How Deep Learning Can Drive Physical Synthesis Towards More Predictable Legalization

Renan Netto, Sheiny Fabre, Tiago Fontana, Vinicius Livramento, Laércio Lima Pilla, José Güntzel

▶ **To cite this version:**

Renan Netto, Sheiny Fabre, Tiago Fontana, Vinicius Livramento, Laércio Lima Pilla, et al.. How Deep Learning Can Drive Physical Synthesis Towards More Predictable Legalization. International Symposium on Physical Design, Apr 2019, San Francisco, United States. 10.1145/3299902.3309754 . hal-02057042

HAL Id: hal-02057042

https://hal.archives-ouvertes.fr/hal-02057042

Submitted on 5 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

How Deep Learning Can Drive Physical Synthesis Towards More Predictable Legalization

Renan Netto[1], Sheiny Fabre[1], Tiago Augusto Fontana[1], Vinicius Livramento[2], Laércio Pilla[3],
José Luís Güntzel[1]
[1]Embedded Computing Lab, PPGCC, Federal University of Santa Catarina, Brazil
[2]ASML, Netherlands
[3]LRI, Univ. Paris-Sud — CNRS, France
{renan.netto,sheiny.fabre,tiago.fontana}@posgrad.ufsc.br

ABSTRACT

Machine learning has been used to improve the predictability of different physical design problems, such as timing, clock tree synthesis and routing, but not for legalization. Predicting the outcome of legalization can be helpful to guide incremental placement and circuit partitioning, speeding up those algorithms. In this work we extract histograms of features and snapshots of the circuit from several regions in a way that the model can be trained independently from region size. Then, we evaluate how traditional and convolutional deep learning models use this set of features to predict the quality of a legalization algorithm without having to executing it. When evaluating the models with holdout cross validation, the best model achieves an accuracy of 80% and an F-score of at least 0.7. Finally, we used the best model to prune partitions with large displacement in a circuit partitioning strategy. Experimental results in circuits (with up to millions of cells) showed that the pruning strategy improved the maximum displacement of the legalized solution by 5% to 94%. In addition, using the machine learning model avoided from 22% to 99% of the calls to the legalization algorithm, which speeds up the pruning process by up to 3×.

KEYWORDS

Physical synthesis, placement, legalization, machine learning

1 INTRODUCTION

The high flexibility provided by machine learning (ML) models allows their use to predict the outcome of physical design algorithms. They have been employed so far to help choose between different clock tree synthesis algorithms [11], to fix miscorrelations between different timing engines [7], and to identify detailed routing violations during the placement stage [2, 17, 19]. The benefits of ML models come from their ability to improve the quality of physical design algorithms by predicting information that would otherwise be too costly to evaluate during execution.

Machine learning techniques have yet to be employed to help predict the outcome of legalization algorithms. As technology nodes advance, new challenges affect modern legalization algorithms. Some of these challenges include pin accessibility, usage of multi-row cell libraries, complex design rules, physical floorplan complexity, as well as tight performance and power constraints. In addition, modern legalization algorithms have to keep a low circuit displacement to avoid degrading the solution of upstream steps.

The prediction of the outcome of legalization algorithms by ML techniques has multiple applications: (1) choosing, among multiple legalization algorithms, the one that will result in the lowest displacement for a given legalization region, similar to what has been done for clock tree synthesis [11]; (2) guiding an incremental placement technique. The ML models could predict which cell movements result in the greatest improvement on different metrics, without requiring the execution of the legalization algorithm itself; (3) guiding a circuit partitioning strategy. Circuit partitioning can be used to decompose the circuit in smaller disjoint parts, which can reduce the execution time of legalization algorithms by more than one order of magnitude. However, it can also degrade some quality metrics since it reduces the solution space available to the legalization algorithm.

In this work we explore mainly option (3) by integrating the proposed machine learning model into a circuit partitioning strategy to avoid partitions that result in large displacement. Furthermore, we partially explore option (2), because the proposed model can be used to predict when some optimizations will largely degrade the solution obtained by upstream steps. We do so by training different ML models to detect when the maximum displacement of a given partition exceeds a specified threshold. Then, we select the model with the best results to be integrated in the partitioning strategy, acting as a pruning mechanism. The main contributions of this paper are:

- We propose a feature extraction strategy for training machine learning models using the information of circuit partitions as input. This set of features is independent of the partition size.
- To the best of our knowledge, this is the first work to use machine learning models to help a legalization algorithm. We evaluate different ML models in order to select the best one for this problem (which achieved an accuracy of 80% and an F-score ≥0.7).
- We employed the best ML model as a pruning mechanism for a circuit partitioning strategy. Results using circuits from both ICCAD 2017 and ICCAD 2015 CAD Contests show that the pruning strategy reduced the maximum displacement of those circuits by up to 94%, and the use of ML accelerated the pruning by up to 3×.

The remaining sections are organized as follows. Section 2 displays the related work on ML models used for physical design, and multi-row legalization. Sections 3 and 4 present the proposed ML methodology and its integration into a legalization algorithm.

Finally, Section 5 shows the experimental results and Section 6 provides concluding remarks.

2 RELATED WORK

Machine learning techniques have been used to solve different physical design problems. The work from [11] trains a regression model to predict the outcome of different clock tree synthesis engines. The authors extract features from architectural, floorplanning and design parameters. In order to handle a large number of linearly correlated parameters, they separate the features in two models, which are trained separately and combined using linear regression. Another application is predicting the outcome of golden signoff timing engines [7], where the authors propose a regression model to correct miscorrelations between two commercial signoff tools. For that, they extract multiple features concerning capacitance, resistance and delay of cells and wires.

Recently, machine learning techniques have been used to predict the violation of routability constraints in a placed netlist. For example, the work from [19] extracts features regarding pin distribution, routing blockage, global routes and local nets in order to predict the number of DRC violations in a placed area. The work from [2] improves this idea by predicting the actual locations of the DRC violations, using a different set of features. Finally, the work from [17] focuses on predicting only the existence of detailed routing short violations, so that this information can be used in a detailed placement flow, for example. Although different works make use of machine learning techniques, none of them aim to predict the quality of legalization algorithms, which is the focus of this work. Since legalization is performed not only after the global placement, but also after other placement optimization techniques, improving the legalization solution consequently improves the quality of those optimizations.

Several recent works have been proposed to legalize circuits from advanced technology nodes, which contain multi-row cells. For example, the works from [4] and [18] propose algorithms that legalize cells one at a time. Their algorithms enumerate a set of valid insertion points for each cell and place each in the location that minimizes circuit displacement. While the algorithm from [4] uses a greedy heuristic, the algorithm from [18] adapts the Abacus legalization algorithm [16] that uses dynamic programming.

Instead of handling each cell at a time, the works from [10] and [3] propose algorithms that simultaneously legalize multiple cells. The authors of [10] propose an Integer Linear Programming (ILP) model to solve the multi-row legalization problem. Due to the high complexity of the ILP model, they divide the circuit into bins, solving the problem for multiple bins in parallel. Such strategy leads to better results, but at the cost of longer run times. The authors of [3], by their turn, relax some constraints of the problem in order to model it as a Linear Complementarity Problem. This way, they can still achieve better results than the previous works, but with a more reasonable run time. Finally, the work of [15] improves over [4] and considers additional metrics for legalization problem, such as routability constraints.

A machine learning model that predicts the outcome of such legalization algorithms can be used to guide incremental optimization techniques or partitioning strategies. For example, the work of [6] proposes a circuit partitioning strategy to speed up legalization by executing the legalization algorithm in smaller regions of the circuit. This way, the authors sped up the legalization, but at the cost of maximum displacement degradation, since the partitioning reduces the solution space of the legalization algorithm. Therefore, in this work we also integrate the proposed machine learning model to this partitioning strategy, so that the model guides this process to avoid partitions with large displacement.

3 MACHINE LEARNING METHODOLOGY

We model the legalization outcome prediction as a binary classification problem. Given a legalization algorithm $\mathcal{L}$, a partition of cells p_i, and a displacement threshold Δ, the output of the ML model is a binary variable $y \in \{0, 1\}$, indicating whether $\mathcal{L}$ legalizes the cells of p_i without any cell displacement exceeding Δ[1].

In this work, instead of using $\mathcal{L}$ and Δ as inputs of the ML model, we train models for a specific combination of $\mathcal{L}$ and Δ. As consequence, the model receives as input the rectangular region (given by $R(p_i)$) and cells of a partition (given by $C(p_i)$), and must predict if the legalization algorithm $\mathcal{L}$ used to train the model is able to legalize this partition under a maximum displacement threshold of Δ. In order to do so, we must provide a large number of samples to train the ML model, so that it can identify which patterns lead to a partition with a large displacement.

3.1 Training data generation

Algorithm 1 shows how we generate the data for training and validation of the ML model. Given a set of cells $C = \{c_1, c_2, ..., c_{n-1}, c_n\}$, a legalization algorithm $\mathcal{L}$ and the rectangular area of the circuit $R = (X_{left}, X_{right}, Y_{top}, Y_{bottom})$, the algorithm aims to generate data from partitions of different sizes. The first step consists in defining the number of samples that will be generated for each partition size (line 1). By generating the same number of samples for each size (1024 in this work), we avoid having the ML model become biased to a specific partition size. In addition, increasing the number of samples acts as a data augmentation strategy, which generates more samples from the same circuit and helps avoiding overfitting.

In order to generate the circuit partitions, we used the strategy from [6], which partitions the circuit using a k-d tree data structure, where each leaf node represents a partition. The strategy receives as input a desired height for the k-d tree and generates 2^{height} partitions. Therefore, Algorithm 1 iterates over different values for the tree height (line 3), and the necessary number of iterations to generate the desired number of samples is calculated in line 4. We limit the maximum height of the k-d tree to 9 because, as the height increases, the partitions become smaller. A height higher than 9 would result in partitions with less than a few hundred of cells, which would be too small.

For each iteration, a new circuit partitioning is generated (line 7). To ensure that the partitions are different from each other, we apply a vector of random movements to the cells in C (line 6). The $MOVE\_CELLS$ function in Algorithm 2 is responsible for applying

[1] The displacement of a cell c_i is given by the Manhattan distance between its legalized location $l(c_i) = (x(c_i), y(c_i))$ and its location before legalization $l'(c_i) = (x'(c_i), y'(c_i))$.

Algorithm 1: GENERATE_DATA(C, $\mathcal{L}$, R)

```
1  $n\_samples \leftarrow 1024$;
2  $max\_height \leftarrow 9$; $\mathcal{F} \leftarrow \emptyset$;
3  for $height \leftarrow 1$ to $max\_height$ do
4      $n\_iterations \leftarrow \frac{n\_samples}{2^{height}}$;
5      for $num\_it \leftarrow 1$ to $n\_iterations$ do
6          MOVE_CELLS($C$, $R$);
7          $P \leftarrow$ CIRCUIT_PARTITIONING($C$, $height$);
8          foreach $p_i \in P$ do
9              $\Lambda$, $result \leftarrow \mathcal{L}(C(p_i), R(p_i))$;
10             $\mathcal{F} \leftarrow \mathcal{F} \cup$ GET_FEATURES($C(p_i)$, $R(p_i)$, $\Lambda$, $result$, $\Delta$);
11         end
12     end
13 end
14 SAVE_DATA($\mathcal{F}$);
```

Algorithm 2: MOVE_CELLS(C, R)

```
1  foreach $c_i \in C$ do
2      $r_x \leftarrow RANDOM(-10000, 10000)$;
3      $r_y \leftarrow RANDOM(-10000, 10000)$;
4      $x(c_i) \leftarrow x'(c_i) + r_x$;
5      $y(c_i) \leftarrow y'(c_i) + r_y$;
6      $x(c_i) \leftarrow min(X_{right} - w(c_i), max(X_{left}, x(c_i)))$;
7      $y(c_i) \leftarrow min(Y_{top} - h(c_i), max(Y_{bottom}, y(c_i)))$;
8  end
```

this movement. For each cell, two random variables $r_x, r_y \in \mathbb{Z}$ are generated representing the cell movement on x and y coordinates, respectively. We generate them using a uniform integer distribution in the range [-10000, 10000] to ensure that the movement is large enough to generate significantly different placements[2]. After moving the cells, we make sure that they lie within the circuit boundaries R (lines 6–7). Observe that the $MOVE\_CELLS$ function is called before partitioning the circuit, so the amount of movement is not dependent of the partition size.

After generating the set of partitions P, Algorithm 1 legalizes each partition (lines 8-11). Observe that it legalizes only the cells in $C(p_i)$ inside the partition and the legalization area is limited to the partition area $R(p_i)$. In addition, the legalization function from line 9 does not actually move the cells, it only finds legal locations for the cells and returns those locations (denoted by Λ). It also returns a boolean variable indicating if the legalization was successful or not (denoted by $result$), as the legalization may fail for a given partition. Observe that the proposed ML methodology is independent from a specific legalization algorithm, as long as the algorithm may be executed for a subset of the circuit cells and for a subregion of the circuit area. For this reason, we do not present the pseudocode of the legalization algorithm, but we used an adaptation of the Abacus legalization algorithm from [16] to handle multi-row cells.

3.2 Feature selection

Given the legalization result and legal locations, the next step is obtaining the features for this partition. In the end, when all features are collected, the data is saved in an output file in line 14. In this work we evaluate traditional and convolutional neural network models, so we need features for both of them. For convolutional models, the input is simply a snapshot image of the partition. We used different colors to distinguish between movable and fixed cells. Movable cells are represented by shades of the same color. This way, the model can identify overlaps since they are represented by the combined colors of multiple cells. On the other hand, for the traditional models we need to select an appropriate set of features. Partitions become hard to legalize with low displacement when they have a high density of cells, or when they have many cell overlaps. Therefore, we selected the features presented in Table 1 for our ML methodology.

Table 1: Features used by the ML model.

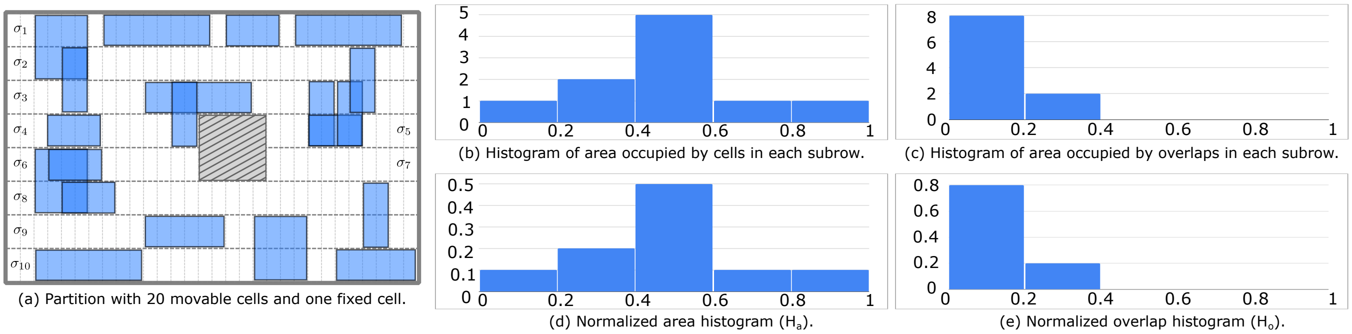
Feature	Meaning
D	Density of the partition area
A_h	Area occupied by cells of each height
H_a	Normalized histogram of area occupied by cells on each partition subrow
H_o	Normalized histogram of area occupied by overlaps on each partition subrow

Algorithm 3: GET_FEATURES($C(p_i)$, $R(p_i)$, Λ, $result$, Δ)

```
$A_f \leftarrow 0$; $A_m \leftarrow 0$; $A_r \leftarrow w(p_i) \times h(p_i)$; $A_h \leftarrow []$;
$\delta max \leftarrow -\infty$;
foreach $c_i \in C(p_i)$ do
    if $fixed(c_i)$ then
        $box \leftarrow intersection(c_i, R(p_i))$;
        $A_f \leftarrow A_f + w(box) \times h(box)$;
    else
        $A_m \leftarrow A_m + w(c_i) \times h(c_i)$;
        $A_h[h(c_i)] \leftarrow A_h[h(c_i)] + \frac{w(c_i) \times h(c_i)}{A_r}$;
    end
    $\delta(c_i) \leftarrow |l(c_i) - \lambda(c_i)|$;
    $\delta max \leftarrow \max(\delta max, \delta(c_i))$;
end
$\Delta \leftarrow \frac{A_m}{A_r - A_f}$;
$y \leftarrow \delta max \leq \Delta$;
$A \leftarrow []$; $O \leftarrow []$; $\Sigma \leftarrow$ partition subrows;
foreach $\sigma_i \in \Sigma$ do
    $C(\sigma_i) \leftarrow$ cells intersecting $\sigma_i$;
    $a(\sigma_i) \leftarrow 0$; $o(\sigma_i) \leftarrow 0$;
    foreach $c_i \in C(\sigma_i)$ do
        $a(\sigma_i) \leftarrow a(\sigma_i) + w(c_i)$;
    end
    $C' \leftarrow$ cells intersecting in $C(\sigma_i)$;
    foreach $c_j \in C'$ do
        $o(\sigma_i) \leftarrow o(\sigma_i) + w(c_j)$;
    end
    $A[\sigma_i] \leftarrow \frac{a(\sigma_i)}{w(\sigma_i)}$; $O[\sigma_i] \leftarrow \frac{o(\sigma_i)}{w(\sigma_i)}$;
end
$H_a \leftarrow$ normalized histogram with values of $A$;
$H_o \leftarrow$ normalized histogram with values of $O$;
return $(y, \Delta, A_h, H_a, H_o)$;
```

The first two features aim to represent global information of the partition density. Since partitions with multi-row cells are harder to legalize, we measure not only the cell density, but also the area occupied by cells of each height. However, just the global information is not enough to identify partitions that are hard to legalize, since some circuit rows may be more crowded than others. The third feature aims to represent this information by means of an histogram of the area occupied by cells on each partition subrow. A subrow is a row segment that does not overlap with any macroblocks and/or

[2] The smallest circuit used in the experiments has an area of 342000 × 342000, so this movement corresponds to at most 3% of the circuit dimensions.



(a) Partition with 20 movable cells and one fixed cell.

(b) Histogram of area occupied by cells in each subrow.

(c) Histogram of area occupied by overlaps in each subrow.

(d) Normalized area histogram (H_a).

(e) Normalized overlap histogram (H_o).

Figure 1: Circuit features extracted for a hypothetical partition.

fixed cells. Finally, a subrow may be overcrowded but with few cell overlaps, which makes it easier to legalize. Therefore, the last feature measures the area occupied by overlaps on each subrow, so that the ML model can correlate this information to the result.

Figure 1 shows by means of an example how the histograms for the last two features are generated for a given partition. Figure 1(a) illustrates a partition with 20 movable cells (blue rectangles), one fixed cell (gray rectangle) and 10 subrows (σ_1 to σ_{10}). Figures 1(b) and (c) show the area and overlap histograms for this partition. For illustration purposes, the histograms contain 5 bins, but we actually used histograms with 20 bins for better precision. Each bin indicates the number of rows whose area of cells (or overlap of cells) is within a given range. We can see that the histogram in Figure 1(b) concentrates on the middle region, with most subrows having from 40% to 60% of their area occupied by cells. On the other hand, the histogram in Figure 1(c) concentrates on the 0% to 20% bin, with only two subrows having more than 20% of their area occupied by overlaps. However, using absolute values for the y axis of those histograms may bias the ML model to large partitions, since they would have a larger number of subrows. In order to avoid this issue, we normalize the y axis with relation to the number of subrows in the partition, resulting in the histograms H_a and H_o in Figures 1(d) and (e). Observe that they have the same structure of the previous histograms, with the only difference being the normalized vertical axis.

Algorithm 3 shows the details of how these features are collected from a given partition. The loop from lines 3–13 calculates the values for the first two features (density and area occupied by cells of each height), so it starts by initializing the necessary variables to do so. Those variables are: the area occupied by fixed cells (A_f), the area occupied by movable cells (A_m), the area of the partition itself (A_r) and a list with the areas occupied by cells of each height (A_h). Observe that the area occupied by fixed cells is not considered in A_h, since those cells are not moved by the legalization algorithm. However, their area is important to measure the partition density, which is given by the area occupied by movable cells A_m divided by the free area in the partition (A_r - A_f) in line 14. In addition, for each height, A_h is normalized by the partition area. The first loop also measures the maximum displacement of the legalized cells in lines 11–12. This information is used to determine the class of this partition in line 15.

The loop from lines 17–29, on the other hand, is responsible for obtaining the data for the histograms H_a and H_o. This is done by iterating through all subrows $\sigma_i \in \Sigma$ of the partition and, for each one, measuring the width occupied by the cells intersecting the area of σ_i, storing this information in variable $a(\sigma_i)$. In addition, for the subset of cells $C' \in C(\sigma_i)$ that have some overlap with other cells in said subrow, we compute how much overlap there is using variable $o(\sigma_i)$. Finally, both $a(\sigma_i)$ and $o(\sigma_i)$ are normalized by the subrow width $w(\sigma_i)$ in line 27. This normalization ensures that the ML model is not biased by large subrows that have larger absolute values of $a(\sigma_i)$ and $o(\sigma_i)$. The last part of the algorithm creates the normalized histograms H_a and H_o in lines 30–31.

4 PHYSICAL DESIGN INTEGRATION

After training and validating the ML model proposed in Section 3, we integrated it in the circuit partitioning strategy of [6] as a use case to evaluate the ML model. As mentioned in Section 3, their work partitions the circuit using a k-d tree data structure, in a way that the leaf nodes represent the partitions, which are legalized separately. If a partition can not be legalized, it is merged with its sibling node, and the legalization proceeds to their parent node. In the end, the whole circuit legalization can be sped up, since the partitioning strategy reduces the input size of the legalization algorithm. However, doing so may degrade the solution quality, especially maximum displacement. Therefore, we modified their partitioning strategy to merge sibling nodes not only when the legalization fails, but when it results in a large displacement as well. This way, we can prune partitions that would otherwise degrade the solution quality.

There are two ways of doing this pruning strategy: (1) running the legalization algorithm and measuring the displacement of the obtained solution; (2) using the ML model to predict the outcome of the legalization without running the legalization algorithm. We compare these two solutions in order to evaluate the efficiency and impact of the ML model.

Algorithm 4 shows how we implemented the first pruning strategy. The algorithm receives as input the partitions to be legalized ($\mathcal{P}$), the legalization algorithm ($\mathcal{L}$) and the maximum displacement threshold (Δ). Then, it iterates through all partitions trying to legalize them. For each partition p_i, it runs the legalization algorithm (line 4) and measures the obtained maximum displacement δ_{max} (lines 5–9). If δ_{max} exceeds the threshold Δ or if the legalization

Algorithm 4: PARTITIONED_LEGALIZATION ($\mathcal{P}$, $\mathcal{L}$, Δ)

```
1  foreach $p_i$ in $\mathcal{P}$ do
2      $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_i\}$;
3      if $parent(p_i) \notin \mathcal{P}$ then
4          $\Lambda$, $result \leftarrow \mathcal{L}(C(p_i), R(p_i))$;
5          $\delta_{max} \leftarrow -\infty$;
6          foreach $c_i \in C(p_i)$ do
7              $\delta(c_i) \leftarrow |l(c_i) - \lambda(c_i)|$;
8              $\delta_{max} \leftarrow \max(\delta_{max}, \delta(c_i))$;
9          end
10         if $\delta_{max} > \Delta \vee \neg result$ then
11             $\mathcal{P} \leftarrow \mathcal{P} \cup \{parent(p_i)\}$;
12         else
13             foreach $c_i \in C(p_i)$ do
14                 $l(c_i) \leftarrow \lambda(c_i)$;
15             end
16         end
17     end
18 end
```

Algorithm 5: ML_PARTITIONED_LEGALIZATION ($\mathcal{P}$, $\mathcal{L}$, Δ, $\mathcal{M}$)

```
1  foreach $p_i$ in $\mathcal{P}$ do
2      $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p_i\}$;
3      if $parent(p_i) \notin \mathcal{P}$ then
4          if $\mathcal{M}(C(p_i), R(p_i))$ then
5              $\mathcal{P} \leftarrow \mathcal{P} \cup \{parent(p_i)\}$;
6          else
7              $\Lambda$, $result \leftarrow \mathcal{L}(C(p_i), R(p_i))$;
8              if $\neg result$ then
9                  $\mathcal{P} \leftarrow \mathcal{P} \cup \{parent(p_i)\}$;
10             else
11                 foreach $c_i \in C(p_i)$ do
12                     $l(c_i) \leftarrow \lambda(c_i)$;
13                 end
14             end
15         end
16     end
17 end
```

fails, the partition is ignored and its parent node is added to $\mathcal{P}$ to be legalized instead (lines 10–11). Otherwise, its cells are placed in their legal locations (lines 13–15). Observe that, by adding the parent node of p_i in $\mathcal{P}$, it is possible to avoid the legalization of the sibling of p_i as well with the verification of line 3. If the parent of p_i is already in $\mathcal{P}$, this means the sibling of p_i resulted in a large displacement or could not be legalized, so p_i should be ignored as well.

Running the legalization algorithm to prune partitions with large displacement results in several unnecessary calls to the legalization algorithm, which may take too much time. Therefore, the second pruning strategy relies on the ML model to predict when a partition should be ignored, as detailed in Algorithm 5. Observe that now the algorithm receives as input the machine learning model $\mathcal{M}$ as well, which is used in line 4 to predict when the partition can be pruned. Then, it runs the legalization algorithm $\mathcal{L}$ only if it was not pruned (line 7), and checks only if the partition was legalized (lines 8–10). If the partition was successfully legalized, it places the cells in the legal locations. In the end, the whole circuit is legalized, but requiring fewer calls to the legalization algorithm $\mathcal{L}$ than in Algorithm 4.

5 EXPERIMENTAL RESULTS

5.1 Experimental setup

This work uses the benchmarks from ICCAD 2015 and ICCAD 2017 CAD Contest [5, 13]. Table 2 presents the names and number of cells of each circuit. We divided the circuits in three groups: ***training***, ***validation*** and ***test***. The *training* and *validation* sets were obtained by randomly separating the circuits from the ICCAD 2017 CAD Contest into those groups using the holdout method for cross validation. They were used to evaluate different ML models and to select the best model for the integration described in Section 4. This way, the ML model is trained and validated using circuits with multi-row cells, which are more challenging to legalize. On the other hand, the *test* set is used along with the *validation* one for a second experiment, which evaluates the quality of the selected ML model when integrated in a circuit partitioning strategy. The *test* set is composed by the ICCAD 2015 CAD Contest benchmarks, which were not used when selecting the best ML model. Although those circuits do not contain multi-row cells, they are much larger than the others. Therefore, they provide a way to evaluate the speedup achieved by the ML model on large designs.

We performed all experiments in a Linux workstation with an Intel® Xeon® E5430 processor with 4 cores @ 2.66 GHz and 16GB DDR2 667MHz RAM. In order to identify the best ML model for our problem we evaluated three options: an artificial neural network (**ANN**) with a single hidden layer with 10 neurons, a decision tree (**DT**) and a deep convolutional neural network (**CNN**), using the resnet34 CNN architecture [8]. The first two models were prototyped using the Knime platform [14], while the CNN was prototyped using the fast.ai library [9]. After selecting the best ML model, it was retrained with the same parameters using the

Table 2: Benchmarks used in the experiments.

Benchmark	# Cells of different heights				Benchmark set	Group
	1	2	3	4		
pci_bridge32_a_md1	26K	1.7K	597	448	ICCAD17	validation
pci_bridge32_a_md2	25K	2K	1.1K	994	ICCAD17	validation
pci_bridge32_b_md1	26K	1.7K	585	439	ICCAD17	validation
pci_bridge32_b_md2	28K	292	292	292	ICCAD17	validation
pci_bridge32_b_md3	27K	292	585	585	ICCAD17	training
fft_2_md2	28K	2.1K	705	529	ICCAD17	training
fft_a_md2	27K	2K	672	504	ICCAD17	training
fft_a_md3	28K	672	672	672	ICCAD17	training
des_perf_a_md1	103K	4.6K	0	0	ICCAD17	training
des_perf_a_md2	105K	1K	1086	1K	ICCAD17	training
des_perf_1	112K	0	0	0	ICCAD17	training
des_perf_b_md1	106K	5.8K	0	0	ICCAD17	training
des_perf_b_md2	101K	6.7K	2.2K	1.6K	ICCAD17	training
edit_dist_1_md1	118K	7.9K	2.6K	1.9K	ICCAD17	training
edit_dist_a_md2	115K	7.7K	2.5K	1.9K	ICCAD17	training
edit_dist_a_md3	119K	2.5K	2.5K	2.5K	ICCAD17	training
superblue18	768M	0	0	0	ICCAD15	test
superblue4	795M	0	0	0	ICCAD15	test
superblue16	981M	0	0	0	ICCAD15	test
superblue5	1086M	0	0	0	ICCAD15	test
superblue1	1209M	0	0	0	ICCAD15	test
superblue3	1213M	0	0	0	ICCAD15	test
superblue10	1876M	0	0	0	ICCAD15	test
superblue7	1931M	0	0	0	ICCAD15	test

Table 3: Results of different ML models for different maximum displacement thresholds.

	Max disp threshold of 5 rows				Max disp threshold of 10 rows				Max disp threshold of 15 rows			
ML model	accuracy	precision	recall	F-score	accuracy	precision	recall	F-score	accuracy	precision	recall	F-score
ANN	81%	86%	72%	0.78	85%	78%	73%	0.75	84%	69%	72%	0.70
DT	76%	83%	63%	0.71	82%	76%	58%	0.66	83%	71%	63%	0.67
CNN	82%	84%	83%	0.83	82%	72%	83%	0.77	80%	70%	66%	0.68

Table 4: Results when integrating the ANN model to the partitioning strategy

Design	Max disp threshold of 5 row						Max disp threshold of 10 row						Max disp threshold of 15 row					
	Avg disp		Max disp		HPWL		Avg disp		Max disp		HPWL		Avg disp		Max disp		HPWL	
	LEG	ANN	LEG	ANN	LEG	ANN	LEG	ANN	LEG	ANN	LEG	ANN	LEG	ANN	LEG	ANN	LEG	ANN
pci_bridge32_a_md2	0.92	0.92	0.77	0.77	0.95	0.95	0.92	0.92	0.77	0.77	0.95	0.95	0.92	0.92	0.77	0.77	0.95	0.95
pci_bridge32_b_md1	0.69	0.69	0.30	0.30	0.99	0.96	0.69	0.69	0.30	0.30	0.99	0.96	0.69	1.01	0.30	0.91	0.99	1.00
pci_bridge32_b_md2	0.90	0.91	0.33	0.33	0.98	0.98	0.90	0.91	0.33	0.33	0.98	0.98	0.90	0.87	0.33	0.45	0.98	0.98
pci_bridge32_a_md1	0.89	0.89	0.95	0.95	0.98	0.98	0.89	0.89	0.95	0.95	0.98	0.98	0.89	0.89	0.95	0.95	0.98	0.98
superblue10	1.01	1.01	0.21	0.21	1.00	1.00	1.01	1.01	0.21	0.21	1.00	1.00	1.01	1.01	0.21	0.21	1.00	1.00
superblue18	0.96	0.96	0.11	0.11	1.00	1.00	0.96	0.92	0.11	0.11	1.00	1.00	0.96	0.92	0.11	0.24	1.00	1.00
superblue4	0.87	0.87	0.17	0.17	1.00	1.00	0.87	0.87	0.17	0.17	1.00	1.00	0.87	0.85	0.17	0.17	1.00	0.99
superblue7	1.04	1.04	0.13	0.13	1.00	1.00	1.04	1.04	0.13	0.13	1.00	1.00	1.04	1.04	0.13	0.13	1.00	1.00
superblue1	1.08	1.08	0.06	0.06	1.00	1.00	1.08	1.08	0.06	0.06	1.00	1.00	1.08	1.08	0.06	0.06	1.00	1.00
superblue16	1.06	1.06	0.52	0.52	1.00	1.00	1.06	1.06	0.52	0.52	1.00	1.00	1.06	1.06	0.52	0.52	1.00	1.00
superblue3	1.01	1.01	0.18	0.18	1.00	1.00	1.01	1.01	0.18	0.18	1.00	1.00	1.01	0.94	0.18	0.28	1.00	1.00
superblue5	1.05	1.05	0.12	0.12	1.00	1.00	1.05	1.05	0.12	0.12	1.00	1.00	1.05	0.96	0.12	0.34	1.00	1.00
Average	**0.96**	**0.96**	**0.32**	**0.32**	**0.99**	**0.99**	**0.96**	**0.96**	**0.32**	**0.32**	**0.99**	**0.99**	**0.96**	**0.96**	**0.32**	**0.42**	**0.99**	**0.99**
Median	**0.98**	**0.98**	**0.19**	**0.19**	**1.00**	**1.00**	**0.98**	**0.96**	**0.19**	**0.19**	**1.00**	**1.00**	**0.98**	**0.95**	**0.19**	**0.31**	**1.00**	**1.00**

Keras framework [12], so that we could integrate the model in the C++ code of the circuit partitioning strategy. All experiments are available under public domain, so that they can be reproduced [1].

5.2 Evaluation of machine learning models

Table 3 shows the results of the ML models on the *validation* set for three different maximum displacement thresholds (Δ): 5, 10, and 15 rows. We evaluated the models by their accuracy, precision, recall and F-score[3]. We selected the best results obtained for each model. Since the number of samples of each class (true positive, false positive, etc.) is not necessarily balanced, it is important to evaluate not only the accuracy of the model, but also the precision and recall. A low precision means a model assumes that some partitions would result in a large displacement when that is not the case, resulting in worse execution times as it takes longer to legalize the larger parent partition. On the other hand, a low recall means that the model will not prune some partitions that should have been pruned. This will result in quality degradation, since Algorithm 5 will accept the legalization even though it should not.

When the maximum displacement threshold is 5 rows, ANN and CNN are very similar in accuracy, with CNN achieving a better F-score. The DT model, on the other hand, seems to be the worst of the three models, with lower accuracy, precision, and recall. When we increased the maximum displacement threshold to 10 rows, there is a slight increase on the accuracy of the DT and ANN models, and no difference for the CNN model. However, there was a reduction on the precision for all models, due to the more unbalanced data for this displacement threshold. The recall was only affected on DT, which again makes it the worst of the three models. Finally, when the maximum displacement threshold is 15 rows, the data is even more unbalanced, with less positive instances. In this case, there was an F-score reduction on both ANN and CNN models. The impact was greater on the CNN, whose F-score dropped from 0.83 (with threshold of 5 rows) to 0.68 (with threshold of 15 rows).

These results led us to the following conclusions: (1) DT has a worse F-score for all maximum displacement thresholds, which makes it a poor model for the analyzed problem; (2) increasing the maximum displacement threshold degrades the quality of all models, due to more unbalanced data. Thus, an evaluation with even higher thresholds would require the generation of data with more positive instances.

5.3 Integration with circuit partitioning

Based on the evaluation of the ML models, we selected the ANN model to be integrated in the circuit partitioning strategy. Although the CNN achieved a higher accuracy and F-score for the maximum displacement threshold of 5 rows, we selected ANN over CNN for the following reasons: (1) the ANN model achieved a higher accuracy and F-score for the highest displacement threshold (15 rows), which suggests that this model is more robust to unbalanced data; (2) the CNN model is more complex, and therefore it takes longer to classify partitions using it. In order for this increased

[3] F-score is calculated by the harmonic mean of precision and recall.

run time to be compensated, the CNN model must have a much higher accuracy than ANN, which is not the case for the evaluated models; (3) further experiments showed that the CNN model has lower accuracy on smaller partitions than on larger partitions. Since smaller partitions are the majority in the partitioning strategy, it is better to have a higher accuracy for them.

Table 4 shows the results of applying the pruning strategy presented in Section 4 on the *validation* and *test* circuits with relation to average displacement, maximum displacement and half-perimeter wirelength (HPWL)[4]. Observe that none of the circuits presented in this table were used for the model training. For each metric we present the ratio of the result when using the pruning strategy by the original result from [6]. This way, a ratio lower than 1 for a given metric means that its value was reduced (and improved) when using the pruning strategy. For each metric, the table shows the results of two versions of the pruning strategy: LEG (Algorithm 4), and ANN (Algorithm 5 using the ANN model). In addition, the table is divided in the same maximum displacement thresholds as Table 3 (5, 10 and 15 rows).

First of all, observe that for LEG, the results remain the same for all metrics even when increasing the threshold from 5 to 15 rows. The ratios were on average 0.96, 0.32 and 0.99 for average displacement, maximum displacement and HPWL, respectively. This means that increasing the maximum displacement threshold does not make a significant difference for those circuits. The improvement was especially high for the maximum displacement, since this is the metric being verified by the pruning strategy. In addition, there was a significant reduction on the average displacement of circuits *pci_bridge32_a_md2*, *pci_bridge32_b_md1*, *pci_bridge32_b_md2*, *pci_bridge32_a_md1*, *superblue18* and *superblue4*, which are the smallest circuits used on this experiment. This happened because, for larger circuits, legalizing the parent nodes in the partitioning strategy may increase the average displacement, since they have a larger area and, therefore, allow more cell movement. However, the increase on average displacement is largely compensated by the reduction on maximum displacement.

When using the ANN model with a maximum displacement threshold of 5 or 10 rows, the results were the same as LEG for almost all circuits, which means the accuracy of the ML model was enough to avoid degrading the solution quality. However, when increasing the maximum displacement threshold to 15 rows, the lower precision of the ANN model results in degradation of the max displacement metric, whose average ratio becomes 0.42. However, this value is still much lower than the results from [6].

Besides analyzing the solution quality, it is important to also analyze how many calls to the legalization algorithm were avoided by using ANN, and what was the impact on the execution time. Figure 2 shows the ratio of the number of calls to the legalization algorithm performed by ANN compared to LEG. For all cases, the ANN model reduced the number of legalization calls, with a greater reduction in the larger circuits. This happens because the larger number of cells in these circuits increases the probability of a cell resulting in a large displacement, requiring more partition merges. In addition, this reduction is greater with a threshold of 5 rows because it requires more merges, which increases the number of legalization calls for LEG. However, even in the worst case, ANN required 22% less calls to the legalization algorithm (*pci_bridge32_a_md1*, 15 rows), but up to 99% in the best case (*superblue7*, 5 rows). This reduction on number of legalization calls resulted in the speedup presented in Figure 3. This figure shows the speedup only for ICCAD 2015 CAD Contest benchmarks, since the speedup is negligible for the other circuits due to their small size. The speedup was calculated as the ratio of the execution time of LEG by the execution time of ANN, so a speedup greater than 1 means ANN is faster.

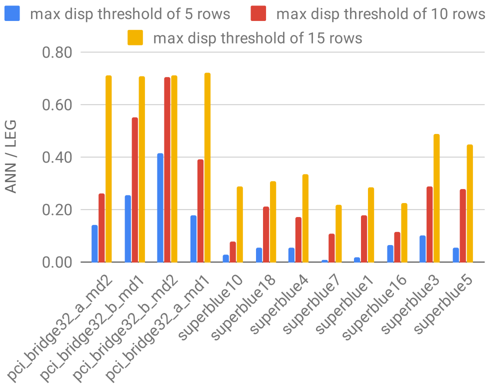


Figure 2: Comparison between the number of calls to the legalization algorithm done by LEG and ANN.

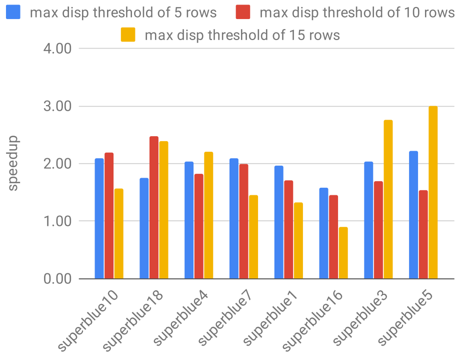


Figure 3: Comparison between LEG and ANN's execution times (speedup = ratio LEG/ANN).

The results in Figure 3 show that ANN is faster for all cases, except for *superblue16* with the maximum displacement threshold of 15 rows, achieving speedups from 0.86 to 3. In addition, for most cases, the speedup is higher when the maximum displacement threshold is lower because, as observed in Figure 2, in those cases the reduction in the number of legalization calls is greater. However, there are some exceptions, such as *superblue18* for thresholds of 10 and 15 rows, as well as *superblue3* and *superblue5* for thresholds of 15 rows. These cases are also situations where the solution quality has changed when increasing the maximum displacement threshold (Table 4). In these cases, ANN failed to identify some partitions with large displacement, which reduces the execution time but can degrade the solution quality.

[4] Although circuit *des_perf_b_md1* belongs to the *validation* set, it was not used in this experiment because the legalization algorithm was not capable of legalizing this circuit.

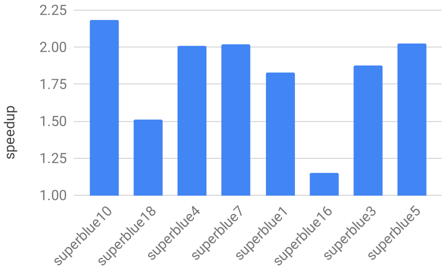


Figure 4: Comparison of the execution times between LEG with threshold of 15 rows and ANN with threshold of 5 rows (speedup = ratio LEG/ANN).

Finally, although ANN degrades the quality of the pruning for some circuits when using larger displacement thresholds, we observed that the pruning quality is the same for all thresholds when using LEG. Therefore, for the evaluated circuits, increasing the maximum displacement threshold does not improve the solution quality, but only reduces the execution time. Figure 4 shows the speedup achieved by the ANN results for the threshold of 5 rows (which achieved the best solution quality for ANN) compared to the results of LEG for the threshold of 15 rows (which has the smallest execution times for LEG). In this figure, it is possible to see that the best ANN solution is still faster than the best LEG solution by at least 1.14 and up to 2.3, which shows that ANN can effectively speed up the pruning strategy without compromising the solution quality. The speedup of at least 2 in some cases means that, for some circuits, it is possible to evaluate two different maximum displacement thresholds using ANN in the time that only one could be evaluated with LEG.

6 CONCLUSIONS

In this work we evaluated how ML models can be used to improve the predictability of legalization algorithms. We evaluated three models, including one deep convolutional neural network. The best model was integrated into a circuit partitioning strategy, to act as a pruning mechanism to identify partitions that will result in large displacement after legalization. This pruning strategy greatly reduces the maximum displacement of the legalized solution, and the ML model accelerates this process by avoiding up to 99% of the calls to the legalization algorithm.

As future work, we intend to investigate the possibility of using ML models not only to predict when a partition will violate a given displacement threshold, but to estimate the resulting displacement itself. This can be used to guide incremental placement algorithms, by using the ML model to evaluate the quality of different optimizations (or different legalization algorithms), without requiring to call the legalization algorithm.

7 ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and by the Brazilian Council for Scientific and Technological Development (CNPq) through Project Universal (457174/2014-5) and PQ grants 310341/ 2015-9.

REFERENCES

[1] Ophidian: an open source library for physical design research and teaching. https://gitlab.com/renan.o.netto/ophidian-research.

[2] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena. Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pages 15–21. ACM, 2017.

[3] J. Chen, Z. Zhu, W. Zhu, and Y.-W. Chang. Toward optimal legalization for mixed-cell-height circuit designs. In *DAC*, page 52. ACM, 2017.

[4] W.-K. Chow, C.-W. Pui, and E. F. Young. Legalization algorithm for multiple-row height standard cell design. In *DAC*, pages 1–6. IEEE, 2016.

[5] N. K. Darav, I. Bustany, A. Kennings, and R. Mamidi. Iccad-2017 cad contest in multi-deck standard cell legalization and benchmarks. In *ICCAD*, 2017.

[6] S. Fabre, J. L. Güntzel, L. L. Pilla, R. Netto, T. Fontana, and V. Livramento. Enhancing multi-threaded legalization through kd tree circuit partitioning. In *Symposium on Integrated Circuits and Systems Design*, 2018.

[7] S.-S. Han, A. B. Kahng, S. Nath, and A. S. Vydyanathan. A deep learning methodology to proliferate golden signoff timing. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 260. European Design and Automation Association, 2014.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] J. Howard and R. Thomas. fast.ai - Making neural networks uncool again. http://www.fast.ai/, 2018. [Online; accessed 28-September-2018].

[10] C.-Y. Hung, P.-Y. Chou, and W.-K. Mak. Mixed-cell-height standard cell placement legalization. In *GLSVLSI*, pages 149–154. ACM, 2017.

[11] A. B. Kahng, B. Lin, and S. Nath. High-dimensional metamodeling for prediction of clock tree synthesis outcomes. In *System Level Interconnect Prediction (SLIP), 2013 ACM/IEEE International Workshop on*, pages 1–7. IEEE, 2013.

[12] Keras. Keras: The Python Deep Learning library. https://keras.io/, 2018. [Online; accessed 28-September-2018].

[13] M. Kim, J. Hu, J. Li, and N. Viswanathan. ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite. 2015.

[14] Knime. Knime - open for innovation. https://www.knime.com/, 2018. [Online; accessed 28-September-2018].

[15] H. Li, W.-K. Chow, G. Chen, E. F. Young, and B. Yu. Routability-driven and fence-aware legalization for mixed-cell-height circuits. In *Proceedings of the 55th Annual Design Automation Conference*, page 150. ACM, 2018.

[16] P. Spindler, U. Schlichtmann, and F. M. Johannes. Abacus: fast legalization of standard cell circuits with minimal movement. In *ISPD*, pages 47–53. ACM, 2008.

[17] A. F. Tabrizi, L. Rakai, N. K. Darav, I. Bustany, L. Behjat, S. Xu, and A. Kennings. A machine learning framework to identify detailed routing short violations from a placed netlist. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[18] C.-H. Wang, Y.-Y. Wu, J. Chen, Y.-W. Chang, S.-Y. Kuo, W. Zhu, and G. Fan. An effective legalization algorithm for mixed-cell-height standard cells. In *ASP-DAC*, pages 450–455. IEEE, 2017.

[19] Q. Zhou, X. Wang, Z. Qi, Z. Chen, Q. Zhou, and Y. Cai. An accurate detailed routing routability prediction model in placement. In *Quality Electronic Design (ASQED), 2015 6th Asia Symposium on*, pages 119–122. IEEE, 2015.