Circuit Area Prediction in Logic Synthesis: Domain-Specific Loss Guided Graph Neural Networks

Adarsh Gupta, Aditya Kumar, Parv Agarwal
220101003, 220101005, 220101124
Department of Computer Science
IIT Guwahati

***Abstract*—Logic synthesis optimization aims to minimize circuit area while preserving functionality, but predicting the impact of synthesis recipes on different circuit designs remains challenging. This paper introduces a hybrid approach combining Graph Neural Networks (GNNs) and Recurrent Neural Networks (RNNs) to predict circuit area throughout the synthesis process. We represent circuits as directed graphs and synthesis recipes as sequences of commands, extracting meaningful features with specialized attention mechanisms. Our approach is enhanced with domain-specific loss functions that emphasize relative area reduction, critical optimization steps, attention interpretability, and meaningful circuit representations. Experimental results demonstrate strong predictive performance, with high accuracy in tracking area changes across synthesis steps. The model provides uncertainty estimates, enabling designers to make informed decisions about optimization strategies. This approach facilitates efficient design space exploration without running expensive synthesis tools for each candidate recipe.**

***Index Terms*—Logic Synthesis, Electronic Design Automation, Graph Neural Networks, Machine Learning for EDA, Area Prediction**

I. Introduction and Problem Statement

Electronic design automation (EDA) tools employ a variety of synthesis and optimization techniques to transform high-level circuit descriptions into optimized gate-level implementations. In the realm of logic synthesis, area optimization is a critical objective that seeks to minimize the number of gates required to implement a circuit while preserving its functionality. This optimization process typically involves applying a sequence of transformation commands (a "recipe") to the circuit, with each command potentially reducing the circuit area.

The efficacy of synthesis recipes varies significantly across different circuit designs, making it challenging to predict the outcome of a particular optimization sequence without running the full synthesis flow. This leads to inefficient design space exploration, as designers must execute numerous synthesis runs to identify effective optimization strategies, consuming valuable time and computational resources.

A. Problem Formulation

We define a circuit design C as a directed acyclic graph (DAG) $G = (V, E)$, where V represents the set of nodes (gates and I/O pins) and E represents the connections between them. Each node $v_i \in V$ is associated with a type attribute $\tau(v_i) \in \{INPUT, OUTPUT, AND, NOT\}$ and other relevant features such as level (topological depth), fan-in, and fan-out.

A synthesis recipe R is defined as an ordered sequence of transformation commands $R = (r_1, r_2, ..., r_n)$, where each command $r_i \in \mathcal{C}$ belongs to a predefined set of available commands $\mathcal{C} = \{$ "rewrite -z", "rewrite -l", "rewrite", "balance", "resub", "refactor", "resub -z", "refactor -z" $\}$.

Let $A(C, R, i)$ represent the area (number of AND gates) of circuit C after applying the first i commands of recipe R. Our objective is to develop a predictive model f_θ with parameters θ such that:

$$\hat{A}(C, R, i) = f_\theta(C, R, i) \tag{1}$$

where $\hat{A}(C, R, i)$ approximates the true area $A(C, R, i)$ for all $i \in \{1, 2, ..., n\}$.

The prediction task can then be formulated as an optimization problem:

$$\theta^* = \arg\min_\theta \mathcal{L}(f_\theta(C, R), A(C, R)) \tag{2}$$

where $\mathcal{L}$ is a suitable loss function comparing the predicted area sequence

$$\hat{A}(C, R) = (\hat{A}(C, R, 1), \hat{A}(C, R, 2), ..., \hat{A}(C, R, n))$$

with the ground truth area sequence

$$A(C, R) = (A(C, R, 1), A(C, R, 2), ..., A(C, R, n))$$

.

II. Overview of Proposed Methodology

We propose a hybrid architecture that combines Graph Neural Networks (GNNs) and Recurrent Neural Networks (RNNs) to address the circuit area prediction problem. Our approach leverages the representational power of GNNs to capture the structural properties of circuit designs and the sequential modeling capabilities of RNNs to process synthesis recipes.

The overall framework consists of three key components:

1) **Circuit Embedding Module**: A GNN-based component that transforms the circuit graph into a fixed-dimensional embedding
2) **Recipe Processing Module**: An RNN-based component that processes the sequence of synthesis commands
3) **Area Prediction Module**: A prediction component that combines circuit and recipe information to estimate area after each synthesis step

Additionally, we introduce domain-specific loss functions that incorporate circuit optimization knowledge to guide the learning process and improve prediction accuracy. These specialized loss functions emphasize relative area reduction, critical optimization steps, attention interpretability, and meaningful circuit representations.

The proposed approach not only predicts the final area after applying the complete recipe but also provides a step-by-step prediction of area changes throughout the optimization process, offering valuable insights into the effectiveness of each synthesis command.

III. Detailed Methodology

A. Circuit Representation and Embedding

1) Graph Construction: Given a circuit description in the bench format, we construct a directed graph $G = (V, E)$ where each node represents a gate or an I/O pin, and edges represent connections between them. For each node $v_i \in V$, we extract a feature vector $\mathbf{x}_i$ that encodes:

- Gate type (one-hot encoded):

$$\mathbf{t}_i \in \{0,1\}^4$$

for

$$\{INPUT, OUTPUT, AND, NOT\}$$

- Level: $l_i \in \mathbb{Z}^+$
- Fan-in count: $f_i^{in} \in \mathbb{Z}^+$
- Fan-out count: $f_i^{out} \in \mathbb{Z}^+$

The feature vector for node v_i is then defined as:

$$\mathbf{x}_i = [\mathbf{t}_i; l_i; f_i^{in}; f_i^{out}] \in \mathbb{R}^7 \tag{3}$$

2) Graph Neural Network Architecture: To learn a fixed-dimensional representation of the circuit, we employ a multi-layer Graph Convolutional Network (GCN). Each GCN layer updates node representations by aggregating information from neighboring nodes:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{W}^{(l)} \sum_{j \in \mathcal{N}(i)} \frac{\mathbf{h}_j^{(l)}}{\sqrt{|\mathcal{N}(i)|} \cdot \sqrt{|\mathcal{N}(j)|}} + \mathbf{b}^{(l)}\right) \tag{4}$$

where $\mathbf{h}_i^{(l)}$ is the feature vector of node v_i at layer l, $\mathcal{N}(i)$ is the set of neighboring nodes of v_i, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are learnable parameters, and σ is a non-linear activation function (ReLU).

To enhance the model's ability to focus on critical circuit structures, we incorporate a graph attention mechanism:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]\right) \tag{5}$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})} \tag{6}$$

$$\mathbf{h}_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right) \tag{7}$$

where $\|$ denotes concatenation, $\mathbf{a}$ is a learnable attention vector, and α_{ij} represents the attention coefficient between nodes i and j.

The final circuit embedding is obtained by applying a global pooling operation over all node representations:

$$\mathbf{z}_C = \frac{1}{|V|} \sum_{i=1}^{|V|} \mathbf{h}_i^{(L)} \tag{8}$$

where L is the number of GCN layers.

Motivation: The choice of GNNs for circuit representation is motivated by their inherent ability to capture both local connectivity patterns and global structural properties of circuits. Traditional feature-based approaches fail to encode the complex relationships between gates, whereas GNNs naturally model these dependencies through message passing. The incorporation of attention mechanisms allows the model to identify critical paths and gates that significantly impact area optimization, mirroring how experienced designers focus on specific circuit structures.

B. Recipe Encoding and Processing

1) Command Encoding: Each synthesis command $r_i \in \mathcal{C}$ is encoded using a one-hot encoding scheme, resulting in an 8-dimensional vector $\mathbf{r}_i \in \{0,1\}^8$ (since $|\mathcal{C}| = 8$). A recipe $R = (r_1, r_2, ..., r_n)$ is thus represented as a sequence of one-hot encoded vectors $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n) \in \{0,1\}^{n \times 8}$.

2) Sequential Recipe Processing: To capture the sequential dependencies between synthesis commands, we employ a bidirectional LSTM network:

$$\overrightarrow{\mathbf{h}}_t = \text{LSTM}_{\text{forward}}(\mathbf{r}_t, \overrightarrow{\mathbf{h}}_{t-1}) \tag{9}$$

$$\overleftarrow{\mathbf{h}}_t = \text{LSTM}_{\text{backward}}(\mathbf{r}_t, \overleftarrow{\mathbf{h}}_{t+1}) \tag{10}$$

$$\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t \| \overleftarrow{\mathbf{h}}_t] \tag{11}$$

where $\overrightarrow{\mathbf{h}}_t$ and $\overleftarrow{\mathbf{h}}_t$ represent the forward and backward hidden states at time step t, and $\mathbf{h}_t$ is the concatenated bidirectional representation.

To further enhance the model's ability to capture dependencies between different optimization steps, we incorporate a self-attention mechanism:

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \mathbf{K} = \mathbf{H}\mathbf{W}_K, \mathbf{V} = \mathbf{H}\mathbf{W}_V \tag{12}$$

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \tag{13}$$

$$\mathbf{H}' = \mathbf{A}\mathbf{V} \tag{14}$$

where $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n]^T$ is the matrix of hidden states, $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are learnable parameter matrices, d_k is the dimension of the keys, and $\mathbf{H}'$ is the attention-enhanced representation.

Motivation: The sequential nature of synthesis recipes necessitates a model that can capture both the individual effects of commands and their interactions. Bidirectional LSTMs allow the model to consider both past and future commands when processing each step, recognizing that the effectiveness of a command often depends on the entire optimization context. The self-attention mechanism enables the model to discover long-range dependencies between non-adjacent commands, capturing complex optimization patterns that might span across multiple steps.

C. Area Prediction Module

1) Stepwise Area Prediction: For each synthesis step i, we combine the circuit embedding $\mathbf{z}_C$ with the corresponding recipe state $\mathbf{h}_i$ to predict the area after applying the first i commands:

$$\mathbf{u}_i = [\mathbf{z}_C \| \mathbf{h}_i] \tag{15}$$

$$\mathbf{v}_i = \text{ReLU}(\mathbf{W}_1\mathbf{u}_i + \mathbf{b}_1) \tag{16}$$

$$\hat{A}(C, R, i) = \mathbf{W}_2\mathbf{v}_i + \mathbf{b}_2 \tag{17}$$

where $\mathbf{W}_1$, $\mathbf{b}_1$, $\mathbf{W}_2$, and $\mathbf{b}_2$ are learnable parameters.

2) Uncertainty Estimation: To quantify the model's confidence in its predictions, we estimate the uncertainty for each predicted area:

$$\sigma^2(C, R, i) = \text{softplus}(\mathbf{W}_\sigma\mathbf{v}_i + \mathbf{b}_\sigma) \tag{18}$$

where $\mathbf{W}_\sigma$ and $\mathbf{b}_\sigma$ are learnable parameters, and softplus ensures positive uncertainty values.

Motivation: The area prediction module is designed to capture how circuit properties interact with synthesis commands to influence area reduction. By predicting area after each step, rather than just the final result, the model provides valuable insights into the progression of optimization. The uncertainty estimation acknowledges that prediction difficulty varies across different circuit-recipe combinations, allowing designers to assess the reliability of predictions and make informed decisions.

D. Enhanced Loss Functions

To improve model performance and incorporate domain knowledge, we developed specialized loss functions that address various aspects of circuit optimization prediction.

1) Relative Area Reduction Loss: This loss function focuses on the relative changes in area rather than absolute values after each optimisation step:

$$\Delta\hat{A}_i = \frac{\hat{A}(C, R, i+1) - \hat{A}(C, R, i)}{\hat{A}(C, R, i) + \epsilon} \tag{19}$$

$$\Delta A_i = \frac{A(C, R, i+1) - A(C, R, i)}{A(C, R, i) + \epsilon} \tag{20}$$

$$\mathcal{L}_{rel} = \frac{1}{n-1}\sum_{i=1}^{n-1}\left(\Delta\hat{A}_i - \Delta A_i\right)^2 \tag{21}$$

where ϵ is a small constant added for numerical stability.

Motivation: In circuit optimization, percentage reductions are often more meaningful than absolute changes, especially when comparing across circuits of different sizes. This loss encourages the model to accurately predict the relative impact of each optimization step, aligning with how designers typically evaluate optimization quality.

2) Critical Step Emphasis Loss: This loss function places greater emphasis on accurately predicting steps that cause significant area changes:

$$E_{i+1} = \left|\hat{A}(C, R, i+1) - A(C, R, i+1)\right| \tag{22}$$

$$\Delta A_i = |A(C, R, i+1) - A(C, R, i)| \tag{23}$$

$$\mathcal{L}_{crit} = \frac{1}{n-1}\sum_{i=1}^{n-1} E_{i+1} \cdot (1 + \Delta A_i) \tag{24}$$

Motivation: Not all synthesis steps have equal impact on area reduction. Steps that significantly reduce area are particularly valuable to predict accurately, as they represent key optimization opportunities. This loss function guides the model to pay special attention to high-impact steps, which are the most relevant for designers seeking efficient optimization strategies.

3) Sequence Dependency Loss: This loss function models how the effectiveness of synthesis commands depends on the sequence of steps that came before:

$$\mathcal{L}_{seq} = \frac{1}{n}\sum_{i=0}^{n-1}\widetilde{\mathcal{E}}_i \tag{25}$$

where $\widetilde{\mathcal{E}}_i$ is the weighted prediction error at step i, computed as:

$$\widetilde{\mathcal{E}}_i = \begin{cases} \mathcal{E}_i & \text{if } i = 0 \\ \mathcal{E}_i \cdot (1 + \mathcal{D}_i) & \text{if } i > 0 \end{cases} \tag{26}$$

Here, $\mathcal{E}_i = (\hat{A}(C, R, i) - A(C, R, i))^2$ is the squared prediction error at step i.

The dependency factor $\mathcal{D}_i$ captures the influence of previous steps on the current step i:

$$\mathcal{D}_i = \sum_{j=0}^{i-1} S_{ij} \cdot \mathcal{E}_j \tag{27}$$

The similarity coefficient S_{ij} between steps i and j is computed from the recipe embeddings:

$$S_{ij} = \langle \mathbf{h}_i, \mathbf{h}_j \rangle \tag{28}$$

where $\mathbf{h}_i$ is the embedding of step i, and $\langle \cdot, \cdot \rangle$ denotes the dot product.

Motivation: This loss function captures a fundamental aspect of circuit optimization: the effect of a synthesis command at step i often depends strongly on what commands were applied before it. By computing similarities between command embeddings, the loss identifies which previous steps are conceptually related to the current step. When the model makes errors on steps that are similar to the current step, the penalty for errors at the current step is increased. This encourages the model to learn patterns such as "command A is less effective after command B" or "commands C and D work well in sequence." The multiplicative factor $(1 + \mathcal{D}_i)$ ensures that predictions for steps with strong dependencies on previous steps are penalized more heavily when those dependencies aren't properly modeled. This loss enables the model to capture the sequential nature of synthesis optimization, where the effectiveness of each command is contextual rather than absolute.

4) Attention Entropy Loss: This loss function encourages the attention mechanism to focus more sharply on relevant information:

$$\mathcal{L}_{attn} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} \log(\alpha_{ij} + \epsilon) \tag{29}$$

where α_{ij} represents the attention weight from step i to step j.

Motivation: Diffuse attention patterns can make the model's decision-making process difficult to interpret. By encouraging more focused attention, this loss function improves the interpretability of the model's predictions, helping designers understand which synthesis steps influence others and why certain predictions are made.

5) Feature Consistency Loss: This loss function ensures that circuits with similar optimization behavior have similar embeddings:

$$d_{ij}^{emb} = \|\mathbf{z}_{C_i} - \mathbf{z}_{C_j}\|_2 \tag{30}$$

$$d_{ij}^{area} = \frac{\|A(C_i, R_i) - A(C_j, R_j)\|_2}{\max_{k,l} \|A(C_k, R_k) - A(C_l, R_l)\|_2 + \epsilon} \tag{31}$$

$$\mathcal{L}_{feat} = \frac{1}{B(B-1)} \sum_{i=1}^{B} \sum_{j \neq i}^{B} \left(d_{ij}^{emb} - d_{ij}^{area}\right)^2 \tag{32}$$

where B is the batch size, and d_{ij}^{emb} and d_{ij}^{area} are the distances between circuit embeddings and area profiles, respectively.

Motivation: This loss function helps create a semantically meaningful embedding space where proximity indicates similar optimization behavior. Such a space facilitates better generalization to new circuits, as the model can relate them to similar previously seen examples. This is particularly valuable in EDA applications, where knowledge transfer between related circuits is essential for efficient optimization.

6) Combined Loss Function: The final loss function combines these specialized components with traditional MSE losses:

$$\begin{aligned} \mathcal{L} = \lambda_1 \cdot \text{MSE}(\hat{A}(C, R, n), A(C, R, n)) \\ + \lambda_2 \cdot \frac{1}{n} \sum_{i=1}^{n} \text{MSE}(\hat{A}(C, R, i), A(C, R, i)) \\ + \lambda_3 \cdot \mathcal{L}_{rel} + \lambda_4 \cdot \mathcal{L}_{crit} + \lambda_5 \cdot \mathcal{L}_{seq} \\ + \lambda_6 \cdot \mathcal{L}_{attn} + \lambda_7 \cdot \mathcal{L}_{feat} + \lambda_8 \cdot \mathcal{L}_{NLL} \end{aligned} \tag{33}$$

where $\mathcal{L}_{NLL}$ is the negative log-likelihood loss for uncertainty estimation:

$$\epsilon_i = (\hat{A}(C, R, i) - A(C, R, i))^2 \tag{34}$$

$$\sigma_i^2 = \sigma^2(C, R, i) \tag{35}$$

$$\mathcal{L}_{NLL} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{2} \log \sigma_i^2 + \frac{\epsilon_i}{2\sigma_i^2} \right) \tag{36}$$

and $\lambda_1, \lambda_2, ..., \lambda_8$ are hyperparameters that control the contribution of each loss component.

IV. Experimental Results

A. Dataset and Experimental Setup

We evaluated our approach on a dataset of circuit designs, each associated with synthesis recipes of up to 20 steps and the corresponding AND gate counts after each step. Each circuit is represented in the bench format, providing structural information that we convert into graph representations.

Our dataset contains the following columns:

- `design_name`: Unique identifier for each circuit
- `Step1` to `Step20`: Synthesis commands applied at each step
- `AND1` to `AND20`: AND gate counts after each synthesis step

The dataset was divided into training (70%), validation (15%), and test (15%) sets, ensuring a representative distribution of circuit types and sizes.

We utilize a 3-layer Graph Neural Network (GNN) with 128 hidden units per layer and residual connections, alongside a 2-layer bidirectional LSTM, also with 128 hidden units per layer. A 4-head self-attention mechanism is employed for capturing dependency relationships. The node features are represented as a 7-dimensional vector, which includes a gate type one-hot encoding, level, fan-in, and fan-out information, while the recipe is encoded using 8-dimensional one-hot vectors. For training, the Adam optimizer is used with a learning rate of 0.001 and a weight decay of 1e-5, processing data in batches of 16. A Cosine Annealing Learning Rate Scheduler is applied to adjust the learning rate over the training epochs, and early stopping is implemented with a patience of 20 epochs. The loss function is balanced with weights set to λ_1 = 1.0, λ_2 = 1.0, λ_3 = 1.0, λ_4 = 0.3, λ_5 = 0.2, λ_6 = 0.1, λ_7 = 0.2, and λ_8 = 0.3.