraints. In this paper, we propose a timing-driven detailed placement framework that leverages unsupervised graph learning techniques. Specifically, we integrate timing-related metrics into the objective function for detailed placement and formulate it into the loss function of a graph neural network (GNN) model. The loss function includes overlap, legality, and timing-related arc lengths, with appropriate weights using Bayesian optimization. Experimental results show that our framework achieves comparable or improved HPWL while significantly reducing total negative slack (TNS) by 5.5%, compared to existing methods.**

I. Introduction

In modern VLSI design, the continuous downscaling of standard cells and increasing density of circuits significantly enhance complexity across all stages of physical design. Achieving optimal results, particularly in the placement and routing stages, has become more challenging. The VLSI placement problem, widely acknowledged as an NP-hard problem, employs various analytical techniques and heuristic approaches to reduce half-perimeter wirelength (HPWL) during both the global and detailed placement stages. In addition to optimizing HPWL, eliminating cell overlaps and aligning cells properly within standard cell rows occur during the transition from global to detailed placement.

Recently, the graph neural network (GNN) has emerged as a potent deep learning model for learning intricate VLSI structures [1]. Fig. 1 illustrates the resemblance between the cells and nets in VLSI design (Fig. 1(a)) and the nodes and edges in its corresponding graph representation (Fig. 1(b)). This suggests that integrating GNN models with this graph representation enables a comprehensive understanding of VLSI structures. Several machine learning (ML) techniques leveraging graph representations and GNN models have been proposed to improve the VLSI design quality across various design stages, such as placement [2]–[4], routing [5], and additional design optimizations [6], [7].

As designs become more complex, addressing timing constraints at earlier stages becomes increasingly advantageous to avoid excessive resource consumption during timing optimization in later stages. However, considering timing metrics during placement presents significant challenges, since routing information is not yet available. Despite the challenges, active research is being conducted on timing-related factors during the placement stage [8]–[10].



(a) VLSI design (b) Graph representation

Fig. 1: Illustration of (a) a sample VLSI design and (b) its graph representation.

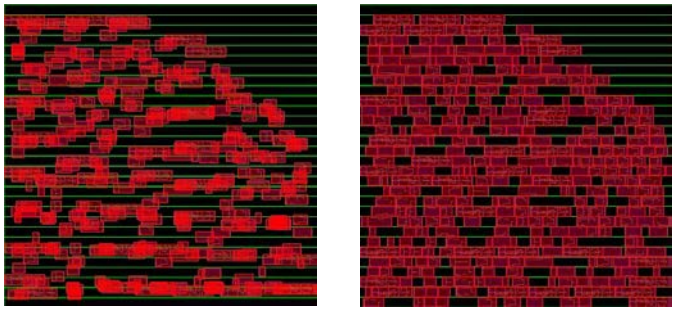
In this paper, we present a timing-driven detailed placement framework that utilizes the graph learning model. In particular, we leverage the GNN model as an unsupervised learning framework, designing a loss function aimed at improving timing-related metrics, such as worst negative slack (WNS) and total negative slack (TNS). Starting from the global placement result, we convert the design into a graph structure and feed it into our GNN model, which adjusts the positions of the standard cells toward their timing-aware detailed placement locations. A key approach in our GNN-based timing-driven detailed placement framework is that, instead of training on large datasets, we directly optimize the cell locations using a timing-related loss function. We guide our GNN model to minimize loss functions specifically designed for the detailed placement optimization goals, which include eliminating cell overlaps, aligning cells to the standard cell rows, and minimizing timing metrics simultaneously.

II. Preliminaries

A. Detailed Placement

Fig. 2 shows the layout after global placement (Fig. 2(a)) and detailed placement (Fig. 2(b)). During the global placement phase, a certain degree of overlap between cells is per-

978-3-9826741-0-0/DATE25/© 2025 EDAA



(a) Global placement (b) Detailed placement

Fig. 2: Layout examples after (a) global placement and (b) detailed placement.

mitted to minimize HPWL. Therefore, the primary objective of detailed placement is to eliminate these overlaps and determine the legal locations for each standard cell instance. However, as technology nodes continue to shrink, the impact of cell displacement during detailed placement on overall chip design quality has become increasingly significant. Consequently, it is essential to consider more on preserving design quality during the detailed placement phase. In this paper, we propose a GNN model that satisfies detailed placement constraints while focusing on reducing timing-related factors (e.g., TNS, WNS).

B. Graph Learning with GNN

Cells and nets in VLSI design can be naturally transformed into a graph structure as illustrated in Fig. 1. Leveraging this characteristic, GNNs are frequently employed in the field of electronic design automation (EDA) to effectively utilize graph-based data. In this study, we train a GNN model using graph convolution networks (GCN), which are effective at representing the information of neighboring nodes. The training process is carried out through the following graph convolution operations:

$$H^{(1)} = \sigma(\hat{A}XW^{(0)}), \tag{1}$$

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)}). \tag{2}$$

Eq.1 represents the first GCN layer, where input features X and weights $W^{(0)}$ are utilized to transform node features. Here, $\hat{A}$ denotes the normalized adjacency matrix that accounts for the graph structure, and σ is a nonlinear activation function. Eq.2 demonstrates the iterative update of the graph layer $H^{(l)}$ from the l-th layer to the $l+1$-th layer. Lastly, the output $H^{(L)}$ from the L-th layer represents the predicted node features by the network. In our work, we utilize GCN to determine the optimal positions of cells during iterative detailed placement.

C. Bayesian Optimization

Bayesian optimization [11] is an efficient framework for solving the problem of finding x that satisfies $\min_{\mathbf{x}} F(\mathbf{x})$ when the objective function $F(\mathbf{x})$ is 'expensive'. Bayesian optimization uses Gaussian Processes (GP) [12] or other suitable probabilistic models (e.g., Bayesian neural networks (BNN) [13] and Random Forests [14]) to model the uncertainty in $F(\mathbf{x})$. The optimization process involves iteratively selecting new candidate points based on probabilistic models, and the core idea is to balance between exploration (i.e., searching uncertain areas) and exploitation (i.e., refining known good areas) through the use of an acquisition function. One commonly used acquisition function is Expected Improvement (EI), which is defined as:

$$EI(x) = \mathbb{E}\left[\max(0, F(x') - F(x))\right],$$

where x' represents the current best-known value of the function. It measures the expected improvement in the objective function when evaluating a new point. The calculation of EI as follow:

$$EI(\hat{x}) = \Sigma_F(\hat{x})\left(Z\Phi(Z) + \phi(Z)\right),$$

where $Z = \frac{\mu_F(\hat{x}) - F(x')}{\Sigma_F(\hat{x})}$, and Φ, ϕ and $\hat{x}$ are the standard normal cumulative distribution, probability density functions and the candidate point being evaluated in the current iteration, respectively. By using Bayesian optimization, we can find the optimal point x induces the optimized (i.e., maximum or minimum) value of $F(x)$ with fewer function iterations, making it especially useful for expensive optimization problems.

III. Timing-driven detailed placement with GNN

A. Overall Framework and GNN Structure

Fig. 3 presents our timing-driven GNN-based detailed placement framework. Starting from the global placement result, we extract a graph representation where cells are treated as nodes and pin-to-pin arcs are represented as edges. We then add several features listed in Table I into each node, which include its current lower-left (X, Y) coordinates from the design exchange format (DEF) file, and its width, height, and area from the library exchange format (LEF) file. We also include a key feature, the driving strength of the instance, which can be obtained either from the cell type (e.g., INV_X1 $\rightarrow$ 1, NAND_X2 $\rightarrow$ 2, etc.) or directly from the cell's SPICE netlist or GDS layout in the process design kit (PDK). The driving strength of a cell reflects its ability to drive load capacitance, which plays an important role in estimating the actual post-route gate delay, in turn determining how closely connected cells must be placed to meet timing constraints. We also conduct a pre-route static timing analysis (STA) to gather information on timing-critical paths, which will be utilized as part of the loss function.

Then, we proceed to the graph learning stage to perform unsupervised graph learning related to the loss function introduced in Sec. III-B. Our GNN model consists of three GCN layers and outputs a graph with two values corresponding to the X and Y coordinates for each node. In other words, our model iteratively generates the detailed placement result while minimizing the loss value. After sufficient learning epochs, we obtain a timing-optimized detailed placement result, which is finalized with minor placement refinement if necessary.

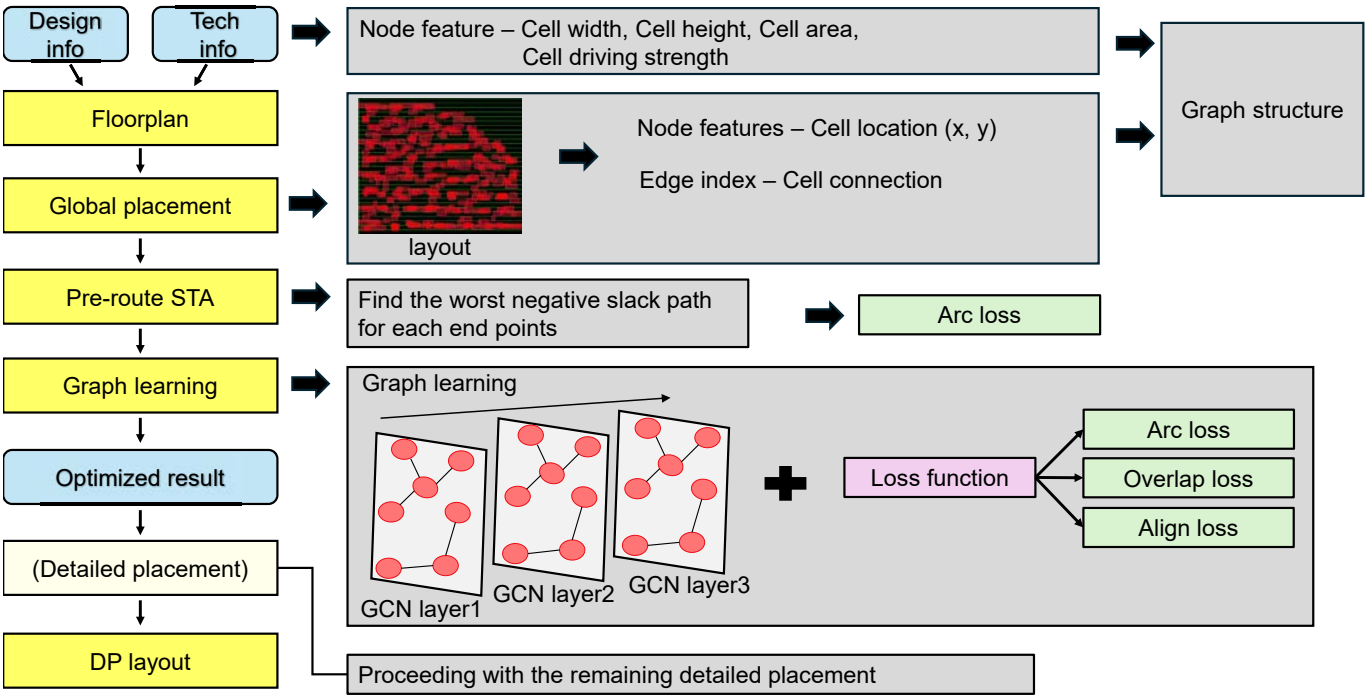


Fig. 3: Overview of our timing-driven detailed placement framework with graph learning.

TABLE I: Node features

Type	Description
X coordinate	X coordinate from global placement result
Y coordinate	Y coordinate from global placement result
Width	Cell width extracted from LEF file
Height	Cell height extracted from LEF file
Area	Cell area calculated from width and height
Driving strength	Cell's ability to drive load capacitance

B. Loss Values

In this subsection, we introduce our loss values that guide our GNN model to perform detailed placement considering timing-related factors. Our loss function consists of three main components: **overlap loss, align loss, and arc loss**. Each component is essential for achieving the primary objectives in timing-driven detailed placement: eliminating cell overlaps, aligning cells to standard cell rows, and *reducing the timing critical path delay*. Thus, by training the GNN model to minimize the loss function, the model's weights are inherently adjusted to produce optimized detailed placement results.

1) Overlap Loss: As shown in Fig. 2(a), the result of the global placement contains some degree of overlap among cell instances. To ensure that our model effectively removes the overlap between cells, we define the cell overlap area as in Fig. 4(a) and integrate it as a part of the loss function for our GNN model. The overlap area of a cell is determined by the product of the width and height of the overlapped area ($over_{width}$ and $over_{height}$, respectively). Then, the total overlap loss is computed as the sum of all cells' overlap losses

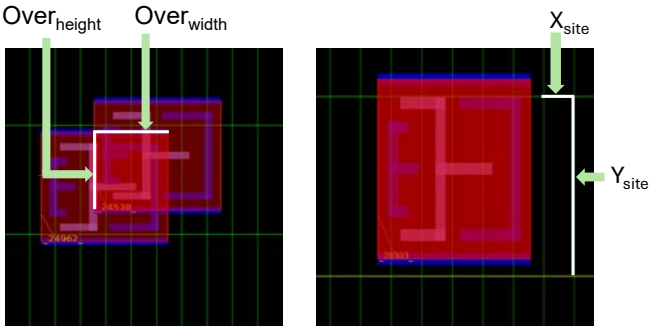


Fig. 4: Visualization of (a) cell overlap and (b) cell misalignment.

as follows:

$$Loss_{overlap} = \sum over_{width} \cdot over_{height}. \quad (3)$$

Updating our GNN model in the direction of minimizing overlap loss inherently generates results with reduced overlap.

2) Align Loss: Since locating cells onto the power/ground rails (i.e., standard cell rows) are not considered during global placement, another key objective of detailed placement is to properly align the cells with the power and ground rails. Fig. 4(b) shows an illustration of a misaligned cell. During detailed placement, we must adjust its lower-left coordinate (X, Y) to align with any site units within the chip core area. In other words, X should be a multiple of X_{site}, and Y should be a multiple of Y_{site}. To resolve this misalignment in our model, we propose an **align loss** as part of the loss function. Since a cell can be aligned with minimum displacement if it

moves to the closest site unit, the align loss for each cell is defined as the Manhattan distance between the cell's lower-left coordinate and the nearest alignment point, where both X and Y coordinates are multiples of the respective X_{site} and Y_{site}. The total align loss is then computed as the sum of the alignment losses of all cells:

$$X_{align\_loss} = \begin{cases} X_{predict} \,\%\, X_{site} & \text{if } X_{predict} \leq \frac{X_{site}}{2} \\ X_{site} - (X_{predict} \,\%\, X_{site}) & \text{if } X_{predict} > \frac{X_{site}}{2} \end{cases} \tag{4}$$

$$Y_{align\_loss} = \begin{cases} Y_{predict} \,\%\, Y_{site} & \text{if } Y_{predict} \leq \frac{Y_{site}}{2} \\ Y_{site} - (Y_{predict} \,\%\, Y_{site}) & \text{if } Y_{predict} > \frac{Y_{site}}{2} \end{cases} \tag{5}$$

$$Loss_{align} = \sum (X_{align\_loss} + Y_{align\_loss}). \tag{6}$$

Updating our GNN model in the direction of minimizing align loss inherently generates results with more cells at legal locations.

3) Arc Loss: The previous overlap loss and align loss are constraint-driven losses, which are designed to satisfy the constraints of detailed placement. If our GNN model is trained solely with these losses, it will produce legal detailed placement results with cells that do not overlap and are aligned to standard cell rows. However, the design quality may significantly degrade, such as larger HPWL, as the quality-driven factor is not included in the loss functions and, therefore, not considered by our model. In this subsection, we introduce the quality-driven loss called **arc loss** that encompasses both timing-related factors and the conventional HPWL.

We first represent the arc length by the Manhattan distance between the connected cells:

$$length_{arc} = \sum (abs(X_{start} - X_{end}) + abs(Y_{start} - Y_{end})) \tag{7}$$

Then, we assign a weight value $weight_{arc}$ to each arc length based on its timing criticality and the driving strength of its driving cell. Specifically, arcs that are part of the timing critical path and connected to weak driving cells (i.e., those with low driving strength) are assigned larger weight values, which guides our GNN model to prioritize the reduction of these arc lengths.

In detail, there are two related weight values called $nspath_{weight}$ and $driving_{weight}$. We identify if the target arc belongs to the critical path with negative slack from the pre-route STA result, and define $nspath_{weight}$ as:

$$nspath_{weight} = \begin{cases} 1 + \exp\left(\frac{slack_{path} - slack_{mean}}{slack_{mean}}\right) & \text{if } neg.\ slack\ path \\ 1 & \text{otherwise,} \end{cases} \tag{8}$$

which becomes larger for arcs included in more timing critical (i.e., worse) paths. To prevent excessively large weights, we normalize the slack value using the average slack of the negative paths. We also define $driving_{weight}$ as the reciprocal of the driving strength of the driving cell, which can be obtained from the node feature, to assign a larger weight to cells with lower driving strength:

$$driving_{weight} = \begin{cases} \frac{1}{driving\_strength} & \text{if } neg.\ slack\ path \\ 1 & \text{otherwise,} \end{cases} \tag{9}$$

Finally, we can define the $weight_{arc}$, and the total arc loss:

$$weight_{arc} = nspath\_weight \cdot driving\_weight \tag{10}$$

$$Loss_{arc} = \sum weight_{arc} \cdot length_{arc} \tag{11}$$

Updating our GNN model in the direction of minimizing arc loss inherently generates results with reduced wirelength, focusing more on the timing-related arcs.

4) Out Loss: During the first few training epochs, there is a practical issue that some cell instances are located outside the core area due to the random weight initialization of the GNN model. To ensure the stability of our training process, we introduce an additional **out loss** that guides the cells to be positioned within the defined core area during training. This helps prevent the cells from moving outside the core area while reducing overlap.

$$Loss_{out} = \sum out_{width} \cdot out_{height}, \tag{12}$$

where out_{width} and out_{height} represent the width and height of the cell area outside the core region, respectively. In contrast to other loss values, the out loss converges to zero after a few iterations.

C. Training Process

This subsection outlines the training process of our GNN model that generates the detailed placement results. We divide the training process into two steps to ensure the practical implementation of graph learning.

1) Step 1: Due to the random initialization of the GNN model's weights and biases, the early stages of training can be trivial, as the model's output is often too far from the desired placement result, resulting in nearly random placements. To mimic the scenario in which conventional detailed placement begins with the global placement result as a baseline, we first train our model to generate placement results that closely approximate the global placement outcome.

In the initial training phase, we define the loss function of our GNN model as the mean absolute error (MAE) loss between the model's output and the global placement result, i.e., the coordinate differences for each cell. Additionally, we add the global placement results (i.e., instance coordinates) into the model's output and treat them as a learnable parameter, which normalizes the output to align with the global placement results and thereby accelerates convergence. Once the loss becomes smaller than a predefined threshold (1% of difference from the global placement result in our work), we proceed to the next step to perform detailed placement concerning the loss function generated from the aforementioned loss values.

2) Step 2: At this step, we train our model to minimize the loss values introduced in Sec. III-B. Since the model's output is close to the global placement results at this moment, the out loss has nearly converged to zero and is assigned the same weight as the overlap loss during the remaining process. To prevent variations of the loss values due to various structures and scales of input circuits, we normalize the loss values to their corresponding base values. In addition, we adjust the weight assigned to the (normalized) align loss to gradually increase over time, which allows our model to prioritize overlap removal and timing optimization in the early stages, while emphasizing cell alignment in the later iterations.

Therefore, the normalized loss values are defined as below:

$$Loss_{overlap\_norm} = \frac{Loss_{overlap}}{Loss_{overlap\_base}} \tag{13}$$

$$Loss_{align\_norm} = \frac{epoch}{total\ epochs} \cdot \frac{Loss_{align}}{Loss_{align\_base}} \tag{14}$$

$$Loss_{arc\_norm} = \frac{Loss_{arc}}{Loss_{arc\_base}} \tag{15}$$

$$Loss_{out\_norm} = \begin{cases} \frac{Loss_{out}}{Loss_{out\_base}} & \text{if } Loss_{out} \neq 0 \\ 10^{-5} & \text{otherwise,} \end{cases} \tag{16}$$

where all the base-related loss values are obtained from the global placement result (e.g., $Loss_{overlap\_base}$ is the total overlap area in the global placement results, etc.).

Then, the final loss function of our GNN model is defined as:

$$\begin{aligned} Loss_{total} =& \lambda_1 \cdot (Loss_{overlap\_norm} + Loss_{out\_norm}) \\ &+ \lambda_2 \cdot Loss_{arc\_norm} + \lambda_3 \cdot Loss_{align\_norm} \end{aligned}$$

Lastly, we optimize our model's loss function by finding coefficients (λ_1, λ_2, and λ_3) and the learning rate (lr) using Bayesian optimization. Using a sample design (`uart` from [15]), we performed 30 iterations for the initial random points, followed by 200 iterations for Bayesian optimization. After optimization, we identified the best configuration as λ_1 = 6.011, λ_2 = 9.081, λ_3 = 0.2058, and lr = 0.009699.

IV. Experimental Results

A. Experimental Setup

Our experiments are conducted on a Linux server with an Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz with 128 threads and 503 GB RAM. The GNN model is trained using two NVIDIA GeForce RTX 3090 GPUs, each with 24 GB of memory. The benchmark circuits used in our experiments are obtained from and physically designed with OpenROAD [15], under the ASAP 7nm PDK [16]. All benchmarks are designed with 60% core utilization and a 1.0 aspect ratio initially. We implement the training process of the GNN model utilizing the PyTorch Geometric library [17].

B. Design Comparison

Table II shows the design comparison between the state-of-the-art open-source detailed placement engine (OpenDP) [18], and the previous GNN-based detailed placement framework that solely targets HPWL reduction (Prev.) [19], and our timing-driven GNN-based detailed placement framework. Overall, we achieve better results in HPWL and TNS, with comparable results in WNS. In detail, Our method shows an average HPWL reduction of 3.3% compared to OpenDP and 1.22% compared to the previous work, and an average TNS improvement of 5.5% compared to OpenDP and 3.18% compared to the previous work (excluding the smallest design `gcd`). It comes from our novel loss function that prioritizes the reduction of arc lengths within timing critical paths, which leads to a uniform reduction in negative slacks throughout the design. While we aim to achieve a balanced optimization between HPWL and timing compared to the previous work [19] that solely targets HPWL reduction, our results demonstrate improvements in both metrics. This indicates that our two-step training approach of first aligning with the global placement result and then *directly optimizing the arc length* yields better HPWL reduction than simply minimizing the displacement from the global placement results (i.e., the displacement loss defined in [19]).

In terms of WNS, our work shows comparable or even inferior results compared to the other two studies. This suggests that while our method effectively reduces TNS by optimizing overall timing paths, it may not adequately address the 'worst' timing path, potentially sacrificing negative slack on that path as a trade-off. Nevertheless, we believe that TNS is a more critical timing metric during the early design stage (i.e., placement) as timing violations of the small number of worst paths can be specifically managed using different techniques in the later design phases.

C. Validation of our 2-Step Training Process

In our method, Step 1 is completed within an average of 10 epochs for the model output to be close to the global placement (within 1% error), and then the training proceeds to the ordinary Step 2. Table III shows the design comparison between the flow without Step 1 and with Step 1 after 500 epochs, which claims a huge sub-optimal result of all design metrics if Step 1 is skipped. Fig. 5 validates our Step 1 from the layout perspective. If Step 1 is skipped, the initial result is almost irrelevant to the global placement results due to the random initialization of model weights, which delays the training process or even impossible to proceed (e.g., Fig. 5(a)).

D. Impact of Bayesian Optimization

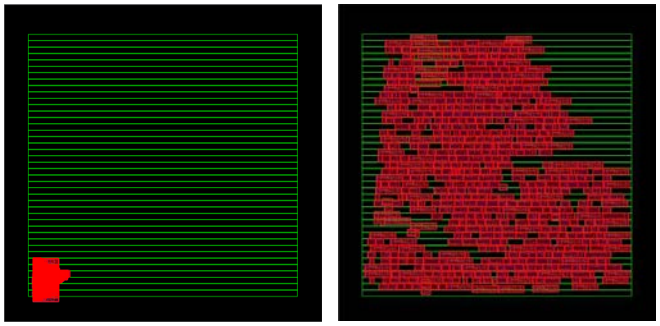
In this subsection, we demonstrate the validity of the Bayesian optimization used in our experiments. Unlike the brute-force-based searching methods (e.g., grid search) that search exhaustively across a predefined parameter space, Bayesian optimization intelligently explores more promising regions, leading to more efficient optimization. Table IV shows a comparison between the results obtained using the

TABLE II: Design comparison between the state-of-the-art detailed placement engines and our work.

Design	Clock Period *(ps)*	#cells	#nets	HPWL *(um)*			TNS *(ps)*			WNS *(ps)*		
				OpenDP [18]	Prev. [19]	Ours	OpenDP [18]	Prev. [19]	Ours	OpenDP [18]	Prev. [19]	Ours
gcd	400	343	411	697.4 (1.05)	679.7 (1.023)	664.2 (1.00)	-2594.86 (0.987)	-2564.32 (0.976)	-2627.79 (1.00)	-89.81 (0.986)	-89.45 (0.982)	-91.1(1.00)
uart	300	622	659	1219.4 (1.063)	1167.0 (1.017)	1147.4 (1.00)	-1348.42 (1.061)	-1328.57 (1.045)	-1270.92 (1.00)	-46.53 (0.999)	-47.35 (1.017)	-46.58 (1.00)
aes	700	15823	16359	56954.7 (1.034)	56400.7 (1.024)	55089.1 (1.00)	-4084.36 (1.061)	-3990.67 (1.037)	-3848.14 (1.00)	-107.73 (0.996)	-107.62 (0.995)	-108.13 (1.00)
ibex	1800	18788	19546	77996 (1.009)	77374.2 (1.001)	77324.2 (1.00)	-50910.38 (1.05)	-49270.33 (1.016)	-48480.08 (1.00)	-262.67 (1.046)	-255.98 (1.019)	-251.16 (1.00)
jpeg_encoder	1300	62115	73592	244291.5 (1.009)	242125.1 (0.996)	242166.4 (1.00)	-10775.95 (1.048)	-10534.98 (1.029)	-10278.63 (1.00)	-201.4 (1.00)	-204.4 (1.015)	-201.47 (1.00)

TABLE III: Comparison of our graph learning framework with and without Step 1.

Design	Cell displacement *(um)*		HPWL *(um)*		TNS *(ps)*		WNS *(ps)*	
	w/o Step 1	w/ Step 1	w/o Step 1	w/ Step 1	w/o Step 1	w/ Step 1	w/o Step 1	w/ Step 1
uart	5281.5	24.0	5526.2	1147.4	-4932.34	-1270.92	-110.78	-46.58
gcd	2229.0	5.8	2328.4	664.2	-4778.05	-2627.79	-143.56	-91.1



(a) After 500 epochs without Step 1 (b) After 500 epochs with Step 1

Fig. 5: Detailed placement results of `uart` benchmark without and with Step 1.

grid search method and those obtained using Bayesian optimization, both within the same parameter range. Bayesian optimization consistently results in better performance across all metrics, highlighting its effectiveness in optimizing hyperparameters.

Fig. 6 shows the changes in total loss value when using Bayesian optimization and grid search. There is a period of small loss during Step 1 of the training, and Bayesian optimization shows faster convergence to the global placement result due to the larger learning rate it finds. Furthermore, in Step 2, the loss function defined with Bayesian optimization induces the model to converge faster and find the best solution in the earlier iteration.

V. Conclusion

In this paper, we proposed a timing-driven detailed placement framework utilizing unsupervised graph learning. Specifically, we built a GNN model that directly minimizes the loss function that is directly tied to the objectives of timing-driven detailed placement, including the elimination of cell overlaps (overlap loss), alignment of the cells within standard cell rows (align loss), and consideration of timing through the minimization of defined arc length (arc loss). We also optimized the loss function using Bayesian optimization to identify the best combination of coefficients for each loss value. Our framework demonstrated superior performance compared to the state-of-the-art detailed placement engine and other detailed placement optimization frameworks, which resulted in improved placement quality and a TNS reduction.

TABLE IV: Results comparison: Bayesian optimization (BO) vs. grid search for hyperparameter tuning.

Benchmark	HPWL *(um)*		TNS *(ps)*		WNS *(ps)*	
	grid	BO	grid	BO	grid	BO
uart	1219.7	1147.4	-1364.30	-1270.92	-47.52	-46.58
aes	60173.7	55089.1	-4705.97	-3848.14	-117.62	-108.13
Parameters	grid	$\lambda_1 = 10.0, \lambda_2 = 5.0, \lambda_3 = 2.5, lr = 0.000316$				
	BO	$\lambda_1 = 6.011, \lambda_2 = 9.081, \lambda_3 = 0.2058, lr = 0.009699$				

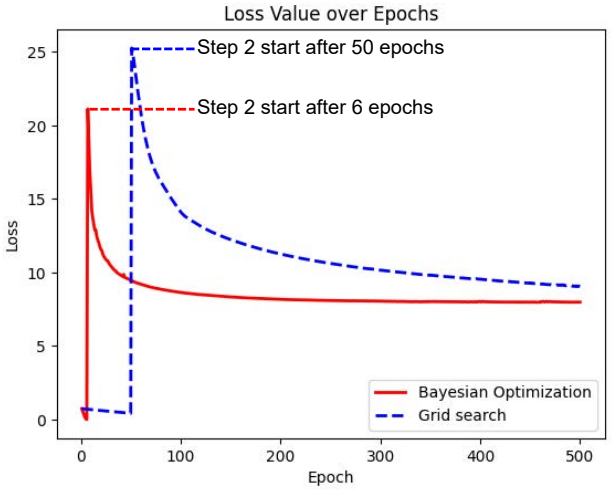


Fig. 6: Loss value changes when using Bayesian optimization and grid search.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) RS-2024-00359696, and RS-2024-00405495.

References

[1] H. Ren et al., "Why are Graph Neural Networks Effective for EDA Problems?" in International Conference On Computer-Aided Design (ICCAD), 2022, pp. 1–8.

[2] A. Mirhoseini et al., "A graph placement methodology for fast chip design," Nature, vol. 594, no. 7862, pp. 207–212, 2021.

[3] A. Agnesina et al., "VLSI Placement Parameter Optimization using Deep Reinforcement Learning," in International Conference On Computer-Aided Design (ICCAD), 2020, pp. 1–9.

[4] Y.-C. Lu et al., "Placement Optimization via PPA-Directed Graph Clustering," in Workshop on Machine Learning for CAD (MLCAD), 2022, pp. 1–6.

[5] K. Baek et al., "Pin Accessibility and Routing Congestion Aware DRC Hotspot Prediction using Graph Neural Network and U-Net," in International Conference On Computer-Aided Design (ICCAD), 2022, pp. 1–9.

[6] Y.-C. Lu et al., "RL-Sizer: VLSI Gate Sizing for Timing Optimization using Deep Reinforcement Learning," in Design Automation Conference (DAC), 2021, pp. 733–738.

[7] ——, "A Fast Learning-Driven Signoff Power Optimization Framework," in International Conference On Computer-Aided Design (ICCAD), 2020, pp. 1–9.
[8] H. Liu, S. Tao, Z. Huang, B. Xie, X. Li, and G. Li, "Instance-level timing learning and prediction at placement using res-unet network," in 2024 IEEE International Symposium on Circuits and Systems (ISCAS), 2024, pp. 1–5.
[9] P. Shrestha et al., "Graph representation learning for gate arrival time prediction," in Workshop on Machine Learning for CAD (MLCAD), 2022, p. 127–133.
[10] X. He et al., "Accurate timing prediction at placement stage with look-ahead rc network," in Design Automation Conference (DAC), 2022, p. 1213–1218.
[11] J. Wu et al., "Hyperparameter optimization for machine learning models based on bayesian optimization," Journal of Electronic Science and Technology, vol. 17, no. 1, pp. 26–40, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1674862X19300047
[12] D. J. MacKay et al., "Introduction to gaussian processes," NATO ASI series F computer and systems sciences, vol. 168, pp. 133–166, 1998.
[13] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter, "Bayesian optimization with robust bayesian neural networks," in Advances in Neural Information Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, 2016.
[14] T. Wang et al., "Random forest-bayesian optimization for product quality prediction with large-scale dimensions in process industrial cyber–physical systems," IEEE Internet of Things Journal, vol. 7, no. 9, pp. 8641–8653, 2020.
[15] T. O. Project, "OpenROAD-flow-scripts: Scripts for the OpenROAD flow," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts, 2023.
[16] L. T. Clark et al., "ASAP7: A 7-nm finFET predictive process design kit," Microelectronics Journal, 2016.
[17] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.
[18] M. Shoemate et al., "OpenDP Library." [Online]. Available: https://github.com/opendp/opendp
[19] D. U. Lim and H. Park, "Graph neural network-based detailed placement optimization framework," in 2024 25th International Symposium on Quality Electronic Design (ISQED), 2024, pp. 1–6.